Réseaux et protocoles

INF 586

Contenu

NB. Veuillez envoyer vos remarques et commentaires sur ces notes à dabbous@sophia.inria.fr

Ces notes sont tirées en grande partie du Keshav, avec des modifications à plusieurs endroits.

Bloc 1

Le réseau téléphonique et l'Intenet

1.0 Le réseau téléphonique

Le réseau téléphonique est peut être le plus grand réseau informatique du monde, mais il n'est pas identifié pas en tant que tel parce qu'il est spécialisé pour supporter la voix. En plus de la voix, le réseau supporte également la vidéo, le fac-similé, et les données de télémétrie. Plus qu'un milliard de téléphones relient ensemble un quart de la population du monde - de l'Argentine à Brunei, et du Yémen jusqu'en Zambie. Bien que le service téléphonique existe depuis plus de cent ans, il continue à se développer rapidement en raison d'innovations telles que le téléphone cellulaire, les pagers, et les modems. Aux Etats-Unis, la compagnie de téléphone AT&T, traite presque 200 millions d'appels chaque jour. Dans la complexité de son interconnexion, le degré de support organisationnel, et l'importance économique, le réseau téléphonique dépasse actuellement l'Internet. Il convient donc d'y apporter une étude soigneuse.

1.1 Concepts

Le réseau téléphonique offre un service de base simple à ses utilisateurs : transfert de voix bidirectionnel et commuté, avec un faible délai de bout en bout et la garantie qu'un appel, une fois accepté, sera maintenu jusqu'à la fin de la communication. Cette qualité de service est garantie par l'établissement d'un *circuit* entre les deux "systèmes finaux". Un circuit est un chemin d'une source à une destination similaire à un circuit électrique, supportant des signaux simultanément dans les deux directions (nous appelons ceci la communication *full-duplex*). Le réseau garantit assez de ressources à chaque circuit pour garantir la qualité de service.

Le réseau téléphonique se compose de *systèmes finaux*, en général des téléphones ordinaires, mais également télécopieurs ou modems, reliés par deux niveaux des commutateurs (voir la figure B1T11[1]). Une paire de fils relie chaque système final à un central téléphonique (Central Office ou CO). Cette partie du réseau s'appelle la boucle locale. Le central téléphonique fournit la puissance pour le fonctionnement du système final. Il reçoit les chiffres composés et établit les appels pour le système final. Il numérise également les signaux entrants (typiquement la voix), et les envoie à travers le réseau à un système final distant. Du point de vue du réseau, un système final est un dispositif sourd-muet, et le réseau fournit toute l'intelligence. Naturellement, avec les ordinateurs équipés de modem et les télécopieurs contrôlés par microprocesseur, les systèmes finaux deviennent de plus en plus intelligents. Cependant, ceci n'a pas sensiblement affecté l'architecture du réseau téléphonique.

Le schéma B1T11 montre une version fortement simplifiée d'un réseau téléphonique typique. Les systèmes finaux sont reliés aux commutateurs dans les centraux téléphoniques d'accès, et ces commutateurs sont reliés aux commutateurs d'épine dorsale ou de cœur de réseau. Le nombre de commutateurs d'épine dorsale est beaucoup plus petit que le nombre de commutateurs d'accès. Par exemple, aux Etats-Unis, il y a environ dix mille commutateurs d'accès, mais moins de cinq cents commutateurs d'épine dorsale. Le cœur du réseau est presque entièrement relié, c.-à-d., chaque commutateur sur l'épine dorsale est directement lié à tous les autres commutateurs. Quand un commutateur d'accès reçoit un appel vers un système final relié au même commutateur (par exemple, un appel de A vers B dans la figure B1T11), il établit un chemin local de la source vers le système final destinataire. Si l'appel n'est pas local (par exemple, de A vers C ou de A vers D), il relaie l'appel à un autre commutateur local (A vers C), ou au commutateur de cœur du réseau le plus proche (A vers D). Ce commutateur relaie à son tour l'appel vers commutateur le plus proche du commutateur d'accès du destinataire. Comme les commutateurs d'épine dorsale sont presque tous connectés directement l'un à l'autre, le choix de ce chemin est facile. Dans certains réseaux téléphoniques, si le chemin direct est encombré, l'appel est détourné à un chemin alternatif de deux "bonds". Le commutateur d'accès relie alors le système final destination à l'initiateur de l'appel.

Considérez un appel fait à partir d'un système final avec le téléphone ayant le numéro 01 69 88 88 88 à un système final ayant le numéro 04 92 38 38 38. A partir du numéro de l'appelant, nous savons qu'il a comme indicatif régional 01. Quand il compose le numéro 04, son commutateur d'accès local note que l'indicatif régional (04) est différent du sien (01). Ainsi, il fait suivre l'appel au commutateur d'épine dorsale le plus proche. Au niveau du cœur du réseau, le réseau établit une liaison directe ou via deux bonds au commutateur desservant la zone ayant le préfix 04, et de là vers le commutateur d'accès desservant les numéros en 92. Finalement, une connexion est établie au système final 383838 dans le commutateur 92. Ainsi, les numéros de téléphone permettent de prendre des décisions de routage simples.

---

[1] La notation pour les figures sera la suivante : BxTy signifie voir le transparent y du Bloc x.

Ceci est une image simplifiée de l'opération du réseau téléphonique. Dans la pratique, cent ans d'évolution ont compliqué sensiblement l'image. Par exemple, quelques centraux téléphoniques transmettent toujours de la voix analogique aux commutateurs d'épine dorsale, de sorte que des commutateurs d'épine dorsale doivent être équipés pour numériser ces appels entrants. La complexité surgit également quand le même commutateur d'épine dorsale sert des indicatifs régionaux multiples, ou quand un indicatif régional est supporté par plusieurs commutateurs de cœur de réseau. Plus encore, des réseaux d'épine dorsale concurrents peuvent relier entre eux les commutateurs locaux. Pris ensemble, ces détails compliquent ce qui est, à l'origine, une technologie simple et robuste.

Les numéros de téléphone forment une hiérarchie, permettant l'attribution distribuée de nouveaux numéros de téléphone, globalement uniques. Une autorité centrale attribue à chaque région ou à chaque ville indicatif différent, constitué des premiers chiffres du numéro de téléphone, et chaque région attribue des nombres uniques à ses commutateurs d'accès. Chaque commutateur d'accès s'assure à son tour que les systèmes finaux gérés par ce commutateur ont des numéros uniques. Ainsi, nous pouvons facilement nous assurer qu'un numéro de téléphone est globalement unique, ce qui est essentiel pour acheminer les appels et pour la facturation.

Le numéro de téléphone du destinataire fournit des éléments pour connaître l'itinéraire. Bien qu'il puisse y avoir plus qu'un chemin pour atteindre un système final particulier, le numéro de téléphone du destinataire identifie les commutateurs du réseau téléphonique que l'itinéraire *doit* contenir. Par exemple, l'itinéraire vers le téléphone le numéro 04 92 38 38 38 doit passer par le commutateur d'épine dorsale desservant l'indicatif régional 04 et le commutateur d'accès 92, mais il peut également passer par d'autres commutateurs d'épine dorsale et locaux. Pour des indicatifs spéciaux, tels que 700, 800, 888, et 900 aux Etats-Unis, le numéro composé est employé en tant qu'index pour interroger une base de données qui traduit le numéro en numéro de téléphone normal. Le fournisseur de service peut changer la traduction selon la source de l'appel, l'heure, ou d'autres propriétés. Cette flexibilité permet aux usagers du téléphone d'accéder à une grande variété de services en utilisant ces indicatifs spéciaux.

Nous étudierons brièvement quatre composants du réseau téléphonique : les systèmes finaux, la transmission, la commutation, et la signalisation.

1.2 Systèmes finaux

Un système final connecté à un réseau téléphonique est en général un téléphone qui se compose de cinq parties :

- Un convertisseur voix-à-signal-électrique,
- Un convertisseur signal-électrique-à-voix,
- Un dispositif de composition,
- Une sonnerie,
- Un connecteur de commutation

Nous décrivons ces parties en plus détail.

1.2.1 Convertiseurs de signal

Alexandre Graham Bell a inventé les premiers convertisseurs de voix-à-signal-électrique et de signal-électrique-à-voix en 1875 - 1876. Un de ses premiers convertisseurs voix-à-signal-électrique était une membrane étirable fixée à une borne en métal A plongée dans une fiole contenant un liquide conducteur B. La borne en métal a été reliée à une borne d'une batterie, et l'autre borne de la batterie a été reliée à la fiole. Les vibrations de voix modulent l'impédance du circuit entre A et B, ce qui cause une courant variable dans le circuit.

Le convertisseur signal-électrique-à-voix est un électro-aimant D qui fait vibrer une petite tige en métal E. Les principes derrière ces convertisseurs ont peu changé avec le temps. Les convertisseurs modernes de voix-à-signal-électrique utilisent des granules de carbone ou un microphone *electret* (miniaturisable, pas très sensible) pour réaliser une résistance variable et ainsi une tension variable. Au niveau du récepteur, le signal électrique alimente un petit haut-parleur plus ou moins de la même façon que le récepteur de Bell d'il y a cent ans.

1.2.2 Sidetones et échos

Pour permettre la communication bidirectionnelle, les circuits de transmission et de réception des systèmes finaux communiquants doivent être reliés. Chaque circuit exige deux fils, ainsi la manière évidente de relier un téléphone au

central téléphonique est avec quatre fils : une paire pour chaque direction. Une solution moins onéreuse, consiste à faire partager la même paire de fils aux deux circuits, de sorte que les signaux transmis et reçus soient superposés. Ceci présente deux problèmes. D'abord, à un système final, le signal électrique du convertisseur de voix-à-signal-électrique circule directement dans le circuit de réception. Ainsi, tout ce qui est dit est entendu fortement par le récepteur. Ceci est appellé le signal sidetone. Bien qu'il soit important que la personne entende sa propre voix dans l'écouteur, il est nécessaire d'atténuer le signal à l'aide d'un circuit d'équilibrage, pour éviter d'assourdir la personne qui parle au téléphone.

Le deuxième problème avec le fait d'employer seulement deux fils entre un système final et un central téléphonique est que ceci provoque un écho dans le circuit. ◦ chaque téléphone, le signal reçu revient au central téléphonique via le circuit de transmission circuit et sont entendus comme écho à l'autre extrémité. Pour des appels locaux, la présence d'un écho n'est pas un problème, parce que l'oreille humaine ne peut pas distinguer un sidetone et l'écho. Cependant, pour des appels internationaux (de longue distance), si le délai est plus de 20 *ms* environ, un écho fort peut être très gênant et doit être atténué ou éliminé. Ceci exige un circuit d'annulation d'écho assez onéreux qui surveille activement le signal transmis et reçu. Puisque le réseau doit fournir un annulateur d'écho pour chaque ligne longue distance, ceci peut engendrer une dépense d'exploitation significative. Pour éviter le coût de annulateurs d'écho, les compagnies de téléphone essayent de réduire le délai dans la mesure du possible. Nous verrons plus tard dans ce chapitre, que les calculs des délais constituaient un facteur déterminant pour le choix des paramètres essentiels des réseaux ATM.

La plupart des raccordements entre un système final et un central téléphonique sont des circuits à deux fils, mais les raccordements entre les centraux téléphoniques sont toujours à quatre fils (avec la transmission et la réception séparées). Le central téléphonique effectue la conversion deux fils à quatre fils, et l'annulation d'écho est effectuée par les commutateurs d'épine dorsale.

1.2.3 Composer et sonner

En plus de la conversion voix à signal électrique et vice versa, le système final fait les opérations, relativement simples, de composer et de sonner. Le dispositif de composition de numéros envoie une série de fréquences vocales ou d'impulsions au central téléphonique. Un ordinateur à usage spécifique appelé "système de commutation" les interprète pour établir un appel ou pour activer des fonctions spécifiques telles que le renvoi d'appel. Le système de commutation envoie également un signal pour activer la sonnerie au téléphone quand il y a un appel entrant. Le central téléphonique assure la puissance pour activer la sonnerie.

1.3 Transmission

Le réseau téléphonique transporte des signaux produits par un téléphone au-dessus de liens de transmission. Dans cette section, nous discuterons de certaines métriques pour décrire ces liens. Nous étudierons aussi comment des signaux peuvent être multiplexés pour réduire le coût par bit.

1.3.1 Caractéristiques de lien

Un lien de transmission peut être caractérisé par sa *capacité*, le *délai* de propagation, et à quelle distance il peut transporter un signal avant que le signal s'affaiblit de façon à ce qu'il ne puisse plus être identifié (*atténuation* du lien).

La largeur de bande ou *bande passante* d'un lien mesure sa capacité. En utilisant une analogie de tuyauterie, c'est la largeur du "tuyau de l'information". Un lien supporte l'information sous forme de symboles analogiques, où un symbole peut correspondre à un ou plusieurs bit. Ainsi, le *débit binaire* d'une ligne, mesuré en bits par seconde, est le produit de sa capacité mesurée en symboles par seconde et du nombre moyen de bit par symbole. Les ingénieurs réseau tendent à confondre les notions de bande passante, de capacité, et de débit binaire, en les utilisant l'un à la place de l'autre par abus de langage. Conformément à cette tradition, nous utiliserons ci-après les termes bande passante et capacité sans distinction (sauf dans la partie concernant la couche physique du chapitre 2, où nous donnons dans le prochain chapitre une définition plus détaillée de ces notions.

Le délai du lien est le temps mis par un signal pour se propager au-dessus du milieu. Ce délai est significatif seulement pour des liens ayant une longueur plus grande que quelques dizaines de kilomètres. Par exemple, la lumière voyage dans la fibre optique à 0.7 fois la vitesse de la lumière dans le vide, correspondant à un délai de propagation d'environ 5 µs/km (~8 µs/mile). Pour donner un exemple, notez que la conversation interactive de voix exige un délai de bout en bout en dessous de la limite de 100 ms. Le délai de propagation entre New York et San Francisco (une distance d'environ 2500 milles), est de 20 ms environ, et le délai entre New York et Londres (une distance d'environ 3500 milles), est de 27 ms environ. Donc sur ces liens, une fraction substantielle du délai de bout en bout autorisé est prise par le délai du propagation du signal dû à la vitesse de la lumière. Le délai est un problème important dans le cas de la transmission par satellite géostationnaire, où le délai propagation de station-à-station est autour de 250 ms ce qui rend les conversations téléphoniques au-dessus d'un lien satellite parfois exaspérantes !

◦ mesure que la longueur d'un lien augmente, la qualité du signal transporté se dégrade. Pour maintenir la capacité du lien, le signal doit être régénéré après une certaine distance. Les régénérateurs sont chers et difficiles à maintenir parce qu'ils ont besoin d'une alimentation d'énergie, parce qu'ils comportent des circuits électroniques actifs qui peuvent tomber en panne, et parce qu'ils sont souvent au milieu de nulle part. Nous préférons donc une technologie de lien qui nécessite le moindre nombre de régénérateurs. Les avancés récentes en technologie de *fibre optique* ont permis d'établir des liens qui ont besoin de régénération seulement une fois tous les 5000 kilomètres environ. D'ailleurs, avec *l'amplification optique*, nous pouvons amplifier un signal optique sans le convertir en forme électronique. Ceci permet d'employer le même lien optique pour transporter des signaux de diverses sortes sans devoir changer tous les régénérateurs le long du chemin. ◦ l'avenir, ces deux technologies pourraient rendre les régénérateurs électroniques désuets.

1.3.2 Multiplexage

La paire de fils entre une maison et un central téléphonique transporte une conversation de voix. Cependant, les "câbles" entre les commutateurs d'épine dorsale doivent transporter des centaines de milliers de conversations. La pose des centaines de milliers de paires de fil entre ces commutateurs n'est pas envisageable. Au lieu de cela, des conversations sont groupées ou multiplexées ensemble avant transmission.

Les réseaux téléphoniques utilisent des multiplexeurs analogiques et numériques. Les multiplexeurs analogiques fonctionnent dans le domaine de fréquence : la voix analogique est filtré en passe-bas à 3.4 KHz puis décalé par une porteuse de sorte que chaque conversation soit dans une partie du spectre sans recouvrement avec d'autres conversation. Cette technologie devient rapidement désuète, car le réseau téléphonique devient tout numérique.

Les réseaux téléphoniques numériques convertissent la voix en signal numérique au niveau du téléphone ou du central téléphonique, et le reste du réseau transporte uniquement des signaux numériques. L'idée principale dans la numérisation de la voix est de comparer l'amplitude d'un signal de voix à un ensemble relativement réduits de niveaux d'amplitude numérotés, appelés des niveaux de quantification. Une amplitude de voix sera représentée numériquement par le numéro du niveau de quantification le plus proche. Si les niveaux de quantification sont correctement choisis, l'erreur de quantification est réduite, et le signal analogique régénéré est acceptable. Le schéma standard de numérisation consiste à échantillonner le signal de voix 8000 fois par seconde, comparant l'amplitude à 256 niveaux de quantification, menant à des échantillons de 8 bits. Ceci a comme conséquence une largeur de bande standard de 64 Kbps pour la voix numérisée selon le standard PCM (Pulse Code Modulation). Typiquement, les niveaux de quantification sont espacés selon une échelle logarithmique, puisque ceci donne une meilleure résolution aux niveaux de signal faibles. Deux choix des niveaux de quantification sont d'usage courant, appelés la loi μ (aux Etats-Unis et au Japon) et la loi A (dans le reste du monde). Malheureusement, ceci signifie que les appels les plus internationaux doivent être ré-échelonnés.

Le multiplexage temporel (TDM ou Time Division Multiplexing) est employé pour combiner les flots de voix numérisée à 64-Kbps pour former de flots de débit plus élevé. Un multiplexeur temporel synchrone (voir B1T18) dispose de *n* liens identiques en entrée et d'un lien de sortie qui est au moins *n* fois plus rapide que les liens d'entrée. Le multiplexeur enregistre les bits entrants en tampons mémoire et les place, à tour de rôle, sur la ligne de sortie. Une *trame* est donc constituée de *n* échantillons en sortie, avec des bits de surcharge constituant l'en-tête. Un récepteur à l'autre extrémité d'une ligne multiplexée peut extraire chaque échantillon des flots constitutifs en se synchronisant par rapport aux frontières de trame.

Des liens multiplexés peuvent être multiplexés davantage. L'ensemble standard des débits des liens multiplexés s'appelle la hiérarchie de signalisation numérique (aux Etats-Unis et au Japon) (voir le tableau B1T20). Une hiérarchie semblable est l'objet d'une norme internationale (comme dans beaucoup d'autres choses, ce qui pratiqué aux USA diffère légèrement des normes internationales).

1.4 Commutation

Le troisième composant important du réseau téléphonique c'est la commutation. ◦ la différence du réseau de télévision, qui fournit une communication de diffusion (1 vers N), le réseau téléphonique fournit la plupart du temps une communication "point à point".

Comment un utilisateur devrait-il être mis en contact n'importe quel autre utilisateur ? Une solution naïve, irréalisable en pratique, est de lier physiquement chaque utilisateur à tous les autres utilisateurs (imaginez un milliard de fils arriver chez vous !). Au lieu de cela, une paire de fils relie chaque utilisateur à un commutateur, qui est relié à son tour à d'autres commutateurs. Quand un appel est établi vers un utilisateur, les commutateurs établissent une connexion provisoire ou *circuit* entre l'appelant et l'appelé.

Un commutateur téléphonique est composé de deux parties (voir figure B1T22) : la matrice de commutation transporte

la voix et le contrôleur du commutateur manipule les demandes d'établissement et de résiliation des circuits. Quand un utilisateur compose un numéro de téléphone, les contrôleurs de commutateurs établissent un circuit de transfert de données de source à la destination. Pendant l'appel, les échantillons de voix traversent les commutateurs sans passer par les contrôleurs de commutateur. Les contrôleurs forment ainsi un réseau de recouvrement appelé le réseau de signalisation. Nous allons évoqué la signalisation dans la section 1.5.

Un commutateur transfère des données arrivant sur une entrée vers une sortie. Ceci peut être compliqué parce qu'un grand commutateur de central téléphonique peut avoir plus de 150.000 entrées et sorties. Les deux manières de base de relier des entrées aux sorties sont la commutation spatiale et la commutation temporelle. Dans cette section, nous étudions des commutateurs spatiaux et temporels simples et les combinaisons permettant d'obtenir de grands commutateurs à partir de ces éléments de base.

Le commutateur spatial le plus simple est une matrice de commutation appelée *crossbar* (figure B1T23). Dans un crossbar, les entrées arrivent le long des lignes de la matrice de commutation, et les sorties sont reliées aux colonnes. Pour relier n'importe quelle entrée à n'importe quelle sortie, le contrôleur de commutateur établit une connexion au point où la ligne d'entrée rencontre la colonne de sortie. Les autres croisements ne sont pas connectés. De cette façon, à un moment donné, le commutateur peut relier n'importe quelle entrée à n'importe quelle sortie. Par exemple, dans la figure B1T23, l'entrée B a été relié à la sortie 2, l'entrée C à la sortie 3, et l'entrée E à la sortie 4. Si les lignes entrant dans un crossbar sont multiplexées, alors les points de croisement doivent être réarrangés à intervalles réguliers (la durée de l'intervalle étant égale à la durée d'un échantillon) afin de commuter correctement les nouveaux échantillons entrants. Au moment de l'établissement d'appel, le contrôleur de commutateur détermine un *programme* qui indique au commutateur associé comment établir les points de croisement à chaque intervalle de temps d'une trame.

Un crossbar est un commutateur spatial parce que les chemins de données sont séparés dans l'espace. Une autre manière est de séparer les chemins de données dans le temps, comme dans un commutateur temporel (figure B1T24). Dans ce commutateur, les *n* entrées sont stockées dans des tampons mémoire. Le commutateur lit des tampons mémoire *n* fois plus vite que chacune des entrées (considérées ayant un débit égal) et écrit le contenu sur le lien de sortie approprié. Par exemple, pour réaliser les mêmes raccordements que sur le schéma B1T23, le commutateur lit le tampon B et écrit sur le lien de sortie 2, lit le tampon C, et écrit sur la sortie 3, et finalement lit le tampon E et écrit sur le lien 4. Il est clair que si le commutateur fonctionne *n* fois plus vite que les entrées, n'importe quelle entrée peut être dirigée vers n'importe quelle sortie. Ce type d'élément de commutation s'appelle également un échangeur d'intervalle de temps (TSI ou Time Slot Interchange). Notez la similitude étroite entre un multiplexeur temporel et un TSI.

Comme tous les échantillons de voix arrivent au même instant, la seule information dont a besoin un commutateur temporel est l'ordre dans lequel écrire sur les sorties (*le programme*). Si une entrée est à vide, le commutateur ne peut pas employer l'intervalle de temps pour servir une autre entrée, donc tout intervalle à vide est gaspillé. C'est inefficace si une conversation a de longues pauses. Nous verrons plus tard en ce chapitre, qu'en ajoutant un en-tête aux échantillons de voix, nous pouvons éviter cette inefficacité. Nous donnons dans la section ci-après des détails concernant le multiplexage et la commutation. Cette section peut être sautée dans une première lecture pour aller directement à la section 1.5.

Nous rappelons d'abord qu'un circuit de téléphone achemine des échantillons de voix de 8 bits correspondant à 125 µs de voix échantillonnée. Notez qu'un échantillon n'a pas d'en-tête identifiant sa source ou sa destination : ces informations sont inférées à partir de la ligne physique sur laquelle l'échantillon est reçu et du temps de réception sur la ligne (c.-à-d., la position de l'échantillon dans la trame). Dans cette section, nous étudions en détail comment commuter des échantillons arrivant sur une entrée d'un commutateur téléphonique à une sortie. Avant que nous étudiions en détail les commutateurs, nous allons aborder les notions de base du multiplexage et du démultiplexage.

1.4.1 Multiplexeurs et démultiplexeurs

La plupart des lignes téléphoniques transportent plus d'un circuit de voix, partageant (ou *multiplexant*) la ligne entre des échantillons provenant de ces circuits. Au niveau d'un central téléphonique, les appels arrivant des boucles locales actives sont combinés avant d'être placés sur un ligne longue distance partagée, et les appels vers des boucles locales actives sont enlevés (ou démultiplexés) des liens entrants. Les multiplexeurs et les démultiplexeurs sont donc, des éléments de base du réseau téléphonique.

Un multiplexeur synchrone est composé de *N* lignes d'entrée et d'une ligne de sortie ayant une capacité N fois plus grande que les lignes d'entrée (voir le schéma B1T18). Chaque ligne d'entrée est associée à un tampon qui peut stocker au moins un échantillon. Le multiplexeur visite chaque tampon d'entrée à tour de rôle, plaçant l'échantillon sur la ligne de sortie. Si les lignes d'entrée sont à 64 Kbps, alors l'échantillon dure 125 µs sur la ligne d'entrée, mais légèrement moins de 125/*N* µs sur la ligne de sortie (à cause des bits de frontière de trame).

Nous pouvons cascader en série des multiplexeurs afin de supporter plus de circuits de voix sur une même ligne

physique. La hiérarchie de signalisation numérique (voir B1T20) définit les débits autorisés sur une ligne multiplexée dans le réseau téléphonique. Par exemple, une ligne T1 supporte 24 circuits de voix à 1.544 Mbps, et une ligne T3 supporte 28 T1s à 44.736 Mbps.

Un démultiplexeur a une ligne d'entrée et *N* lignes de sortie ayant une capacité N fois plus faible que la ligne d'entrée. Un tampon mémoire de la taille d'un échantillon (8 bits) est associé à chaque ligne de sortie. Les échantillons entrants sont placés dans un tampon de sortie dans à tour de rôle. De là, ils sont transmis sur les lignes de sortie correspondantes. Notez que ni un multiplexeur ni un démultiplexeur n'a besoin des informations d'adressage l'information. Leur fonctionnement exige seulement l'information précise de synchronisation.

1.4.2 Les commutateurs de circuit

Un commutateur de circuit traitant N circuits de voix se compose conceptuellement de N entrées et de N sorties (dans la pratique, N peut être aussi grand que 120.000 voire plus). En réalité, le nombre de lignes physiques d'entrée et de sortie est beaucoup plus faible car ces lignes sont multiplexées. Par exemple, si les entrées sont des DS3s donc chacune porte 672 appels de voix, le nombre de lignes physiques est réduit par un facteur de 672. Dans cette section, si les entrées sont multiplexées, nous supposerons que toutes les entrées sont au même niveau dans la hiérarchie de multiplexage, et que les échantillons de voix sur toutes les entrées arrivent de façon synchrone[2]. L'objectif d'un commutateur de circuit est de transmettre un échantillon entrant à la bonne ligne de sortie et au bon endroit (time-slot) dans la trame de sortie. Avant que nous étudiions en détail les stratégies de commutation, nous distinguons les commutateurs qui relient les lignes téléphoniques individuelles, appelées les commutateurs de ligne, et les commutateurs qui relient les lignes multiplexés, appelés les commutateurs de transit. Un commutateur de ligne doit relier une entrée spécifique à *une sortie spécifique*. En revanche, un commutateur de transit devrait relier une entrée à *une des sorties* allant dans la bonne direction. Par exemple, un commutateur de transit à Chicago peut commuter quelques appels d'un DS3 (à 672 appels) du New-Jersey à un autre DS3 allant en Californie. Dans ce cas-ci, le commutateur devrait relier une ligne d'entrée à une des sorties allant en Californie.

La différence entre la commutation de ligne et la commutation de transit devient importante quand on considère la probabilité de blocage dans un commutateur. Un appel est bloqué si aucun chemin n'existe pour connecter l'entrée à la sortie. Au moment de l'établissement d'appel, les contrôleur de commutateur refusent les appels bloqués. Il y a deux types de blocage : blocage interne et blocage de sortie. Le blocage interne se produit si la sortie est disponible, mais il n'y a aucun chemin à travers le commutateur. Le blocage de sortie se produit si deux entrées concurrencent pour la même sortie, dans ce cas l'une d'entre elles est bloquée en sortie. Pour retourner à la commutation de ligne et de transit, notez que si une entrée peut être transmise sur n'importe quelle sortie d'un ensemble, alors le blocage interne et le blocage de sortie sont tous les deux réduits. Ainsi, un commutateur de transit peut réaliser la même probabilité de blocage qu'un commutateur de ligne avec moins de matériel.

Nous allons maintenant détailler les deux stratégies de base dans la commutation de circuit qui sont la commutation temporelle et la commutation spatiale. Puis, nous considérons les commutateurs à plusieurs étages qui combinent la commutation temporelle et spatiale.

1.4.2.1 La Commutation temporelle

Un échangeur d'intervalle de temps ou TSI utilisé dans la commutation temporelle, a seulement une entré et une sortie, mais ces lignes porter des échantillons provenant de N circuits multiplexés. Un TSI écrit les échantillons entrants dans un tampon de taille N octets, un octet par circuit entrant, et les lit sur les liaisons de sortie dans un ordre différent. Le fait d'avoir un ordre différent pour les échantillons sur la liaison de sortie signifie (une fois cette liaison est passée à travers un démultiplexeur), que les échantillons seront attribués aux différentes lignes de sortie. En d'autres termes, un TSI commute des échantillons en réarrangeant l'ordre des échantillons sur une ligne multiplexée.

Exemple 1.1

Considérez une compagnie qui a deux succursales A et B, chacune ayant 24 lignes téléphoniques. Nous voulons que n'importe quel téléphone dans A puisse se relier via une ligne T1 à n'importe quel téléphone à B. Nous faisons ceci en installant de petits commutateurs, appelés les centraux téléphoniques privés (PBXs), aux sites A et B. Chaque PBX a 24 lignes le reliant aux 24 téléphones locaux et un ligne T1 le reliant à l'autre PBX (voir figure B1T27a). Des échantillons provenant de chaque téléphone sont multiplexés dans un certain ordre fixe et placés sur la ligne T1. Des échantillons arrivant sur la connexion T1 sont passés à un TSI, qui a 24 tampons d'un octet chacun dans lesquels il écrit ces échantillons de façon cyclique. Le TSI échange l'ordre des échantillons avant de les envoyer à un démultiplexeur.

---

[2] En réalité, on doit lisser la gigue (variation des temps d'arrivée des échantillons) par un tampon d'entrée.

Supposez que le deuxième téléphone dans A, correspondant au deuxième échantillon dans la trame partant de A, voudrait se connecter au troisième téléphone en B. Afin d'établir cette connexion, le TSI du site B place le deuxième échantillon d'entrée dans le troisième intervalle de temps (time-slot) dans la trame de sortie (voir figure B1T27b). Quand la ligne de sortie du TSI est démultiplexée, le deuxième échantillon en provenance de A est ainsi placé sur la troisième ligne de B.

Si nous voulions établir un grand commutateur à partir d'un seul TSI, quelle taille maximale pourrions nous avoir ? La limite pour le passage à l'échelle d'un TSI est le temps mis pour lire et écrire un échantillon en mémoire. Supposez que nous avons voulu établir un commutateur aussi grand que les plus grands commutateurs disponibles actuellement, qui traitent 120.000 appels environ. Pour construire un TSI qui peut commuter 120.000 circuits, nous devons lire *et* écrire des échantillons en mémoire 120.000 fois toutes les 125 µs. Ceci exigerait que chaque opération de lecture et écriture soit effectuée en 0.5 *n*s environ, ce qui est inaccessible avec la technologie actuelle. Ainsi, construire un commutateur de cette taille avec un seul TSI n'est pas pratique. De nos jours, les temps d'accès typiques de mémoire DRAM sont de 80 *n*s environ, et les mémoires ayant un temps d'accès autour de 40 ns peuvent être achetées tout à fait à bon marché. Si nous employons une mémoire 40-ns, le temps d'accès est 80 fois plus grand que de ce que nous avons besoin pour un commutateur supportant 120.000 lignes. Ainsi la capacité de commutateur est tout au plus 120.000/80 = 1500 circuits, qui est en fait très faible ! Donc, pour établir des commutateurs de plus grande capacité, nous devons nous tourner vers d'autres techniques.

1.4.2.2 La commutation spatiale

Quand on effectue une commutation spatiale, chaque échantillon suit un chemin différent à travers le commutateur, selon sa destination. Nous illustrons ceci par un exemple.

Exemple 1.2

Supposez que nous avons voulu employer la commutation spatiale dans le PBX de l'exemple 1.1. Nous pouvons faire cela en démultiplexant d'abord la ligne T1 sur 24 lignes différentes, reliant ensuite les sorties du démultiplexeur au bon téléphone (voir figure B1T29). L'échantillon provenant du deuxième téléphone dans A apparaît à la deuxième ligne de sortie du démultiplexeur. Cette ligne est reliée, à travers un commutateur spatial, au troisième téléphone sur le site B. Les autres échantillons peuvent être simultanément commutés le long d'autres chemins à travers le commutateur spatial.

Le commutateur crossbar

Le commutateur spatial le plus simple est le crossbar (voir figure B1T30). Les échantillons arrivent sur les lignes d'entrées et sont transmis vers les sorties le long des colonnes. Des éléments actifs appelés les points de croisements sont placés entre les lignes d'entrée et de sortie. Si le croisement est activé, les données peuvent circuler de l'entrée vers la ligne de sortie ; sinon, les deux sont déconnectés. Notons qu'un crossbar n'est pas nécessairement une matrice carrée (ayant le même nombre de lignes d'entrée et de lignes de sortie). Une matrice crossbar $n \times k$ dispose de n lignes d'entrée et de k lignes de sortie. Si les lignes d'entrée ne sont pas multiplexées, un crossbar relie toujours les mêmes entrées aux mêmes sorties (pendant la durée de la conversation téléphonique). Cependant, si les lignes d'entrée sont multiplexées, et différents échantillons ont différentes destinations, un crossbar nécessitera un *programme* qui lui indique quels croisements activer pendant chaque intervalle de temps d'un échantillon. Selon ce programme, des échantillons sont transférés à partir de l'entrée choisie vers la sortie choisie.

Un commutateur crossbar a un taux de blocage interne nul (c.-à-d., aucun échantillon n'est bloqué dans le commutateur attendant la disponibilité d'une ligne de sortie). Malheureusement, un crossbar $N \times N$ nécessite N^2 points de croisement et est donc onéreux pour de grande valeur de N, telles que N = 100.000. Un deuxième problème concernant la construction de crossbar de grande taille est que le temps de distribution de la signalisation ; c'est-à-dire, le temps mis par un contrôleur pour activer/désactiver les croisements croît rapidement avec N. Troisièmement, une panne sur un seul croisement isole une ligne d'entrée et de sortie, rendant le crossbar mono-étage moins robuste aux pannes que les crossbars multi-étages que nous discutons après. Cependant, les crossbars sont une excellente solution pour construite des petits commutateurs (8×8 ou 64×64). Nous verrons ci-après, que des conceptions de commutateurs plus complexes se composent de blocs de commutations plus petits, qui sont en fait des crossbars.

Crossbars multi-étages

Une généralisation simple du crossbar nous permet d'économiser un grand nombre de points de croisements. Notez que pendant chaque intervalle de commutation (le temps pris pour transférer un échantillon à travers la matrice de commutation crossbar) seulement un croisement dans chaque ligne ou colonne est activé. Ceci est inefficace. Si nous pouvons faire en sorte qu'un élément interne puisse recevoir des échantillons de plusieurs lignes d'entrée, nous réduirons cette inefficacité. Ceci peut être réalisé en construisant des commutateurs multi-étages (voir figure B1T31). Dans ce cas, les entrées sont réparties en groupes de *n* entrées, gérés chacun par une matrice de commutation d'entrée, avec un étage

de commutation "interne" constitué d'un nombre *k* de crossbar. Plusieurs chemins (en fait, *k* chemins) sont possibles entre une entrée et une sortie. Ces chemins partagent les crossbar de l'étage central, réduisant de ce fait le nombre de points de croisement. Dans un commutateur multi-étages $N \times N$ typique, le premier étage se compose de N/n matrices de commutation de taille $n \times k$ chacune, le deuxième étage se compose de *k* matrices de taille $N/n \times N/n$, et le troisième étage se compose de N/n matrices de commutation de taille $k \times n$. Dans la figure B1T31, $N = 20$, $n = 10$, et $k = 3$.

Notez que nous disposons maintenant de *k* chemins disjoints de n'importe quelle matrice d'entrée à n'importe quelle matrice de sortie. Si *k* est petit, comme dans la cas de la figure B1T31, il est possible qu'une entrée et une sortie soient toutes les deux libres, mais qu'il n'y ait aucun chemin pour relier l'entrée considérée à la sortie (c.-à-d., le commutateur peut avoir une blocage interne). Quelle valeur minimale devrait avoir *k* pour que le commutateur ait une taux de blocage interne nul ? Clos a prouvé que si un contrôleur de commutateur est disposé à *réarranger* les connexions existantes quand un nouvel appel arrive (c'est à dire de *rerouter* certaines connexions), la condition est :

$$k > 2n - 2$$

Ainsi, un commutateur basé sur un réseau de Clos est un commutateur sans blocage interne avec réarrangement. Dans la pratique, les commutateurs de circuit réarrangent rarement les connexions existantes, ainsi ce résultat est principalement d'intérêt théorique.

Un réseau de Clos de taille $N \times N$ a un nombre de points de croisements égal à

$$2N(2n - 1) + (2n - 1).(N/n)^2$$

ce qui est inférieur à N^2, comme cela est le cas pour un crossbar. Ce nombre est réduit au minimum quand *n* est de l'ordre de $\sqrt{(N)}$.

Dans un commutateur mono-étage, trouver un chemin entre une entrée et une sortie est trivial. Par contre, dans un commutateur multi-étages, le contrôleur de commutateur doit trouver un chemin valide d'une entrée à une sortie au moment de l'admission d'appel. Le contrôleur maintient une liste des entrées et des sorties libres pendant chaque intervalle de temps, et emploie un algorithme de recherche (e.g. profondeur-d'abord) pour trouver un chemin. Ce chemin est alors stocké dans un *programme* de commutateur et remis au commutateur pour exécution pendant le transfert de données réel.

Même un réseau de Clos ne passe pas à l'échelle très bien avec *N*. Par exemple, pour $N = 120.000$, le nombre d'entrées optimal pour les matrices du premier étage $n = 346$; Nous avons alors besoin qu'il y ait $k > 690$ éléments intermédiaires. Prenons $k = 691$. Le réseau de Clos correspondant aura 248.8 millions de point de croisements ! Pour réduire ce nombre à des valeurs "raisonnables", nous devons soit utiliser une combinaison d'éléments de commutation spatiale et temporelle, ou choisir d'utiliser un commutateur ayant un taux de blocage interne non nul, et accepter le risque qu'un appel soit rejeté à cause du blocage interne. Une compagnie de téléphone doit établir un compromis entre la probabilité de blocage et le coût du commutateur. Dans la pratique, la plupart des commutateurs de téléphone ont taux de blocage faible, mais non nul.

1.4.2.3 La commutation spatio-temporelle

Nous rappelons qu'un échangeur d'intervalle de temps (TSI) a une seule entrée et une seule sortie, et peut être considéré comme servant à réarranger des échantillons sur la ligne d'entrée. Dans la commutation de spatio-temporelle, les sorties de plusieurs TSIs alimentent un commutateur spatial (voir figure B1T33). Il y a deux manières de considérer un commutateur spatio-temporel. Selon un premier point de vue, on peut considérer que le TSI retarde les échantillons de sorte qu'ils arrivent au bon moment selon le programme du commutateur spatial. S'il n'y avait pas d'étage de commutation temporelle, il se peut qu'à un moment donné, nous ayons plusieurs entrées de l'étage spatial destinées à la même sortie, ce qui causerait un blocage. En réarrangeant l'ordre dans lequel l'étage spatial reçoit les échantillons, nous gagnons un degré de liberté en sélectionnant le programme du commutateur spatial. Ceci nous permet d'établir un commutateur spatial avec moins de croisements qu'un commutateur de Clos, et pourtant sans blocage.

Exemple 1.3

Considérez un crossbar 4×4 (représenté dans la figure B1T33) qui a quatre entrées multiplexées, chacune supportant quatre circuits. Ainsi, le commutateur commute 16 circuits, quatre à la fois. Notez que durant le premier intervalle de temps arrivent les échantillons provenant du circuit 1 et 2. Si tous les deux doivent être envoyés sur la sortie 1, l'une d'entre elles sera bloquée. Nous appelons ceci blocage de sortie. Maintenant, supposez que nous insérons un TSI à chaque entrée. Nous pouvons réarranger l'échantillon provenant de 1 de sorte qu'il arrive à l'intervalle de temps 2 et non au premier intervalle de temps, de sorte que 1 et 13 ne soient plus en conflit. C'est l'avantage principal du commutateur spatio-temporel : il peut réduire le blocage en réarrangeant les entrées.

Une deuxième manière de considérer la commutation espace-temps est de penser que le commutateur spatial permet de permuter des échantillons entre deux lignes de sortie des TSIs. Un TSI nous permet d'échanger l'ordre des échantillons dans une ligne multiplexée, mais ne nous permet pas de placer un échantillon sur une autre ligne. En raccordant ensemble des TSIs et un commutateur spatial, nous pouvons prendre un échantillon arrivant par exemple sur la ligne d'entrée 1, et de le transmettre sur la ligne de sortie 3.

1.4.2.4 La commutation Temps-eSpace-Temps

Un commutateur Temps-eSpace-Temps est semblable à un cross de trois étages, dans lequel nous remplaçons les étages d'entrée et de sortie par des TSIs (voir figure B1T34). La manière la plus facile de comprendre le fonctionnement d'un commutateur TST est de le comparer à un commutateur spatio-temporel. Un commutateur spatio-temporel sert réarranger des échantillons dans trame sur une ligne d'entrée, et les commute vers une ligne de sortie différente. Supposons maintenant que deux lignes d'entrée veulent envoyer simultanément un échantillon à la même ligne de sortie, ce qui mène à un blocage de sortie. Nous pourrions alors employer les TSIs à l'entrée du commutateur spatio-temporel pour réarranger les échantillons de sorte qu'il n'y ait plus de blocage de sortie. Cependant, ce réarrangement peut avoir comme conséquence une arrivée hors séquence d'échantillons aux sorties. En plaçant un deuxième TSI à chaque sortie, nous pouvons réaliser l'ordonnancement désiré sur chaque ligne de sortie. Ainsi, nous pouvons employer n'importe quelle intervalle de temps disponible de commutateur spatial pour commuter les échantillons entre les lignes, réduisant ainsi la probabilité d'un blocage d'appel dans le commutateur spatial.

Exemple 1.4

Sur le schéma B1T34, supposez que les échantillons 1 et 2 provenant de la ligne d'entrée 1 et les échantillons 13 et 14 provenant de la ligne d'entrée 4 devraient être commutés à la ligne de sortie 1. De plus, supposez que les deux premiers échantillons dans la trame de la sortie 1 sont envoyés sur le groupe de liaisons A (TGA), et les deux échantillons suivants sur le groupe de liaisons B (TGB). Nous voulons donc que les échantillons 13 et 14 soient envoyés sur TGA, et 1 et 2 sur TGB. Notons d'abord que les échantillons 1 et 13 et échantillons 2 et 14 souffrent d'un blocage de sortie. Au niveau des TSIs d'entrée, nous réarrangeons ces échantillons comme montré dans le schéma pour éviter le blocage de sortie. Cependant, l'ordre au niveau de la ligne de sortie, 14, 2, 1, 13 est erroné. Le TSI sur la ligne de sortie nous permet de réarranger l'ordre convenablement.

Plusieurs variantes plus générales des commutateurs TST sont employées dans la pratique. Par exemple, nous pouvons remplacer l'étage S (Spatial) par un commutateur spatial multi-étages, réduisant de ce fait le nombre de points de croisement. Ceci mène à un cœur de commutation TSSST, utilisé dans le commutateur EWSD fabriqué par Siemens. Le commutateur 4ESS de Lucent Technologies, qui emploie un principe semblable, est un commutateur TSSSST avec quatre étages spatiaux internes et deux étages TSIs aux extrémités. Ces commutateurs multi-étages assez complexes peuvent commuter 100.000 à 200.000 circuits et sont largement déployés dans le réseau téléphonique.

1.5 Signalisation

Le dernier élément du réseau téléphonique que nous étudierons est la signalisation. Nous rappelons qu'un système de commutation se compose d'un commutateur, par lequel transite les données, et un *contrôleur* de commutateur, qui est responsable d'établir un circuit entre l'appelant et l'appelé. Quand un abonné compose un numéro, les impulsions ou les tonalités représentant le numéro appelé envoyées au commutateur d'accès du central téléphonique local. A ce stade, le contrôleur de commutateur interprète les signaux pour décider de la destination de l'appel. Si l'appel est local, alors le contrôleur établit un circuit au niveau du commutateur local entre l'entrée et la ligne de sortie correcte. Il envoie alors un signal sur le fil reliant le central téléphonique au téléphone de l'appelé pour activer la sonnerie. Quand le combiné est décroché, la voix des deux côtés est numérisée et envoyée à travers le circuit établi. Le contrôleur met en place également les enregistrements de facturation avec les numéros de téléphone appelant et appelés, l'heure de l'appel, et la durée. Quand l'une ou l'autre partie raccroche, le contrôleur de commutateur libère le circuit du commutateur et marque les ligne correspondante comme "libres".

Si l'appelé n'est pas local, le contrôleur de commutateur envoie un message d'établissement de circuit au contrôleur du commutateur de cœur de réseau le plus proche, demandant un circuit à la destination à distance. Ce contrôleur relaie la demande d'appel à un commutateur de cœur de réseau distant. Si la ligne de l'appelé est libre, alors le message retourne avec succès et chaque commutateur sur le chemin de retour est programmé afin d'établir le circuit. Le central téléphonique distant active la sonnerie du téléphone de l'appelé et le central téléphonique local envoie une tonalité de sonnerie au combiné de l'appelant. Quand l'appelé décroche, les commutateurs situés le long du circuit commencent à transférer les échantillons de voix.

Notez que le contrôleur de commutateur ne traite pas les échantillons de voix : il est seulement responsable du contrôle

du réseau. Une autre manière de dire cela est que les commutateurs sont dans le *plan de données*, tandis que les contrôleurs de commutateur sont dans le *plan de contrôle*.

1.6 Défis

Dans cette section, nous discutons quelques défis faisant face au réseau téléphonique global.

1.6.1 Multimédia

La communication multimédia, également appelée la communication à intégration de services, est la transmission simultanée de la voix, vidéo, et des données. Un exemple de communication de multimédia est une vidéoconférence avec une espace de travail partagé.

Le réseau téléphonique existant est inapproprié pour supporter un trafic multimédia et ce pour trois raisons. D'abord, la vidéo exige entre 0.35 et 3.5 Mbps de débit par flot. Bien que le réseau téléphonique supporte dans le cœur de réseau la transmission de la voix numérisé, le service téléphonique régulier entre le commutateur d'accès et l'abonné (la boucle locale) est analogique et utilise comme support les paires torsadées. Les meilleurs modems actuels ne peuvent pas réaliser 3.5 Mbps par dessus ce type de lien. Même si on utilise pour la transmission des données un lien numérique RNIS (Réseau Numérique à Intégration de Services), nous disposons seulement de circuits $K \times 64$ Kbps. Donc afin de pouvoir supporter la transmission de la vidéo numérique, le réseau doit augmenter le débit disponible aux utilisateurs par presque deux ordres de grandeur. D'ailleurs, les commutateurs doivent être capables de manipuler deux ordres de grandeur de plus en terme de bande passante, ce qui augmente la difficulté.

En second lieu, la vidéo numérique compressée et les donnés constituent par nature un trafic sporadique (bursty). En d'autres termes, la valeur maximale du débit calculé sur une période très courte (appelé débit crête) dépasse le débit moyen calculé sur une longue période (appelé débit moyen). Pour certains trafic de données typiques, la rapport débit crête/débit moyen peut atteindre 1000. Nous avons vu plus haut que les TSIs sont les éléments critiques pour la commutation dans le réseau téléphonique. Pour qu'un TSI travaille correctement, une source doit envoyer des données à un TSI au débit auquel il est servi ou à un débit moindre. Sinon, les tampons dans le TSI se rempliront, ce qui mène à la perte de paquets. Deux choix se présentent donc dans le cas d'une source de trafic sporadique. Soit on dimensionne le TSI pour fonctionner au débit maximal de la source, soit on dimensionne les TSI pour réserver des tampons mémoire pour stocker les données arrivant en rafale[3] (ou burst). Si le TSI traite les données au débit maximal de la source, dans les périodes où la source envoie plus lent que son débit maximal, le TSI fonctionnerait "à vide", gaspillant ainsi la bande passante. Si le TSI fonctionne à un débit plus faible que le débit maximal de la source, la taille des tampons mémoire au TSI doit être assez grande pour pouvoir absorber le burst le plus important, ce qui peut coûter cher en place mémoire. Nous concluons qu'un TSI, et donc le réseau téléphonique, est intrinsèquement coûteux soit en terme de bande passante soit en terme de tampons mémoire. Le réseau téléphonique n'est donc pas adéquat au transport des données multimédia.

Troisièmement, la capacité des liens du cœur de réseau est décidée en se basant sur le comportement statistique d'un grand nombre d'appels téléphoniques. Par exemple, suite à une observation effectuée le long de plusieurs décennies, les ingénieurs de télécommunication peuvent estimer de façon très précise le nombre d'appels entre deux grandes villes en un jour ouvrable typique. Ceci leur permet de planifier des ressources suffisantes sur le lien entre les commutateurs de cœur de réseau correspondants. Avec l'avènement de la communication multimédia, il n'est pas clair si les statistiques correspondant aux appels téléphoniques restent appropriées. Par exemple, les appels de modem (données) ont des durées beaucoup plus longues que les appels de voix – ceci signifie que le réseau doit planifier une bande passante plus grande pour les liens qui transportent plus d'appels de modem. Comment les durées de communication et les débit globaux changeront-ils si les sources étaient des vidéoconférences ? On ne peut pas le prédire de façon précise au jour d'aujourd'hui.

1.6.2 Compatibilité avec l'existant

Puisque les compagnies de téléphone dans les pays développées ont déjà un investissement énorme en infrastructure existante, les changements du réseau doivent être compatibles avec l'existant. N'importe quel nouvel équipement doit interopérer avec les équipements existants, et les services fournis aux clients doivent continuer à être supportés. Ce besoin de compatibilité "en arrière" contraint les choix disponibles en ce qui concerne l'ajout des services au réseau. Néanmoins, de nouveaux services doivent être ajoutés pour satisfaire aux besoins des clients. La téléphonie cellulaire, les numéros verts, et les services de fax sont des exemples de nouveaux services qui ont été intégrés avec succès dans l'infrastructure du réseau téléphonique. Le défi est d'entretenir le matériel et les systèmes de génération dépassée tout en

[3] Certains utilise la traduction "salve" pour le mot anglais burst. Nous utiliserons dans le polycopié le terme anglais.

ayant la possibilité d'ajouter de nouveaux services rapidement. ∘ cet égard, les pays en voie de développement ont probablement un avantage par rapport aux pays développés, parce qu'ils n'ont pas un grand investissement préalable en infrastructure.

1.6.3 Réglementation

La plupart des pays considèrent que le réseau téléphonique fait partie intégrante de l'infrastructure économique. Pour cela, les compagnies de téléphone partout dans le monde sont sujettes à des réglementations très strictes. Dans beaucoup de pays, un entreprise appartenant à l'état et ayant le monopole fournit le service téléphonique. En France, on l'appelle l'opérateur "historique". Cette tradition de réglementation mène d'habitude à étouffer l'innovation, en particulier quand la compagnie de téléphone est un monopole. Par exemple, les répondeurs téléphoniques ont été longtemps illégaux en Allemagne. Ainsi, ce service commode n'était pas proposé aux consommateurs allemands en raison de la réglementation démodée. Les innovations technologiques exigent une grande réactivité, et le défi est de faire ceci dans un environnement de réglementation.

1.6.4 Concurrence

Beaucoup de pays ont commencé à ouvrir leurs services téléphoniques à la concurrence. L'idée est que les forces du marché libres réduiront les prix et favoriseront la productivité. Du point de vue d'une compagnie de téléphone, ceci signifie la fin des beaux jours, et une incitation à se serrer la ceinture. L'exemple le plus en avant était la décomposition de AT&T en 1984. Jusqu'à ce temps là, AT&T contrôlait aux Etats-Unis, non seulement les lignes longues distances, mais également le service téléphonique local et les terminaux téléphoniques eux-mêmes (la plupart des équipements étaient loués par les abonnés). Le département de justice des Etats-Unis a statué que c'était monopolistique et que ce n'était pas dans l'intérêt public. En 1984, la AT&T a été morcelée en deux parties - une compagnie de téléphonie longue distance qui a gardé le nom AT&T, et un ensemble de compagnies régionales à qui il a été accordé un monopole sur le service local. Le service longue distance était ouvert à la concurrence, ouvrant la voie à des compagnies comme MCI et Sprint - en 1995, plus que cent compagnies de téléphonie longue distance opéraient aux Etats-Unis. Des changements semblables se sont produits ou sont en train de se produire en Allemagne, au Japon, en Finlande, au Royaume-Uni et en France. La concurrence a apporté des changements importants aux opérateurs historiques. Avec la perte de sources de revenu presque garanties, ils ont du licencier des personnes, recycler leur main d'œuvre, et être plus réactifs aux innovations technologiques. Ceci est parfois réalisé aux dépens de la vision et la planification long terme qui ont caractérisé les compagnie de téléphone dans le passé.

Bien que la concurrence soit une bonne chose pour la société, la transition d'un monopole géré par le gouvernement la libre concurrence doit être sagement contrôlée. Ceci peut être le défi le plus difficile à surmonter pour les réseaux téléphoniques dans le monde entier.

1.6.5 Inefficacités dans le système

La prolifération des systèmes et des formats semblables mais incompatible, est un legs d'une longue histoire du réseau téléphonique. Par exemple, le système de facturation de AT&T traite plus de deux mille types de bulletins de facturation pour générer une facture. Quelques formats remontent à plusieurs décennies et doivent toujours être entretenus au vingt et unième siècle à moins que le système de facturation soit complètement révisé - un exercice onéreux et rébarbatif. Malheureusement, ce genre d'inefficacité est plutôt la norme que l'exception. Pour prendre un autre exemple, les commutateurs sont des systèmes composés typiquement de plusieurs sous-ensembles indépendamment développés. Quelques sous-ensembles ont été écrits en langages spécifiques ou même en assembleur pour des ordinateurs à usage spécifique. Cette spécialisation était nécessaire quand les machines génériques n'étaient pas puissantes ou assez fiables. Avec la croissance de la puissance de calcul, un grand nombre de systèmes peuvent être supportés par des systèmes génériques, ce qui faciliterait la maintenance. Le défi faisant face au système est d'identifier et d'éliminer les inefficacités dues aux décisions dues à des contraintes technologiques nécessaires dans le passé, mais qui n'ont plus lieu aujourd'hui.

2.0 Les Réseaux ATM

Les concepteurs des réseaux ATM (Asynchronous Transfer Mode) avaient l'ambition de combler l'inadéquation du réseau téléphonique concernant le support des application multimédia en définissant une architecture de réseau permettant de fournir des garanties de qualité de service (QdS aussi appelée *QoS* de l'anglais Quality of Service) pour un trafic multimédia (audio, vidéo et données). Les mots clés qui sont cités lors de la présentation de l'architecture ATM sont le support du haut débit, le passage à l'échelle[4], et la gestion aisée. Les réseaux ATM ont fait l'objet de longues controverses entre les défenseurs d'un service réseau "sans connexion" (plutôt du coté des ingénieurs ayant participé à la conception de l'Internet) et ceux qui prônent un service réseau "orienté connexion" comme celui proposé par le réseau Transpac. Nous n'allons pas détailler les arguments des uns et des autres à ce stade. Nous présenterons plutôt les réseaux ATM comme une évolution de la technologie des opérateurs de télécommunications permettant d'avoir une meilleure gestion de la bande passante. Nous ne détaillerons pas non plus les différentes classes de service ATM: CBR, VBR, etc. Nous nous contenterons de décrire les idées de base et le positionnement de l'architecture ATM par rapport au réseau téléphonique (duquel elle est dérivée) et par rapport à l'Internet. L'architecture ATM a été normalisée par l'Union Internationale des Télécommunications - Secteur de la normalisation des télécommunication (ITU-T). Nous focaliserons sur la présentation de l'architecture de base indépendamment de ces normes.

Les réseaux ATM sont basés sur des concepts importants : (1) les circuits virtuels, (2) la taille fixe des paquets, (3) la taille réduite des paquets, (4) le multiplexage statistique, et (5) l'intégration de service. L'idée de base était que si ces concepts sont tous mis en œuvre, ils devraient à priori permettre d'établir des réseaux qui peuvent supporter des classes de trafic multiples (à la différence du réseau téléphonique) avec des garanties de QoS fournies aux flots individuels (à la différence de l'Internet). Cela devrait aussi permettre la construction de grands commutateurs parallèles et fournir un cadre uniforme pour la gestion du réseau (à cause de l'intégration). Nous étudions ces concepts ci-après.

2.1 Circuits virtuels

2.1.1 Mode de Transfert Synchrone (ou STM)

Les réseaux ATM ont été conçus à l'origine en réponse aux problèmes avec la transmission en mode de transfert synchrone (STM) utilisée dans les réseaux téléphoniques. Nous étudions donc d'abord comment deux circuits multiplexés en STM pourraient partager un lien physique.

Considérons une liaison entre deux centraux téléphoniques reliant deux villes par exemple Paris et Berlin. Imaginons deux utilisateurs (A et B) à Paris partageant le lien à un correspondant *commun* à Berlin comme suit : de 1:00 A.M. à 2:00 A.M., le lien est attribué à l'utilisateur A. De 2:00 A.M. à 3:00 A.M., le lien est attribué à l'utilisateur B, et ainsi de suite. Ainsi, si les émetteurs et le récepteur partagent un base de temps commune, le récepteur aura besoin uniquement de connaître le temps courant pour déterminer l'émetteur. Par exemple, s'il est 1:15 A.M., les données arrivant sur le lien doivent provenir de l'émetteur A. Donc, si les émetteurs et le récepteur partagent un base de temps commune, c.-à-d., s'ils sont *synchrones*, il n'y aura jamais de confusion au sujet des données des deux utilisateurs. Nous appelons ce type de partage du lien de transmission le mode de transfert synchrone (STM). Notez que dans STM, les données arrivant au récepteur n'ont pas besoin d'un en-tête descriptif (pour identifier la source en particulier), le fait que les données arrivent à un moment particulier est suffisant pour identifier leur source.

Dans la pratique, la synchronisation exacte entre l'émetteur et le récepteur est impossible, ainsi l'émetteur ajoute des bits de synchronisation spéciaux ayant une configuration bien connue. Le récepteur se synchronise à l'émetteur en se verrouillant à la configuration binaire transmise. Il peut alors extraire l'information correspondant à chaque utilisateur multiplexé, parce que cette information est présente dans un ordre fixe dans la trame transmise. Les problèmes avec STM décrits ci-après restent valables même avec ces modifications aux détails opérationnels.

STM a deux problèmes principaux. D'abord, même si A n'a rien à envoyer - disons, de 1:45 A.M. à 2:00 AM. - B ne peut pas employer le lien à la place. Dans STM, les intervalles de temps inutilisées sont gaspillées, parce que le récepteur distingue les données en provenance d'un émetteur uniquement à partir du temps courant (ou, pour être précis, de l'ordre des "slots" dans une trame).

Le deuxième problème est un peu plus subtile. Le mode transfert synchrone, dans la théorie, n'interdit pas aux utilisateurs A et B, de partager un lien en utilisant un *programme* arbitraire. Par exemple, A aurait pu envoyer pendant 21 secondes, puis B pendant 18 minutes, puis A pendant 12.43 minutes. Ceci permettrait à A et à B de partager le lien de façon arbitraire. Le récepteur doit, cependant, connaître le programme entier à l'avance, ce qui complique son

[4] Traduction du mot anglais "scalability". On pourrait aussi utiliser le terme scalabilité.

opération. Pour simplifier les choses, les liens sont partagés dans la pratique en utilisant un programme cyclique fixe, où tous les intervalles de temps du programme ont la même longueur. Puis, la seule information dont aura besoin le récepteur est la longueur de l'intervalle de temps et le programme. Par exemple, l'intervalle de temps pourrait être de 10 ms, et le programme pourrait être ABAABB. Si le programme se compose de N intervalle de temps, et la capacité du lien partagé est C bps, alors un utilisateur peut obtenir un débit multiple de C/N bps uniquement. Dans le réseau téléphonique, les intervalles de temps sont choisis pour correspondre à 64 Kbps (suffisant pour un appel téléphonique) de sorte qu'un utilisateur du réseau téléphonique puisse obtenir un circuit dont le débit est multiple de 64Kbps uniquement. En d'autres termes, STM limite la gamme des capacités disponibles à une connexion. Ceci représente un problème, par exemple, si une application veut 90 Kbps de bande passante : elle pourra obtenir uniquement 64 ou 128 Kbps.

2.1.2 Réseaux de commutation de paquets

Le service STM n'est pas flexible : les circuits basés sur STM doivent avoir une capacité multiple d'une valeur donnée, et les source perdent les slots qu'elles n'utilisent pas. Nous résolvons ces problèmes dans un réseau à commutation de *paquets* en rajoutant un en-tête (ou des méta-données) pour décrire les données, en particulier, leur source et leur destination (voir figure B1T44). L'en-tête permet aux commutateurs intermédiaires de stocker des paquets, afin de les expédier quand cela sera convenable. Ainsi, par exemple, si A n'emploie pas le lien, B peut l'employer, et le récepteur peut identifier la source des données par les méta-données. Par ailleurs, un partage plus compliqué du lien est possible, puisque le récepteur ne doit pas se baser sur un format fixe de trame pour décider la source des données. En particulier, une source peut être attribuée une bande passante qui n'est pas nécessairement un multiple d'un taux de base fixe, tel que 64 Kbps. Ainsi, les réseaux de commutation de paquets sont *efficaces* et *flexibles*.

2.1.3 Datagrammes et circuits virtuels

Il y a deux manières de réaliser un réseau à commutation de paquets. La première est d'ajouter à chaque paquet un en-tête contenant l'adresse complète de la destination (tout comme une adresse postale sur une enveloppe). C'est ce qu'on appelle l'approche datagramme, et chaque paquet contenant l'adresse complète de la destination s'appelle un datagramme. Les adresses peuvent être longues, ainsi si la longueur moyenne de datagramme est petite, ceci gaspille la bande passante. Par exemple, si la longueur moyenne de données est de 50 octets, et l'adresse de destination est de 4 octets, 8% de la bande passante est gaspillée à cause de l'adresse.

Selon la deuxième manière de réaliser un réseau à commutation de paquets, les en-têtes de paquet portent des identificateurs au lieu des adresses, et chaque commutateur maintient une table de correspondance entre l'indentificateur et la destination. Ceci permet d'économiser l'espace d'en-tête parce que les identificateurs ont une taille plus petite que les adresses destination. Cependant, la correspondance entre l'identificateur et la destination doit être mise en place dans chaque commutateur le long du chemin avant que la transmission de données commence. En d'autres termes, une phase d'*établissement d'appel* doit précéder la phase de *transmission de données*. Nous appelons cette approche la commutation de circuits virtuels (VC).

2.1.4 Les circuits virtuels

Considérons un commutateur S1 relié à trois systèmes finaux comme dans la figure B1T45 (nous appellerons les systèmes finaux des machines "hôtes", en utilisant la terminologie Internet). Le hôte H1 peut envoyer un message au hôte H2 selon une de deux approches. Selon l'approche datagramme, il rajoute un en-tête avec l'adresse de H2et le passe à S1 pour l'acheminer. Le commutateur lit l'adresse de la destination (H2) et consulte sa table de routage qui lui indique de passer la paquet à S2. S2 expédie à son tour le paquet à H2.

Dans le cas de l'approche "circuits virtuels", H1 envoie un message d'établissement de circuit virtuel à S1 demandant un connexion vers H2. En utilisant la même table de routage que celle citée auparavant, S1 décide que le prochain commutateur le long du chemin est S2. Il fait suivre le "paquet" d'établissement de circuit à S2 et renvoie au hôte un identificateur appelé l'Identificateur de Circuit Virtuel (VCI) (dans l'exemple, le VCI "1" est attribué au circuit initié par H1). Il initialise également un enregistrement par VC dans lequel est stockée l'interface de sortie correspondant au VCI attribué. Après la fin de la phase d'établissement de circuit, si H1 veut envoyer un message à H2, il lui suffira de mettre le VCI dans l'en-tête au lieu de l'adresse de destination complète. En examinant l'identificateur, S1 sait que ce message est destiné à S2, et l'expédie tout simplement sur l'interface correcte de sortie.

Chaque bit de l'en-tête représente une surcharge inévitable parce que chaque paquet doit contenir un en-tête. Ainsi, il est préférable d'avoir un en-tête le plus petit possible. Il est à noter que les VCIs économisent l'espace en en-tête, parce que les identificateurs ont une taille plus petite que les adresses de destination complètes. Dans ce sens, les circuits virtuels sont plus efficaces dans leur utilisation de la bande passante que les datagrammes.

En retournant à notre exemple, supposons que l'hôte H3 veut également communiquer avec H2. Il doit envoyer un

message de demande d'établissement de circuit à S1 et obtenir un VCI en retour. Considérons le lien de S1 à S2. Les paquets en provenance de H1 sur ce lien ont "1" en tant que VCI dans l'en-tête. Si les paquets en provenance de H3 ont également le même identificateur sur ce lien, alors on ne pourra pas distinguer entre les paquets de H1 et de H3. Clairement, les paquets de H3 doivent employer un VCI différent. Il y a deux manières d'assurer ceci.

Dans une première solution, deux circuits virtuels quelconques partageant un lien doivent être garantis d'avoir des VCIs différents. Un hôte qui veut établir une nouvelle connexion doit alors savoir quels VCIs sont déjà employés par les autres hôtes partageant n'importe quel lien le long du chemin vers la destination. Bien que cette solution soit théoriquement possible, elle ne passe pas à l'échelle dans de grands réseaux. Une meilleure solution est alors pour le commutateur S1 de résoudre les conflits en effectuant une permutation des VCIs. Quand S1 reçoit des paquets de H3, il modifie le VCI dans l'en-tête de 1 à 2. Ainsi, les paquets sur le lien S1-S2 ont des VCIs uniques. La permutation de VCI exige de chaque commutateur de maintenir une table de translation permettant d'associer un VCI en entrée à un VCI en sortie. Quand un commutateur accepte un appel, il choisit un VCI unique (localement) sur le lien de sortie, et inscrit la correspondance entre le VCI d'entrée et le VCI de sortie dans la table de translation.

Les circuits virtuels représentent en fait l'approche "orientée connexion" selon laquelle une source doit établir un circuit ou une connexion avant de pouvoir transférer des données (par exemple, dans le réseau téléphonique, vous devez composer et attendre l'aboutissement de l'appel avant de pouvoir parler). En revanche, nous appelons les réseaux de datagramme des réseaux "sans connexion", parce qu'une source n'a pas besoin d'établir une connexion (au niveau du réseau) avant de pouvoir communiquer (par l'exemple vous pouvez envoyer une lettre sans informer le destinataire, ni les bureaux de poste intermédiaires aux préalable). Nous faisons maintenant quelques observations à propos de l'approche commutation par circuits virtuels.

- Chaque commutateur doit maintenir des informations d'état sur chaque circuit virtuel pour déterminer le lien et le VCI de sortie pour un paquet reçu. Cette information est maintenue dans une table appelée la table de translation. La table de translation pour le commutateur S1 sur la figure B1T44 est montrée ci-dessous.

Lien en entrée	VCI en entrée	Lien en sortie	VCI en sortie
H1-S1	1	S1-S2	1
H3-S1	1	S1-S2	2

- Puisque chaque paquet porte un identificateur et non l'adresse de destination complète, et que l'identificateur est connu uniquement des commutateurs qui ont participé à la phase d'établissement d'appel, tous les paquets transmis pendant la phase de transmission de données doivent suivre le même itinéraire. Si l'un des commutateurs le long de l'itinéraire tombe en panne, le circuit virtuel est perdu. Bien que le réseau puisse ré-acheminer un circuit virtuel, cette opération reste compliquée et fastidieuse. En revanche, chaque datagramme peut "choisir" indépendamment le chemin pour atteindre la destination. Si un commutateur tombe en panne, les datagrammes stockés dans ce commutateur seront perdus, mais le réseau peut facilement re-router les autres datagrammes par un autre chemin vers la destination.

- Chaque commutateur maintient un état par circuit virtuel (VC), c.-à-d., une entrée par VC dans la table de translation. Ceci permet à un commutateur de stocker aussi d'autre information par VC, telle que la priorité relative du VC et la bande passante réservée à ce VC. En stockant cette information d'état, un commutateur peut fournir une qualité de service par VC. En revanche, les "routeurs" traitant les datagrammes ne stockent pas d'information d'état par "flux". En fait, les routeurs ne distinguent même pas entre les paquets de sources différentes. Ainsi, il est plus difficile de fournir une qualité de service par "flux" avec les routeurs de datagramme.

- Les actions à accomplir pendant les phases de transfert de données et d'établissement d'appel (ou signalisation) sont distinctes. Les contrôleurs de commutateur établissent et libèrent les appels et ne sont pas impliqués dans le transfert effectif des données. Cette séparation des "données" et du "contrôle" nous permet de réaliser la partie transfert de données dans les commutateurs entièrement en matériel (hardware), de façon à augmenter la vitesse de commutation, réduire les coûts et simplifier le conception des commutateurs. Les procédures de signalisation complexes nécessaires une fois au début pour l'établissement de l'appel, sont réalisées par un contrôleur du commutateur qui fonctionne à une échelle de temps plus lente. En revanche, dans un réseau de datagramme un commutateur doit lire l'adresse complète et

rechercher dans la table de routage pour chaque paquet reçu. Il est plus difficile de réaliser cette tâche en hardware[5].

- Un circuit virtuel ne fournit automatiquement aucune garantie de fiabilité. Il y a mythe selon lequel les circuits virtuels fourniraient nécessairement des faibles taux de perte et une remise en séquence des paquets de données. Un réseau orienté connexion est sujet aux mêmes types de perte qu'un réseau de datagramme. La seule différence évidente est que parce que tous les paquets suivent le même chemin, ils est beaucoup plus plausible qu'ils arrivent dans l'ordre.

- Le petite taille des identificateurs des circuits virtuels réduit la surcharge au niveau de l'en-tête. Ceci permet également à des commutateurs de stocker l'état par VC, tel que l'interface de sortie sur laquelle les paquets devraient être expédiés, dans un enregistrement de la table indexé par le VCI. Quand une cellule arrive, le VCI dans l'en-tête pourrait alors être utilisé pour localiser directement l'information d'état correspondant à ce VC, ce qui rend plus facile la construction des commutateurs en hardware.

- Une source peut transférer des données sur un circuit virtuel seulement après la fin de la phase d'établissement de connexion. Cette phase nécessite au moins un temps égal au délai aller-retour, parce que le message de d'établissement de connexion doit transiter de la source à la destination et la confirmation devrait revenir à la source. Si la source et la destination sont distants l'un de l'autre, le délai aller-retour peut être important. Par exemple, si les deux hôtes sont à une distance de 4000 Km, l'établissement de la connexion nécessitera 39.6 ms, alors que la transmission de données nécessitera 19.8 ms pour que le paquet d'un octet de contenu soit transmis. En revanche, dans un réseau de datagramme, il n'y a pas de phase d'établissement de connexion au niveau *réseau*[6], et ainsi l'octet peut arrivera au bout de 19.8 ms soit trois fois plus vite que le premier cas.

- Un circuit virtuel établi pour une durée d'un appel, puis libéré, s'appelle un circuit virtuel commuté ou un SVC (Switched Virtual Circuit). L'établissement d'un SVC nécessite donc une phase de signalisation, assez complexe, et prend du temps. Pour éviter cette phase, un utilisateur peut "câbler" une entrée dans la table de translation de chaque commutateur le long du chemin entre la source et la destination d'un VC. Un circuit établi de cette manière s'appelle un circuit virtuel permanent ou un PVC.

- Une façon de réduire le délai de la phase d'établissement d'appel, est d'allouer au préalable une plage d'identificateurs de circuits virtuels entre une source particulière et un destinataire particulier. Puis, en choisissant un VCI dans cette place, la source évite la phase d'établissement d'appel avant le transfert de données. Nous appelons cette plage de circuits virtuels pré-établis un chemin virtuel (ou Virtual Path VP). Les chemins virtuels réduisent donc la signalisation, mais aussi la taille de la table de translation sur les commutateurs, parce qu'ils permettent d'agréger des informations sur plusieurs circuits virtuels en une seule entrée. Par contre, les ressources allouées à un chemin virtuel (les VCIs) sont inutilisables pour d'autres connexions même si aucun circuit virtuel ne les emploie.

- Une autre manière d'éviter le délai de la phase d'établissement d'appel est de consacrer un VCI pour transporter des datagrammes. Dans ce cas, un commutateur, remettra les paquets arrivant sur ce VCI à un routeur de datagramme localisé dans le même système[7], qui expédie le paquet en examinant son adresse de destination.

2.2 Paquets de taille fixe

Les réseaux ATM utilisent des paquets de taille fixe (appelés "cellules") et non des paquets de longueur variable pour trois raisons : (1) un matériel plus simple pour les tampons mémoire, (2) un ordonnancement plus simple au niveau du lien de sortie, et (3) une plus grande facilité pour construire de grands commutateurs parallèles.

2.2.1 Tampons mémoire

Nous avons vu qu'un commutateur de paquets doit avoir des tampons pour stocker les paquets avant de les expédier. L'allocation dynamique de tampons mémoire pour des paquets de taille variable est plus difficile que pour des cellules à taille fixe. Pour sauvegarder des paquets de taille variable en mémoire, un commutateur doit trouver un espace contigu assez grand pour contenir le paquet, ou sinon lier des zones mémoire non contiguës à l'aide de pointeurs. Quand un paquet est transmis, une zone mémoire de taille variable est libérée, ce qui rend la gestion de la mémoire disponible plus

---

[5] Il existe cependant des routeurs IP qui utilisent un matériel spécifique pour accélérer la fonction de relais de paquet (fast packet forwarding).

[6] TCP est un protocole de *transport* "orienté connexion" qui fonctionne au dessus d'un service *réseau* "sans connexion" fourni par IP. UDP est un protocole de transport "sans connexion".

[7] Un même "système" peut contenir un routeur IP (couche 3) au dessus d'un commutateur ATM (couche 2).

coûteuse. Avec des cellules de taille fixe, un commutateur peut facilement maintenir une zone de tampons libres et allouer l'espace mémoire pour une cellule. Puisque toutes les cellules ont la même taille, aucune fragmentation de mémoire ne se produira.

2.2.2 Ordonnancement sur le lien de sortie

Regardons de plus près le commutateur S1 sur le schéma B1T45. Notez que les hôtes H1 et H3 partagent le lien S1 - S2. Un ordonnancement au niveau de ce lien détermine la nature de ce partage en décidant le prochain paquet à envoyer sur le lien partagé. Par exemple, si on (l'ordonnanceur) décide d'envoyer trois cellules de H1 pour chaque cellule de H3, H1 aura trois fois plus de bande passante que H3. Si l'ordonnanceur décide de donner aux cellules de H1 la priorité par rapport aux cellules de H3, les cellules de H1 auront un délai d'attente inférieur à celui des cellules de H3. Ainsi, les séries de transmissions de paquet sur un lien de sortie déterminent la bande passante et le délai constatés pour les paquets d'un circuit virtuel. En programmant l'ordonnanceur de lien convenablement (par le biais d'une discipline d'ordonnancement), un opérateur de réseau peut attribuer différentes bandes passantes et délais aux différents circuits virtuels. Cette attribution est simplifiée si les paquets sont de taille fixe. Par exemple, un ordonnanceur qui sert trois cellules de H1, puis une de H3 garantit un quart de la bande passante du lien à H1. Si chaque paquet pouvait avoir une taille différente, la gestion de l'allocation de la bande passante et des délais serait plus difficile. Par exemple, supposons que H1 envoie des paquets de 560 octets et H2 envoie des paquets de 132 octets, et que l'on voudrait partager la bande passante du lien de façon égale entre les deux hôtes. L'ordonnanceur doit se rappeler de donner au paquet suivant de H2 la priorité par rapport au paquet de H1, afin de pouvoir égaliser leur part de bande passante. Ainsi, il est plus difficile de réaliser des rapports simples avec des paquets de taille variable.

2.2.3 Commutateurs de paquets parallèles de grande taille

Une manière simple de réaliser un commutateur est de mémoriser les paquets à chaque entrée, puis avoir un processeur qui traite les entrées à tour de rôle, en copiant les paquets à la sortie appropriée. Les commutateurs téléphoniques de grande taille, pouvant avoir plus que cent mille entrées et sorties, ne peuvent pas être réalisés avec un seul processeur ; le commutateur doit traiter des paquets en parallèle. La manière habituelle de réaliser un commutateur parallèle est de le diviser en trois parties : tampons mémoire d'entrée, la matrice de commutation et les tampons mémoire de sortie. La matrice de commutation est une interconnexion d'éléments de commutation qui transfère des paquets d'une entrée quelconque à une sortie quelconque. Pour maximiser le parallélisme, tous les éléments de commutation doivent accomplir un transfert partiel de paquet simultanément. Puisque le temps de transfert d'un paquet dépend de sa longueur, un parallélisme maximal est réalisé quand tous les paquets sont la même taille. Avec des paquets de taille variable, quelques éléments de commutation seront en état de famine, attendant qu'un élément précédent termine son traitement, ce qui diminue le degré de parallélisme.

2.2.4 Problèmes de la taille fixe

Les cellules de taille fixe ont certains avantages par rapport aux paquets de taille variable, mais présentent également quelques problèmes. Une source qui veut envoyer une "grande quantité" de données, plus grande que la taille choisie pour les cellules doit alors segmenter les données pour les transmettre dans des cellules de taille fixe, et la destination pourrait alors être amenée à effectuer le réassemblage de ces paquets. Cette opération peut être coûteuse en temps de traitement, en particulier si la taille choisie pour les cellules est plus petite que l'unité de données moyenne de niveau application (Application Data Unit ou ADU), et si le réseau fonctionne à un débit élevé (même dans des réseaux de paquets de taille variable, les systèmes finaux doivent segmenter et rassembler les unités de données plus grandes que la taille maximum autorisée pour les paquets. Cette taille maximale (Maximum Transmission Unit ou MTU), est cependant en général plus grande que la taille moyenne de l'ADU. Réciproquement, si une source veut envoyer moins de données que la taille de cellule choisie, la bande passante est gaspillée. Un deuxième problème est qu'en fragmentant un long message en des cellules à taille fixe, la dernière cellule peut ne pas être remplie entièrement. Ceci gaspille aussi la bande passante. Les concepteurs de la technologie ATM ont estimé que les avantages d'une taille fixe de cellules étaient supérieurs aux problèmes.

2.3 Paquets de taille réduite

Les cellules ATM sont de taille réduite à cause de la manière avec laquelle les réseaux ATM transportent le trafic de voix : l'émetteur numérise les échantillons de voix, les met dans une cellule, ajoute un en-tête de cellule, et remet la cellule au réseau pour transmission. Le récepteur reçoit les "échantillons", les convertit en format analogique, puis les passe au haut parleur. Le délai entre le moment où l'utilisateur parle du côté émetteur et le moment où le son est entendu par l'utilisateur du côté récepteur dépend du nombre d'échantillons de voix rassemblés dans chaque cellule. Le délai de mise en paquet (ou paquetisation) est minimal si l'émetteur met un seul échantillon par cellule. Cependant, ceci cause également le surcoût maximal à cause des en-têtes de cellules, l'envoi d'un seul échantillon par cellule est donc inefficace. Il y a un compromis entre un délai de paquetisation faible et un surcoût d'en-tête important.

2.3.1 Taille standard des cellules

Quand le CCITT (Comité Consultatif International Télégraphique et Téléphonique), l'organisme de normalisation international pour les télécommunications (remplacé maintenant par ITU-T), décidait la taille de paquet pour les réseaux ATM en 1989, les avantages et les inconvénients d'une petite et d'une grande taille de cellules. Nous avons cité dans la section 1.2.2 que les compagnies de téléphone préfèrent un délai de paquetisation réduit, parce que ceci réduit le besoin d'annulation d'écho sur les liens de nationaux ou internationaux. Ainsi, il y avait une forte pression des représentants de compagnie de téléphone pour une taille de paquet aussi petite que possible, tout en maintenant toujours un surcoût d'en-tête tolérable. D'autre part, comme les Etats-Unis est un pays étendu géographiquement, ce qui signifie un délai de propagation de bout en bout assez important, les compagnies de téléphone américaines avaient déjà investi dans des équipements d'annulation d'écho pour le cœur du réseau. Ce qui les préoccupait était donc la réduction du surcoût d'en-tête autant que possible : elles voulaient une taille de cellules de 64 octets, correspondant à un délai de paquetisation de 8 ms. Les opérateurs européens et japonais n'avaient pas investi beaucoup dans les équipements d'annulation d'écho parce que leurs pays sont moins étendus géographiquement et pour éviter la nécessité d'ajouter des e'quipements d'annulation d'écho à toutes leurs lignes de longue distance, ils ont demandé une taille de cellules de 32 octets, correspondant à un délai de paquetisation de 4 ms. Les deux groupes ont finalement décidé de couper la poire en deux, et la taille de cellule ATM adoptée dans la norme internationale ITU-T est de 48 octets. L'en-tête étant de 5 octets, la taille totale de la cellule ATM est de 53 octets. Ironiquement, cette décision nous fait perdre l'un des avantages principaux des circuits virtuels, le faible surcoût des en-têtes. L'efficacité d'un réseau ATM est "garantie" pour être inférieure ou égale à 90.57% parce que sur chaque 53 octets, 5 octets sont du gâchis.

Exemple 2.1

(a) Calculez le délai de paquetisation pour une cellule ATM et pour un paquet 1500-byte. (b) Considérant un débit de 64 Kbps, et que les appels de voix peuvent tolérer un délai de 100 ms, quelle distance maximale entre émetteur et récepteur sans qu'il y ait un besoin d'utiliser des annulateurs d'écho pour des cellules 32 et 64 octets ?

Solution : (a) Le délai de paquetisation pour une cellule ATM est de 6 ms. Pour un paquet de 1500 octets, le délai de paquetisation est de 182.5 ms! (b) Avec des cellules de 32 octets, le temps de transmission est de 4 ms, et le temps de paquetisation est de 4 ms. Puisque ce délai est observé dans les deux sens 100 - 2 * 8 = 84 ms peuvent être employées par le délai de propagation. La vitesse de la lumière dans la fibre est de 210.000 km/s, les 84 ms correspondent donc à une distance de 12.640 kilomètres (aller-retour). L'émetteur et le récepteur doivent alors être séparés par au plus 8820 Km si on ne veut pas utiliser des annulateurs d'écho. Avec des cellules de 64 octets, le temps de transmission est de 8 ms, et le temps de paquetisation est également de 8 ms. Il reste un budget de 68 ms pour le délai de propagation, ce qui correspond à une distance aller-retour de 14.280 kilomètres et à une distance maximale entre l'émetteur et le récepteur de 7140 kilomètres.

2.4 Le multiplexage statistique

Considérons un multiplexeur temporel multiplexant les cellules ATM de quatre sources sur une ligne de sortie partagée (voir B1T50). Si une cellule arrive à chacune des quatre entrées précisément chaque seconde, le multiplexeur doit avoir un débit de sortie au moins de 4 cellules/seconde, et des tampons mémoire à chaque entrée pour au plus une cellule. Puisqu'une cellule doit attendre tout au plus la transmission de trois autres cellules avant d'être "servie", le délai d'attente d'une cellule est au pire trois fois le temps de service d'une cellule. Maintenant, supposons que les cellules arrivent regroupées en des "bursts" de 10 cellules espacées d'une seconde, ce qui donne un débit maximal d'une cellule par seconde. Si l'intervalle de temps moyen entre la fin d'un burst et le début du burst suivant est de 100 secondes, de sorte que le débit moyen d'arrivée soit de 0.09 cellules/seconde (10 cellules arrivent toutes les 110 secondes), que devrait être le débit de la ligne de sortie ?

Il n'y a pas une réponse unique à cette question. Nous pourrions disposer de façon *conservatrice* d'une ligne de sortie à 4 cellules/seconde. Dans ce cas, même si les quatre bursts arrivaient simultanément, ils ne surchargeraient pas le multiplexeur. Il est probable, cependant, qu'un seul burst soit en activité à un moment donné. Ainsi, nous pourrions disposer de façon *optimiste* d'une ligne de sortie ayant un débit d'une cellule/seconde, même s'il faudrait alors faire attendre les paquets en tampons mémoire si la ligne de sortie est occupée.

La relation entre le débit de la ligne de sortie et le délai moyen d'attente d'un flux en entrée est montrée sur la figure B1T51. Si la ligne de sortie a un débit supérieur à 4 cellules/seconde, alors le délai d'attente maximal sera trois fois le temps de service d'une cellule (par exemple pour un débit de 4 cellules/seconde, ce délai est de 0.75 s). Si le débit de la ligne de sortie est inférieur au débit moyen agrégé de 0.36 cellule/seconde, alors le délai d'attente n'est pas borné, et les tampons mémoire seront débordés. Si le débit varie entre ces deux valeurs, le délai maximal est une fonction décroissante du débit de la ligne de sortie.

Supposons maintenant que le débit de la ligne de sortie est de 1.25 cellules/seconde. Puisque nous ne servons pas les cellules au débit maximal de 4 cellules/seconde, nous obtenons un *gain de multiplexage statistique* de 4/1.25 = 3.2.

Cependant, ceci aboutit à un délai d'attente maximal supérieur à 0.75 s. Nous pouvons interpréter la courbe sur le schéma B1T51 comme une mesure de la relation entre le gain de multiplexage statistique et le délai maximal. Pour avoir un taux de perte nul, le buffer d'entrée doit être assez grand pour absorber les cellules pendant le délai maximal. La courbe mesure donc également la relation entre le gain de multiplexage statistique et la taille de buffer pour un taux de perte nul.

Nous avons considéré jusqu'ici le multiplexage statistique dans un multiplexeur temporel simple. Les mêmes arguments s'appliquent également à un commutateur de cellules parce que l'ordonnanceur du lien de sortie est identique à un multiplexeur temporel. Un commutateur de cellules peut donc exploiter le gain de multiplexage statistique tout comme le fait un multiplexeur temporel.

Nous pouvons généraliser la notion du gain de multiplexage statistique à partir de notre exemple. Supposons qu'une ressource est employée par n sources sporadiques qui ont un débit maximal p et un débit moyen a. Dans le cas le pire, les demandes de toutes les sources pourraient arriver simultanément, ainsi un schéma conservateur pour l'allocation de ressource serait de les servir à un débit np. Cette approche étant onéreuse, nous pouvons également accepter une mise en attente de certaines demandes de ressource, et servir à un débit r tel que $na < r < np$. Dans ce cas là, on définit le gain de multiplexage statistique par le rapport np/r. Pour réaliser ce gain, des demandes de ressource (bande passante du lien de sortie) doivent être mises en attente (paquets en tampons mémoire). Si la durée maximale des bursts est de b secondes, les tampons mémoires requis pour éviter de perdre des paquets est approximativement $nb(p - r)$. Si le buffer est plus petit, alors des cellules peuvent être perdues. Le principe général est que toutes les fois que le débit moyen à long terme diffère du débit maximal, nous pouvons employer le multiplexage statistique afin de régler le débit de service en fonction du délai moyen.

Le gain de multiplexage statistique est un concept central dans les réseaux ATM. ◦ la différence des réseaux STM, où les échantillons de voix ne peuvent pas être stockés indéfiniment attendant une ligne de sortie, les cellules ATM peuvent être stockées avant d'être expédiées, permettant aux commutateurs ATM d'établir un compromis entre le délai d'attente des paquets (que l'on souhaite aussi réduit que possible) et le débit de la ligne de sortie (que l'on souhaite faible également). Comme le coût de la bande passante sur les liens de longue distance est beaucoup plus élevé que celui des tampons mémoire, ceci résulte en un coût d'exploitation des réseaux ATM moins élevé qu'un réseau STM équivalent (i.e. permettant de transporter le trafic considéré). Notons que le gain de multiplexage statistique dépend considérablement de la différence entre les débits crête et moyen d'une source. Si la charge du réseau ATM est telle que les débits crête et moyen sont proches, alors le réseau ne peut pas exploiter le gain de multiplexage statistique.

2.5 L'intégration de service

2.5.1 Le service traditionnel

Le trafic de voix a traditionnellement été transporté par les réseaux téléphoniques, le trafic vidéo par les réseaux de télé-diffusion, et le trafic de données par les réseaux de communication de données. Bien que les trois réseaux partagent souvent le même lien physique, ils sont différents parce que les données transmises sont commutées par différents commutateurs, et parce qu'ils sont contrôlés par des organismes séparés. En effet, même si le même lien physique supporte la voix (du réseau téléphonique), la vidéo numérique, et les données entre ordinateurs, ce fait est caché des utilisateurs des trois réseaux.

Les trois réseaux sont séparés parce qu'ils supportent différents types de trafic, avec différents besoins de service. Les sources de trafic de voix nécessite un délai de bout en bout faible (autour de 100 ms pour des conversations interactives), et une bande passante presque constante entre 10 et 64 Kbps (selon l'algorithme de codage utilisé). Les sources vidéo ont besoin d'un débit qui peut atteindre quelques dizaines de Mbps (selon l'algorithme de compression) et ne sont pas affectées par un délai inférieur à quelques secondes. Les sources du trafic de données souhaitent le "maximum " de bande passante et un délai de bout à bout "minimal", mais peuvent s'accommoder à beaucoup moins que cela.

2.5.2 L'intégration de service

Si l'on peut transporter les trois types de média en utilisant une technologie de réseau universelle en permettant à des utilisateurs d'accéder à un ou plusieurs médias simultanément, nous pouvons développer des applications multimédia multiples et fournir ainsi des services nouveaux et intéressants. Les exemples évidents sont la vidéoconférence - où on transmet l'audio et la vidéo, les jeux interactifs - où deux joueurs ou plus partagent un environnement virtuel et peuvent communiquer avec la voix et/ou la vidéo. Par ailleurs, la gestion et la planification du réseau deviennent plus simples, puisque les barrières artificielles entre les réseaux disparaissent. Du point de vue de l'utilisateur, une seule prise réseau à la maison peut fournir la voix, la vidéo, et le service de données. Cette unification de service est une raison forte qui a poussé à la conception des réseaux ATM. L'objectif d'origine des réseaux ATM était donc de servie de réseau d'intégration de service et donc de remplacer les trois réseaux (téléphone, télévision, et réseaux de transmission de

données). Cet objectif est loin d'avoir été atteint ou même approché.

2.5.6 La bande passante élevée - ou le mythe du haut débit

Les réseaux ATM peuvent fournir une gigantesque capacité de commutation parce que les cellules ATM de taille fixe peuvent être commutées très efficacement en hardware. Avec les circuits intégrés spécialisés, le coût d'un commutateur ATM 8×8 d'une capacité totale de 1 Gbps est inférieur à 90 Euros. Cette bande passante est nécessaire pour le support de sources vidéo qui exigent une bande passante moyenne de 1.5 Mbps environ. Avec un débit si élevé, même un commutateur de capacité 1 Gbps ne peut supporter que 666 flux vidéo.

2.5.7 Différents types de support du trafic

Un réseau ATM supporte des besoins de qualité de service différents en permettant aux utilisateurs d'indiquer la qualité de service désirée pendant la phase d'établissement de connexion, et en effectuant la gestion des ressources pour répondre à ces besoins. Pendant l'établissement de la connexion, les commutateurs sur le chemin entre la source et la destination vérifient s'il y a suffisamment de ressources pour accepter la connexion tout en maintenant la qualité de service pour les connexions existantes (sinon la connexion est refusée, cette opération s'appelle le contrôle d'admission). Une connexion acceptée aura la garantie de qualité de service requise. Le réseau peut fournir des garanties de service à une source uniquement dans le cas a décrit son comportement par un *descripteur de trafic* au moment de l'établissement de la connexion. Le réseau doit s'assurer que le trafic généré réellement par source obéit à la description fournie pendant l'établissement d'appel (cette opération s'appelle le contrôle du trafic). Par exemple, un utilisateur peut demander une bande passante de 1.4 Mbps pendant l'établissement de la connexion. Cette demande de ressource est transmise à chaque contrôleur de commutateur le long du chemin. Si un contrôleur n'a pas la capacité d'accepter la connexion, il rejette la demande. Dans le cas contraire, les contrôleurs de commutateurs garantissent le débit requis à la connexion et mettent à jour leur enregistrements afin de refléter la nouvelle attribution de ressource. Pendant la phase de transfert de données, les commutateurs s'assurent que la source ne transmet pas plus que 1.4 Mbps.

En plus contrôle d'admission et du contrôle du trafic, un réseau ATM devrait disposer d'un ordonnanceur de lien intelligent pour répondre aux besoins des utilisateurs. Le trafic de voix et de vidéo sont relativement continus, à la différence du trafic de données, qui est bursty. Un appel de voix fournit des données à un commutateur périodiquement, et un ordonnanceur de lien doit permettre aux échantillons de voix d'être transmis périodiquement sur le lien de sortie. En revanche, les données textuelles (fichiers, pages web) arrivent sporadiquement, et alors les cellules de données devraient être programmées seulement quand elles arrivent. L'établissement d'un programme flexible est possible dans les réseaux ATM, à la différence du réseau téléphonique, parce que les méta-données dans les cellules (les en-têtes de cellules) permettent à l'ordonnanceur de lien de remettre la transmission d'une cellule à plus tard pour des connexions de faible priorité (en les stockant) pendant qu'il sert des connexions prioritaires.

Ainsi, nous voyons que la technologie ATM permet l'intégration des services considérés traditionnellement comme séparés. Ceci peut avoir comme conséquence les économies d'échelle, et la création des nouveaux services qui étaient précédemment infaisables ou chers.

2.6 Les défis

2.6.1 La qualité de service

L'élément clef pour fournir l'intégration de service est de fournir aux connexions de type différent une qualité de service différente. Cette qualité doit être fournie par les systèmes finaux, les contrôleurs de commutateur, et les matrices de commutation. Un défi fort aux réseaux ATM réside en la proposition des normes et des systèmes qui permettent à des applications sur des systèmes finaux d'exprimer leurs besoins et de recevoir la qualité du service du réseau. Actuellement, fournir la qualité du service de bout en bout n'est pas réalisable. Beaucoup de travaux de recherches ont été menées et aucun mécanisme standard de garantir la qualité de service dans un réseau intégré n'a vu le jour.

2.6.1 Le passage à l'échelle

Les réseaux ATM existants sont de petite taille. Ces réseaux ont été à l'origine conçu pour remplacer le réseau téléphonique, le réseau de télédiffusion le réseau de communication de données. Ces réseau existants avec tous les défauts qu'ils ont, passent bien à l'échelle : c'est pourquoi ils sont étendus et réussis. Lors de la conception des réseaux ATM, il était important de s'assurer que les algorithmes et les protocoles choisis pour le réseau passent bien à l'échelle. Par exemple, le réseau ne devrait pas avoir un contrôle centralisé, et les identificateurs globalement uniques doivent être

attribués de façon hiérarchique. Comme les "grands" réseaux ATM n'ont pas été déployés à ce jour[8] il est difficile d'évaluer l'aptitude de passage à l'échelle des réseaux ATM.

2.6.2 La concurrence d'Ethernet

ATM a été considéré pendant plusieurs années comme étant une technologie à l'abri du vieillissement pour la construction de réseaux locaux à haut débit (> 10 Mbps). Les problèmes d'équipements onéreux et peu fiables, ont décrédibilisé les solutions ATM par rapport à d'autres technologies de réseaux locaux à haut débit, telles que 100 Mbps ou Gigabit Ethernet. Ces autres technologies ne permettent pas de supporter une transmission dans un réseau étendu, mais sont fiables et largement disponible à un prix raisonnable. Ces technologies ont donc dominé le marché du réseau local à haut débit. Il est à noter que la non disponibilité de la technologie ATM au niveau du réseau local, rend plus difficile la réalisation de la qualité de service de bout en bout.

2.6.3 Les normes

Dans leur début, le réseau téléphonique et le réseau Internet ont été conçus par une poignée d'individus compétents qui ont décidé des normes essentielles par consensus et en se basant sur le mérite technique. Maintenant que ces réseaux se sont développés, et comme plus de personnes sont devenues impliquées[9], le processus de normalisation a nécessairement ralenti, mais les décisions (techniquement solides) prises au début continuent à guider l'évolution de réseau[10]. En revanche, les normes ATM ont été établies dans le cadre de l'ATM forum, un grand regroupement de compagnies ayant des conflits d'intérêts, et qui n'avaient pas toutes une expérience des réseaux ATM. En essayant de forcer l'avancement de la normalisation avant que les chercheurs aient trouvé une solution de consensus, l'ATM forum a produit des normes qui reflètent parfois des estimations périmées des possibilités techniques, telles que la taille mémoire et la puissance de traitement réalisables sur des commutateurs. Ces normes font par ailleurs l'objet de compromis purement politiques. L'expérience avec l'OSI, qui a essayé de normaliser la commutation par paquets avant que la technologie ait été mature, suggère que ce processus mène probablement à des réseaux extrêmement coûteux, donc inutilisables en pratique.

2.6.4 ATM et IP

L'Internet mondial, qui est basé sur le protocole IP, représente l'infrastructure réseau la plus vaste et en croissance continue. Ce n'est pas la technologie ATM. Pour exister, les réseaux ATM doivent inter-opérer avec l'Internet. Malheureusement (pour l'ATM), cet interopérabilité est problématique, parce que les réseaux ATM et l'Internet ont des philosophies de conception fondamentalement différentes. Par exemple, l'Internet adopte le modèle "sans connexion", tandis que les réseaux ATM sont "orientés connexions". Les réseaux ATM affichent comme spécificité la réservation de ressource et la qualité du service, alors que l'Internet vante le mérite de la connectivité et le service "au mieux" (best effort). Même les extensions proposés à l'Internet pour permettre des garanties de qualité de service (au niveau IP) sont incompatibles avec les mécanismes ATM (de niveau liaison). ◦ cause de toutes ces incompatibilités, les tentatives de développement d'architecture d'interopérabilité entre ATM et IP ont échoué. La solution adoptée en fin de compte consiste à utiliser les liens ATM comme des liaisons spécialisées pour le support de trafic IP entre deux routeurs de l'Internet, sans aucune fonctionnalité au niveau ATM.

---

[8] Et ne le seront probablement jamais.

[9] 21 ingénieurs et chercheurs ont participé à la première réunion de l'IETF (Internet Engineering Task Force, organisme de normalisation des protocoles de l'Internet) tenue à San Diego en janvier 1986. 2810 personnes ont assisté au 49ème IETF en décembre 2000 (à San Diego aussi).

[10] L'IAB (ou Internet Architecture Board, 16 membres triés sur le volet) veille, entre autres, au maintien des principes de base qui ont fait le succès de l'Internet.

3.0 L'Internet, concepts et défis

L'Internet relie des centaines de millions d'ordinateurs partout dans le monde, leur permettant d'échanger des messages et de partager des ressources. Les utilisateurs de l'Internet peuvent échanger des courriers électroniques, lisent et postent des messages dans des les "*news*", accèdent à des fichiers n'importe où dans le réseau, et mettent leurs informations à disposition d'autres utilisateurs. L'Internet, qui a commencé comme projet de recherche reliant quatre ordinateurs en 1969, s'est développé continuellement pour atteindre un réseau de plus de cent millions d'ordinateurs en 2001, liant plus de un demi milliard de personnes environ.

L'Internet est une interconnexion lâche de réseaux de technologies différentes organisés en une hiérarchie multi-niveaux. Au niveau le plus bas, entre dix et cent ordinateurs peuvent être reliés entre eux, et à un routeur, par un réseau local. Un routeur est un ordinateur spécialisé qui effectue l'acheminement des données dans l'inter-réseau. D'autres peuvent être connectés à un routeur à travers le réseau téléphonique en utilisant un modem ou à travers un lien sans fil. Le réseau d'une grande entreprise ou d'un campus universitaire peut comporter plusieurs routeurs. Ces routeurs sont liés à travers le réseau du campus à un "routeur de campus" qui gère tout le trafic entrant et sortant du campus. Les routeurs de campus sont typiquement reliés par les lignes spécialisées aux routeurs appartenant à un Fournisseur de Service Internet (Internet Service Provider ou ISP). Ces routeurs, se relient à leur tour aux routeurs de cœur de réseau appelé épine dorsale ou plus communément *backbone*. Dans un pays, il y a habituellement quelques backbones liant tous les ISPs. Aux Etats-Unis, les backbones sont inter-connectés à un nombre restreint de points d'accès de réseau (Network Access Points ou NAPs). En fin de compte, les backbones nationaux sont interconnectés en un réseau maillé en utilisant les lignes internationales.

Exemple 3.1

Considérons un paquet envoyé à partir d'un ordinateur de l'INRIA Sophia Antipolis à un ordinateur du département informatique à l'Université de Wisconsin (voir B1T59). Le paquet traverse d'abord le réseau local de l'INRIA vers le "routeur de campus" t4-gateway. Puis traverse le réseau régional puis le réseau national de Renater (même ISP régional et national), puis à travers le réseau opentransit, traverse l'atlantique vers New York (à travers la câble car le délai de bout en bout est de l'ordre de 90 ms, comme l'indiquent les chiffres à côté). Puis à travers le réseau de l'ISP BBN, le paquet continue son cheminement vers la passerelle de l'Université de Wisconsin, puis à travers plusieurs routeurs arrive à la machine destination parmesan.cs.wisc.edu.

Les campus sont habituellement administrés par une seule autorité, de sorte que tous les ordinateurs dans un campus se fassent mutuellement "confiance" (les réseaux où tous les ordinateurs se font confiance sont appelés des Intranets). ه des niveaux plus élevés, cependant, le réseau est hétérogène, et les différents domaines administratifs se font rarement confiance.

3.1 Concepts

Les concepts de base de l'Internet sont l'adressage, le routage, et le protocole internet (Internet Protocol ou IP). L'adressage fournit une manière uniforme d'identifier une destination dans le réseau. Le routage permet aux paquets de données de traverser le réseau d'une l'extrémité à l'autre. Par ailleurs, le protocole IP permet à des données d'être interprétées de façon uniforme pendant qu'elles transitent à travers le réseau. Une définition assez commune de l'Internet est : l'ensemble des ordinateurs accessibles via IP.

Peut-être la manière la plus facile de comprendre ces concepts est de regarder les trois étapes pour rendre un ordinateur "apte à joindre l'Internet" :

D'abord, les gens devraient pouvoir vous adresser des données, ce qui signifie que vous avez besoin d'une adresse Internet. Vous pouvez demander une adresse à quelqu'un qui gère une partie de l'espace d'adressage IP, tel que l'administrateur du réseau de votre organisation, ou à un ISP. Si vous avez l'intention de devenir ISP, vous pouvez réserver un lot d'adresses Internet auprès de l'IANA (Internet Addressing and Numbering Authority).

Après cela, les données provenant de votre ordinateur doivent de façon ou d'autre être transmises vers la destination. Si vous avez un seul raccordement à l'Internet, toutes vos données seront transmises sur ce lien. Cependant, si vous décidez d'avoir deux liens de raccordement à l'Internet (pour plus de fiabilité par exemple), vous devrez alors exécuter un protocole de routage pour décider quel lien fait partie du meilleur chemin vers chaque destination. Ce protocole de routage doit communiquer avec les autres instances du protocole fonctionnant sur les ordinateurs voisins pour découvrir la topologie du voisinage, ce qui veut dire que le protocole de routage devrait être généralement connu, du moins au niveau dans le domaine local. Le protocole de routage devrait aussi annoncer votre présence au reste du réseau d'une manière bien connue.

Par ailleurs, quand vous envoyez des données, vous devez avoir un logiciel sur votre ordinateur qui les prépare selon le

format IP, de sorte que les routeurs intermédiaires le long du chemin vers la destination sachent traiter ce qu'ils reçoivent. Sinon, vos paquets ne sont lisibles par personne.

Une fois que vous avez accompli ces trois étapes, correspondant à l'attribution d'une adresse IP, le support d'un protocole de routage et le formatage des paquets selon le protocole, vous pouvez envoyer des données dans le réseau, en espérant qu'ils arriveront par la suite à leur destination. Vous pouvez également recevoir des données à partir de n'importe quel autre ordinateur dans le réseau.

Une raison du déploiement très rapide de l'Internet est que les trois notions de base, adressage, conduite, et IP, sont conçues pour passer à l'échelle. Par exemple, l'Internet n'a pas une autorité d'adressage centralisée, ou un point de coordination du routage. N'importe ISP peut attribuer une partie de l'espace d'adressage qui lui a été attribuée au préalable sans davantage de référence à une plus Haute Autorité. De même, le routage change dynamiquement avec l'ajout et la suppression de machines ou si les lignes de transmission tombent en panne ou sont mises en fonctionnement. Cette conception permettant un fonctionnement distribué et un autorétablissement en cas de pannes, a permis à l'Internet de passer à l'échelle.

3.2 Technologie de base d'Internet

Deux idées principales concernant l'Internet sont la commutation de paquets et la transmission store & forward.

3.2.1 La commutation de paquets

L'Internet transporte l'information en utilisant des paquets. Un paquet est composé de deux parties : le contenu, appelé "payload", et des informations sur le payload, appelées les méta-données ou l'en-tête. Les méta-données se composent des champs tels que l'adresse source et l'adresse destination, la taille du payload, le numéro de séquence du paquet, et le type des données. L'introduction des méta-données est une innovation fondamentale par rapport au réseau téléphonique. Pour l'apprécier pleinement, rappelez-vous que le réseau téléphonique transporte la voix en utilisant les échantillons numérisés, qui ne sont permettent pas une auto-identification des échantillons. Ainsi, le réseau ne peut pas déterminer d'où viennent les échantillons, ou bien où ils vont, sans information additionnelle de contexte. Les méta-données rendent possible l'auto-identification de l'information, permettant au réseau d'interpréter les données sans information additionnelle de contexte. En particulier, si les méta-données contiennent les adresses source et destination, le réseau est capable de savoir d'où vient ce paquet et où il va, quelque soit la position du paquet dans le réseau. Le réseau peut alors stocker le paquet au besoin, pendant le temps qu'il faut, puis "débloquer" sa transmission, tout en sachant ce qui doit être fait pour remettre les données. En revanche, dans le réseau téléphonique, la destination d'un échantillon est déduite à partir de l'intervalle de temps dans lequel il arrive à un commutateur. Si un commutateur précédent stocke l'échantillon, cette information de synchronisation est perdue. Ainsi, contrairement aux réseaux de paquet, les réseaux téléphoniques ne peuvent pas stocker puis expédier des échantillons.

3.2.2 Le stockage et réexpédition (Store & forward)

Les méta-données permettent une forme de relais de données appelée stockage puis réexpédition (S&F). Selon cette méthode, un paquet est stocké successivement dans une série de routeurs, et par la suite livré à la destination. Le point clé est que l'information stockée peut être réexpédiée quand ceci est convenable. Par analogie avec le fonctionnement de la poste, un bureau de poste locale peut expédier les lettres au bureau de poste principal une fois par jour, au lieu d'envoyer un facteur chaque fois qu'une personne envoie une lettre. La transmission S&F dans le réseau postal est moins coûteuse que la transmission immédiate, puisque le coût de transmission est amorti sur un grand nombre de lettres. De même, un réseau à commutation de paquets stocke les paquets jusqu'à ce qu'ils puissent être expédiés. En accumulant un certain nombre de paquets stockés avant d'accéder à un lien de transmission onéreux, un réseau de paquet permet la partage du coût de ce lien (tel qu'une ligne téléphonique internationale) entre un grand nombre d'utilisateurs. Par ailleurs, si un lien tombe en panne, les paquets entrants peuvent être stockés et livrés quand le lien revient en état de fonctionnement. Ainsi, la rupture d'une seule liaison ne cause pas une panne générale de la communication.

3.2.3 Problèmes avec le routage S&F

Un réseau S&F est moins cher à opérer qu'un réseau téléphonique, mais les utilisateurs du réseau rencontrent trois problèmes. D'abord, il est difficile pour les utilisateurs de contrôler le temps pendant lequel leurs paquets seront retardés dans le réseau. Un paquet stocké pourrait en effet être expédié après un long délai de stockage. S'il est important d'éviter les délais, comme dans le cas d'un communication voix sur IP (VoIP), alors une approche S&F de base pourrait ne pas être suffisante. En second lieu, puisque les paquets doivent être stockés, les routeurs ont besoin de mémoire, ce qui pourrait augmenter leur prix. Finalement, si beaucoup d'utilisateurs décident d'envoyer des paquets à la même destination simultanément, et il n'y a pas assez de tampons mémoire pour contenir tous les paquets de données entrants,

alors il y aura une perte de paquets. ◦ moins qu'il y ait plus de contrôle dans le réseau, les utilisateurs du service S&F doivent supposer que leurs paquets pourraient être écartés en raison de l'arrivée des bursts de paquets en provenance d'autres utilisateurs à un routeur intermédiaire ou à la destination. En revanche, comme tous les échantillons arrivent à un commutateur téléphonique à un débit fixe, aucun échantillon n'est perdu à cause d'un débordement de tampons mémoire.

Néanmoins, les méta-données et la commutation par paquets S&F sont des idées puissantes. Elles nous permettent d'établir des systèmes où le coût élevé des liens de communication peut être partagé par un grand nombre d'utilisateurs.

3.3 Adressage

Une adresse IP, correspond à un carte d'interface dans une machine (la carte d'interface est le dispositif qui relie un ordinateur à un réseau). Un ordinateur avec deux cartes a besoin de deux adresses IP, une pour chaque interface. Pour les machines ayant seulement une interface réseau, ce qui est la norme[11], une adresse IP attribuée à la carte d'interface peut être considérée comme étant l'adresse IP de l'ordinateur.

Les adresses IP sont structurées en une hiérarchie de deux niveaux (voir B1T67). La première partie est le numéro de réseau et la deuxième partie est le numéro d'interface dans le réseau. Puisque les numéros de réseau sont globalement uniques, et les numéros d'interface dans un réseau sont également uniques, chaque interface dans l'Internet est identifiée de façon non ambiguë. Une fois qu'une autorité centrale attribue à un administrateur de réseau un numéro de réseau unique, l'administrateur peut attribuer une adresse IP globalement unique avec ce préfixe, permettant ainsi un contrôle décentralisé de l'espace d'adressage. Si les adresses IP étaient "plates" ou sans hiérarchie, une autorité centrale aurait besoin de vérifier l'unicité de chaque nouvelle adresse IP avant attribution. Nous étudions l'adressage Internet plus en détail dans le bloc 3.

3.3.1 Les classes d'adresse

Un problème intéressant est de décider combien de bits de l'adresse IP devraient correspondre au numéro de réseau et combien au numéro d'interface. Si l'Internet était constitué d'un grand nombre de réseaux, mais chaque réseau avait seulement quelques interfaces, il vaudrait mieux attribuer plus de bits au numéro de réseau qu'au numéro d'interface. S'il y avait beaucoup d'interfaces par réseau et peu de réseaux, alors nous devrions inverser l'attribution. Les adresses de la version actuelle de IP (IPv4) ont une taille de 32 bits. Si nous employons 24 bits de l'adresse pour le numéro de réseau et 8 bits pour le numéro d'interface, nous pouvons adresser 2^{24} (16.772.216) réseaux, chacun ayant 256 machines au maximum. Si nous employons 8 bits de l'adresse pour le numéro de réseau et 24 bits pour le numéro d'interface, nous pouvons adresser 256 réseaux, chacun avec 2^{24} interfaces. Les concepteurs d'Internet avaient choisi au début 8 bits pour le numéro de réseau, parce qu'ils ont estimé que l'Internet n'aurait jamais plus de 256 réseaux ! L'attribution des bits de l'adresse permet actuellement une plus grande flexibilité.

Divisons l'espace d'adresse en plusieurs classes, les adresses de classe A, de classe B, de classe C, et de classe D avec un nombre différent de bits attribués aux numéros de réseau et d'interface dans chaque classe. Une adresse de classe A est composée 8 bits pour le numéro de réseau et 24 bits pour le numéro d'interface. Une grande composante de l'Internet utiliserait alors une adresse de classe A. En revanche une adresse de classe C est composée de 24 bits de numéro de réseau et seulement 8 bits pour le numéro d'interface. Seulement 256 machines peuvent être installées dans un réseau de classe C, ce qui pourrait être suffisant pour la plupart des réseaux locaux (Local Area Networks ou LANs).

On distingue parmi les quatre classes d'adresses en vérifiant le premier bit du numéro de réseau. Si le premier bit est un 0, alors l'adresse est une adresse de classe A ; si les deux premiers bits sont 10, c'est une adresse de classe B ; si les premiers bits sont 110, c'est une adresse de classe C ; s'ils sont 1110, c'est une adresse de classe D. L'utilisation de bits du numéro de réseau afin de distinguer parmi les classes d'adresse signifie que chaque classe contient moins de réseaux qu'il pourrait avoir sinon. Ainsi, par exemple, les numéros de réseau de classe A ont une longueur de 7 bits seulement (il peut y avoir seulement 128 grands réseaux) et les numéros de réseau de classe B ont une longueur de 14 bits seulement, permettant 16.384 réseaux "moyens".

Bien que l'adressage basé sur des classes permette d'avoir des réseaux de différentes tailles, il n'est pas suffisamment flexible. Par exemple, beaucoup de campus ont plus de 256 nœuds et ont ainsi besoin au moins d'une adresse de classe B. D'autre part, ils ont beaucoup moins que les 65.536 machines autorisées dans un réseau de classe B. Ainsi, ces campus gaspillent une fraction substantielle de l'espace d'adressage IP. Afin d'essayer de réduire ce gaspillage, l'Internet a récemment adopté des adresses sans classes, appelées les adresses CIDR (Classless Inter Domain Routing addresses).

---

[11] On voir de plus en plus deux interfaces (Ethernet et 802.11) pour les ordinateurs portables, avec tous les problèmes de sécurité et gestion que cela implique.

Avec cette forme d'adressage, le numéro de réseau peut avoir une longueur quelconque, de sorte que la taille d'un réseau puisse être bornée par n'importe quelle puissance de 2. Plus de détails au sujet des adresses CIDR peuvent être trouvés dans bloc 3.

3.3.2 De plus longues adresses

Les adresses IPv4 ont une taille de 32 bits et peuvent adresser 2^{32} (4.294.962.296) interfaces si l'espace d'adressage est complètement utilisé. En réalité, une fois que l'autorité centrale aura attribué une partie de l'espace d'adressage, elle ne peut pas contrôler comment cette partie est utilisée. Même si une organisation à qui est a été attribuée une partie de l'espace d'adressage en utilise seulement une petite fraction, le reste de l'Internet doit respecter cette attribution et ne peut pas réutiliser les adresses.

Aux débuts de l'Internet, quand seulement quelques organismes étaient connectés, des adresses de classe A et de classe B ont été facilement données à des organismes qui ne pouvaient pas utiliser complètement leur part de l'espace adressage. En conséquence, leur espace d'adressage comporte des plages d'adresses libres, mais qui ne peuvent pas être reprises. Des organismes plus récemment connectés à l'Internet ne peuvent pas obtenir un espace d'adressage aussi grand qu'ils en ont besoin. On estime que l'espace d'adressage sera épuisé dans un futur proche[12]. Pour résoudre ce problème, la version 6 du protocole IP est définie avec des adresses de 128 bits. Les concepteurs de IPv6 s'attendent à ce que ce gigantesque espace d'adressage résolve une fois pour toutes le problème de manque d'adresses IP.

3.4 Le routage

L'Internet expédie les paquets IP d'une source vers une destination en utilisant le champ "adresse IP de destination" dans l'en-tête de paquet. Un routeur est défini comme étant un hôte connecté à au moins deux réseaux. Chaque routeur le long du chemin entre la source et la destination dispose d'une table de routage ayant au moins deux champs : un numéro de réseau et l'interface sur laquelle envoyer les paquets avec ce numéro de réseau dans le champ adresse IP de destination. Le routeur lit l'adresse de destination dans l'en-tête d'un paquet reçu et utilise la table de routage pour l'expédier sur l'interface appropriée. Le schéma (B1T70) illustre ce concept. Dans ce schéma, les boîtes rectangulaires représentent les interfaces de hôtes et le cercle du milieu représente un routeur.

Il y a trois réseaux, numérotés 1, 2, et 3, et les adresses de interfaces de hôtes sont établies à la IP. Par exemple, l'adresse 2.3 désigne l'interface de hôte ayant le numéro 3, connecté au réseau ayant le numéro de 2. Dans la figure, le routeur a une interface sur chacun des trois réseaux.

Supposons maintenant qu'une interface sur un réseau peut joindre toutes les autres interfaces sur ce réseau. (ceci est facilement réalisable sur les réseaux qui partagent un milieu physique de transmission, tel que Ethernet.) Le routeur peut alors joindre toutes les interfaces sur les trois réseaux parce qu'il a une interface sur chaque réseau. Par symétrie, chaque interface de hôte peut joindre le routeur. Ainsi, quand un hôte veut envoyer un paquet à un hôte distant, il l'envoie au routeur, qui l'expédie en utilisant sa table de routage. Puisque chaque hôte dans les trois réseaux sur le schéma B1T70 peut accéder tous les autres hôtes via le routeur, nous pouvons maintenant considérer le groupement de ces trois réseaux comme étant une zone entièrement connexe comme nous l'avons supposé pour chacun des réseaux. Nous pouvons alors établir des groupements plus grands, en mettant en place des routeurs ayant des interfaces appartenant à au moins deux zones. Nous pouvons continuer ce processus et aboutir à des domaines très grands de cette façon. Ceci est description très simplifiée du routage dans l'Internet. Les mécanismes et protocoles de routage seront étudiés dans le bloc 3.

Il convient de remarquer que les adresses IP (qui sont en fait des adresses hiérarchiques) contiennent un indication au sujet de la localisation géographique de l'interface. Par exemple, un hôte ayant une adresse IP 3.1 indique qu'il est dans le réseau 3. Si cette hiérarchie est toujours respectée, la table de routage dans le routeur ne doit pas maintenir une entrée pour chaque adresse dans le réseau 3. On peut y mettre tout simplement "si l'adresse de destination est 3.*, envoyer le paquet correspondant à l'interface 3." L'indication géographique dans les adresses IP permet l'agrégation des adresses de destination dans les routeurs et représente un élément essentiel pour le passage à l'échelle du routage dans l'Internet.

3.4.1 Routes par défaut

▫priori, une table de routage doit maintenir le prochain nœud (next hop) pour chaque numéro de réseau dans l'Internet. L'Internet regroupe maintenant plus de 80.000 réseaux, et demander à chaque routeur de maintenir l'information concernant le next hop pour tous ces réseaux est très contraignant. Ainsi, un routeur aura à maintenir les itinéraires détaillés (c'est-à-dire, itinéraires pour des adresses de réseau absolues telles que 135.104.53, et non des adresses

[12] Ce délai dépend du temps nécessaire pour que tous les chinois aient accès à l'Internet ou que tous les téléphones portables soient connectés au réseau.

agrégées telles que 135.*) uniquement pour les réseaux "voisins". Tous les paquets ayant une adresse de réseau destination qui n'est pas "voisine" sont expédiés à une adresse de défaut, qui est l'adresse d'un routeur qui connaît plus de choses au sujet des réseaux éloignés. Par exemple, un routeur de campus connaîtrait typiquement le next hop seulement pour les réseaux locaux du campus. S'il reçoit un paquet IP avec une adresse de destination inconnue, il expédie automatiquement ce paquet sur la route par défaut qui mène au niveau supérieur dans la hiérarchie de l'Internet, c.-à-d., un routeur du réseau régional. De même, les machines hôtes sur un réseau garde en général une seule route, qui n'est autre que la route par défaut vers le routeur local. Seule les routeurs du cœur de réseau doivent alors maintenir une table de routage "globale" contenant l'information sur le next hop pour chaque réseau dans l'Internet.

3.4.2 Calcul des tables de routage

Nous avons regardé comment le routage est effectué quand une table de routage existe déjà. Nous discutons maintenant comment cette table est calculée. Une façon de réaliser ceci serait que chaque hôte et routeur annonce son identité et une liste de ses voisins à une autorité *centrale* qui calculerait périodiquement les tables de routage optimales pour tous les routeurs. Cette solution n'est pas souhaitable, parce qu'elle ne passe pas à l'échelle. En d'autres termes, à mesure que la taille du réseau augmente, la charge sur l'ordinateur central augmente au delà de la capacité des plus grandes machines disponibles, limitant ainsi la taille du réseau. D'ailleurs, si cet ordinateur central tombe en panne, le réseau entier serait inutilisable.

Une meilleure approche serait alors que les routeurs échangent entre eux des messages qui indiquent les réseaux qu'ils peuvent atteindre, et le coût d'atteindre ces réseaux. En examinant son état local et en le combinant avec l'information reçue de ses pairs, chaque routeur peut calculer de façon *distribuée* une table de routage cohérente.

3.5 Le contrôle de bout en bout

Comme les idées de base de la conception d'Internet mettent en valeur le contrôle décentralisé, les systèmes finaux ne peuvent pas se baser sur le réseau pour assurer le transfert de données fiable ou en temps réel. Un concept importante de la philosophie de l'Internet est donc le contrôle de bout en bout. L'idée est que le transfert de données fiable doit être fourni par des protocoles fonctionnant au niveau des systèmes finaux, et non dans le réseau. L'intelligence dans les systèmes finaux devrait alors réparer les déficiences du réseau (délai, perte, etc). C'est exactement l'inverse de la philosophie de conception du réseau téléphonique, où l'on considère que les systèmes finaux (appelés d'ailleurs terminaux) sont sans intelligence et que toute l'intelligence est fournie par les équipements du réseau.

La fiabilité de bout en bout dans l'Internet fournie par une couche de protocole qui fonctionne au-dessus du protocole IP. IP fournit la connectivité de bout en bout sans garantie de fiabilité. Une protocole de contrôle de transmission au-dessus d'IP, appelé TCP (Transmission Control Protocol), fournit habituellement la contrôle d'erreur et la contrôle de flux pour permettre un transfert de données fiable contrôlé au dessus d'un réseau non fiable. Nous étudierons en détail le protocole TCP dans le bloc 4.

Il est à noter que la philosophie du contrôle de bout en bout permet de construire un réseau avec peu d'hypothèses, ce qui simplifie l'intégration de nouvelles technologies de réseau (tels que les réseaux radio et les réseaux satellite) dans l'Internet. Malheureusement, ceci implique également qu'un changement de TCP nécessite des changements à tous les systèmes finaux, ce qui explique la grande difficulté de supporter de nouvelles versions du protocoles. En revanche, dans le réseau téléphonique, il est difficile d'intégrer de nouvelles technologies de réseaux, mais l'introduction de nouveaux services est plus facile que dans le cas de l'Internet car elle ne nécessite pas la coopération de l'utilisateur. Par exemple, l'introduction des téléphones cellulaires s'est effectué sans aucun changement en ce qui concerne les téléphones existants.

3.6 Défis

3.6.1 Le manque d'adresses IP

Un des défis technologiques les plus immédiats auxquels l'Internet devrait faire face c'est le manque d'adresses IP. Comme nous avons vu, les adresses d'IP sont structurées selon une hiérarchie, avec un numéro de réseau et un numéro d'interface. Une fois qu'une autorité centrale attribue à une organisation un numéro de réseau, l'organisation peut attribuer à sa guise des adresses dans ce réseau. Malheureusement, si l'organisation n'utilise pas entièrement l'espace d'adresses disponible, cet espace n'est pas utilisable par d'autres organismes. Aux débuts de l'Internet, des adresses de classe A et B ont été facilement attribuées, parce que personne n'avait prévu le succès formidable du réseau. Cependant, même les organismes les plus grands ne peuvent pas utiliser entièrement les 2^{24} adresses disponible dans un réseau de classe A. En réalité, l'utilisation réelle de l'espace d'adresse de 32 bits est tout à fait clairsemée.

Une solution consiste en l'attribution de plusieurs adresses de classe C aux nouveaux organismes qui demandent des adresses au lieu de leur attribuer des adresses de classe B ou de classe A. Cependant, chaque adresse de classe C

correspond à une entrée dans la table de routage des routeurs de backbone. Or une taille importante des tables de routage rend l'expédition des paquets plus lente (à cause des temps de recherche de l'adresse dans la table).

Actuellement, des efforts sont fournis pour déployer IPv6[13] ce qui permettra d'avoir des adresses de 128 bits. La compatibilité avec les adresses IPv4 est aussi considérée. On attend donc le déploiement IPv6 pour oublier à jamais la pénurie d'adresses IP. Ceci dit, l'adoption de CIDR (sera décrit dans le bloc 3) a retardé l'échéance, du moins pour quelques années.

3.6.2 Problèmes du contrôle décentralisé

Le décentralisation du contrôle, qui est en fait essentielle à la scalabilité de l'Internet, pourrait également le transformer en un système anarchique où aucun service fiable ne peut être garanti. Il suffit qu'un seul routeur écarte des paquets ou les altère de façon imperceptible pour causer des problèmes partout dans le réseau (par exemple, si un paquet de routage altéré cause un état incohérent des tables de routage partout dans le réseau).

Un problème plus pernicieux est la sécurité. Les paquets envoyés sur l'Internet sont visibles à tous ceux qui souhaitent les examiner. La sécurité n'était pas un problème sérieux quand l'Internet était un joyau entre les mains des universitaires. Maintenant qu'on voudrait faire du business sur Internet, il faudrait considérer plus sérieusement la fait que les paquets envoyés sur l'Internet pourraient être examinés ou sujets à une usurpation d'adresse IP par des nœuds intermédiaires sans scrupules. Le chiffrement de bout en bout résout partiellement ce problème, mais augmente la complexité au niveau des systèmes finaux et nécessite un mécanisme de distribution de clés scalable et efficace. Ces problèmes sont moins graves dans un Intranet, où tous les ordinateurs et routeurs sont administrés par une seule organisation et se font donc confiance.

Un troisième problème du au contrôle décentralisé est qu'il n'y a pas de moyen uniforme d'effectuer la comptabilité sur l'Internet. Ni les universitaires, ni les militaires ne se sont préoccupés de la comptabilité, les mécanismes pour la facturation et le partage des frais sont inexistants. Ceci a mené à la facturation forfaitaire basée sur la bande passante du lien d'accès[14]. Ce système est simple et fonctionne assez bien, mais le support du trafic multimédia pourrait augmenter le besoin de comptabilité (collecte des informations) et de facturation.

Quatrièmement, il n'y a sur Internet un équivalent aux pages blanches ou aux pages jaunes de l'annuaire de téléphone. Rechercher une adresse e-mail de façon sûre est impossible ; même si on en trouve une, il n'y a aucune moyen de vérifier si l'adresse est toujours utilisé ou si elle a été usurpée par un utilisateur sans scrupules.

Finalement, le contrôle décentralisé signifie que le routage peut parfois être sous-optimal. Comme il n'y a pas d'autorité centrale responsable de vérifier que les décisions sont optimales au niveau global, une série de décisions localement optimales peut avoir comme conséquence un chaos global. Comme exemple extrême, on a découvert que pendant plusieurs mois, tout le trafic entre la France et le Royaume-Uni était envoyé par l'intermédiaire des Etats-Unis. Bien qu'un tel routage sous-optimal pourrait avoir lieu dans n'importe quel réseau, le point est que ce problème de routage n'a pas été détecté pendant plusieurs mois parce qu'aucune autorité n'était responsable du routage.

3.6.3 Le support du multimédia

Les applications "multimédia" qui ont besoin des garanties de performance temps réel telles que un délai borné et un débit minimal ne sont pas bien supportés par l'Internet courant. Ces paramètres appelés souvent les paramètres de la qualité de service ou QoS. Un défi important auquel l'Internet devrait faire face, réside en l'intégration de la qualité de service dans l'architecture existante.

Si un routeur (basé sur le mode S&F) stocke le trafic en provenance de tous les utilisateurs dans une file d'attente partagée et les sert[15] dans l'ordre de réception (Premier Arrivé, Premier Servi - FIFO ou FCFS en anglais), un utilisateur qui injecte un burst de paquets qui remplissent la file d'attente partagée peut causer un délai et un taux de perte plus élevé aux autres utilisateurs. Ceci est dû au fait que les paquets appartenant à d'autres utilisateurs sont stockés derrière le burst des paquets et seront servis après que le burst entier a été servi. Si le burst occupe tous les buffers disponibles, les paquets en provenance d'autres utilisateurs seront écartés par manque de place mémoire pour les stocker. Si un des utilisateurs en "compétition" avec le burst envoie des paquets de voix, le récepteur entendra des coupures, qui peuvent

---

[13] Voir http://www.ietf.org/html.charters/ipv6-charter.html

[14] Par exemple, on trouve en France un abonnement ADSL 512 Kbps pour moins de 30 euros/mois.

[15] Servir un paquet signifie le transmettre sur la ligne de sortie

être très gênantes. De même, si une vidéo est transférée, le récepteur peut voir un image saccadée ou carrément figée.

Un réseau peut fournir aux utilisateurs une meilleure qualité de service si les utilisateurs partagent plusieurs tampons mémoire (ou files d'attente) au niveau d'un routeur de façon équitable, et si le routeur sert les paquets non pas en FIFO, mais sert les différentes files d'attentes à tour de rôle. Dans notre exemple de poste, ceci correspondrait à avoir les files d'attente séparées pour demander un renseignement et pour envoyer des colis, de sorte que des utilisateurs qui ont besoin d'un service rapide ne soient pas retenu derrière d'autres. Cependant, la plupart des routeurs dans l'Internet mettent tous les paquets dans une seule file d'attente et fournissent un service de type FIFO. Ainsi, pour garantir la qualité du service, tous les routeurs existants doivent appliquer une discipline de service différente de FIFO (discipline de service à tour de rôle ou Round-Robin). Le contrôle décentralisé rend ce changement particulièrement difficile.

Un autre mécanisme est requis pour fournir la qualité du service: il s'agir de la signalisation dont l'objectif est d'informer les routeurs le long du chemin à propos de la qualité du service requise par chaque "flot" des paquets. Théoriquement, ce mécanisme de signalisation devrait établir des paramètres de QoS pour chaque "classe" de trafic, et chaque paquet entrant doit être classifié comme appartenant à une des classe prédéfinis. Il faudrait aussi appliquer un contrôle d'admission pour les nouveaux flots. Tout ceci est contraire à la philosophie de base de l'Internet qui consiste à garder le réseau simple pour assurer sa scalabilité.

4.0 Sommaire

Ce bloc est composé de trois grandes sections: la section 1 décrit de façon générale le réseau téléphonique, la section 2 les réseaux ATM et la section 3 l'Internet. Dans la première section, nous avons étudié quatre composants essentiels du réseau téléphonique : (a) les systèmes finaux, (b) la transmission, (c) la commutation, et (d) la signalisation.

La section 2 décrit les réseaux ATM qui avaient pour objectif de combiner les meilleures idées du réseau téléphonique, telles que le service connecté et la qualité de service de bout en bout du service, et ceux des réseaux informatiques, tels que la commutation par paquets. Ils sont basés sur cinq concepts importants : (1) les circuits virtuels, (2) la taille de paquet fixe, (3) la taille de paquet réduite, (4) le multiplexage statistique, et (5) l'intégration de service. Malheureusement, les problèmes de passages à l'échelle ont empêché un large déploiement de ces réseaux qui se cantonnent aujourd'hui au marché des liens à haut débit entre routeurs IP.

Finalement dans la section 3 nous décrivons brièvement les deux technologies principale utilisées dans l'Internet: la commutation de paquets et la transmission par stockage puis réexpédition (Store & Forward). L'Internet est par ailleurs basé sur l'adressage, le routage et le format des paquets définis par le protocole IP. L'Internet a connu un succès formidable à cause du principe du contrôle décentralisé et de la transmission par paquets. Justement, ces deux aspects rendent l'introduction de nouveaux services tels que la comptabilité, la facturation et la qualité de service très difficile. L'Internet a rejeté toutes les tentatives d'enrichissement du service fourni au nom de la scalabilité. Il est peut probablement condamné à l'ossification.

Bloc 2

Les liens de communication et l'accès multiple

This bloc is divided in two sections. Section 1 introduce some concepts and terms related to the physical layer. Section 2 describes in detail Medium Access Control Level mechanisms and protocols.

1.0 The Physical layer

1.1 Transmission Terminology

Data transmission occurs between transmitter and receiver over some transmission medium. Transmission media may he classified as guided or unguided. In both cases, communication is in the form of electromagnetic waves. With guided media, the waves are guided along a physical path; examples of guided media are twisted pair, coaxial cable, and optical fiber. Unguided media provide a means for transmitting electromagnetic waves but do not guide them; examples are propagation through air, vacuum, and sea water.

The term direct link is used to refer to the transmission path between two devices in which signals propagate directly from transmitter to receiver with no intermediate devices, other than amplifiers or repeaters used to increase signal strength. Note that this term can apply to both guided and unguided media.

A guided transmission medium is point-to-point if it provides a direct link between two devices and those are the only two devices sharing the medium. In a multipoint guided configuration, more than two devices share the same medium.

A transmission may be simplex, half duplex, or full duplex. In simplex transmission, signals are transmitted in only one direction; one station is transmitter and the other is receiver. In half-duplex operation, both stations may transmit, but only one at a time. In full-duplex operation, both stations may transmit simultaneously. In the latter case, the medium is carrying signals in both directions at the same time.

Frequency, Spectrum, and Bandwidth

In this section, we are concerned with electromagnetic signals, used as a means to transmit data. Consider a signal that is generated by the transmitter and transmitted over a medium. The signal is a function of time, but it can also be expressed as a function of frequency; that is, the signal consists of components of different frequencies. It turns out that the *frequency-domain* view of a signal is more important to an understanding of data transmission than a *time-domain* view. Both views are introduced here.

Time-Domain Concepts

Viewed as a function of time, an electromagnetic signal can be either continuous or discrete. A continuous signal is one in which the signal intensity varies in a smooth fashion over time. In other words, there are no breaks or discontinuities in the signal. A discrete signal is one in which the signal intensity maintains a constant level for some period of time and then changes to another constant level. The continuous signal might represent speech, and the discrete signal might represent binary 1s and 0s.

The simplest sort of signal is a periodic signal, in which the same signal pattern repeats over time. A sine wave is an example of a periodic continuous signal and a square wave is an example of a periodic discrete signal. The sine wave is the fundamental periodic signal.

Frequency-Domain Concepts

In practice, an electromagnetic signal will he made up of many frequencies. For example, the signal $s(t) = (4/\pi) \times (\sin(2\pi ft) + (1/3)\sin(2\pi(3f)t))$ is composed of two sine waves of frequencies f and $3f$. The second frequency is an integer multiple of the first frequency. When all of the frequency components of a signal are integer multiples of one frequency, the latter frequency is referred to as the fundamental frequency. It can be shown, using Fourier analysis, that any signal is made up of components at various frequencies, in which each component is a sinusoid. This result is of tremendous importance, because the effects of various transmission media on a signal can be expressed in terms of frequencies.

The spectrum of a signal is the range of frequencies that it contains. The absolute bandwidth of a signal is the width of

the spectrum. Many signals, have an infinite bandwidth. However, most of the energy in the signal is contained in a relatively narrow band of frequencies. This band is referred to as the effective bandwidth, or just *bandwidth*.

One final term to define is dc component. If a signal includes a component of zero frequency, that component is a direct current (dc) or constant component.

Relationship between Data Rate and Bandwidth

We have said that effective bandwidth is the band within which *most* of the signal energy is concentrated. The term most in this context is somewhat arbitrary. The important issue here is that, although a given waveform may contain frequencies over a very broad range, as a practical matter any transmission system (transmitter plus medium plus receiver) that is used will be able to accommodate only a limited band of frequencies. This, in turn, limits the data rate that can be carried on the transmission medium.

Indeed, it can be shown that the frequency components of the square wave with amplitudes A and -A can be expressed as follows:

$$s(t) = A \times 4/\pi \times \Sigma^{\infty}_{k \text{ odd}, k=1} (\sin(2\pi kft))/k$$

Thus, this waveform has an infinite number of frequency components and hence an infinite bandwidth. However, the peak amplitude of the k^{th} frequency component, kf, is only $1/k$, so most of the energy in this waveform is in the first few frequency components. What happens if we limit the bandwidth to just the first three frequency components? The shape of the resulting waveform is reasonably close to that of a square wave.

Suppose that we are using a digital transmission system that is capable of transmitting signals with a bandwidth of 4 MHz. Let us attempt to transmit a sequence of alternating 1s and 0s as the square wave of B2T9. What data rate can be achieved? We look at three cases.

Case 1. Let us approximate our square wave by the first three terms. Although this waveform is a "distorted" square wave, it is sufficiently close to the square wave that a receiver should be able to discriminate between a binary 0 and a binary 1. If we let $f = 10^6$ cycles/second = 1 MHz, then the bandwidth of the signal is $(5.10^6) - 10^6 = 4$ MHz. Note that for f = 1 MHz, the period of the fundamental frequency is $T = 1/10^6 = 10^{-6} = 1$ μs. If we treat this waveform as a bit string of 1s and 0s, one bit occurs every 0.5 μs, for a data rate of $2.10^6 = 2$ Mbps. Thus, for a bandwidth of 4 MHz, a data rate of 2 Mbps is achieved.

Case 2. Now suppose that we have a bandwidth of 8 MHz. Let us consider that $f = 2$ MHz. Using the same line of reasoning as before, the bandwidth of the signal is $(5.2.10^6) - (2.10^6) = 8$ MHz. But in this case $T = 1/f = 0.5$ μs As a result, one bit occurs every 0.25 μs for a data rate of 4 Mbps. Thus, other things being equal, by doubling the bandwidth, we double the potential data rate.

Case 3. Now suppose that the square wave can be approximated by the first two terms. That is, the difference between a positive and negative pulse in resulting waveform is sufficiently distinct that the waveform can be successfully used to represent a sequence of 1s and 0s. Assume as in Case 2 that $f = 2$ MHz and $T = 1/f = 0.5$ μs so that one bit occurs every 0.25 μs for a data rate of 4 Mbps. The bandwidth of the signal is $(3.2.10^6) - (2.10^6) = 4$ MHz. Thus, a given bandwidth can support various data rates depending on the ability of the receiver to discern the difference between 0 and 1 in the presence of noise and other impairments.

In summary,

• **Case 1:** Bandwidth 4 MHz; data rate 2 Mbps

• **Case 2:** Bandwidth 8 MHz; data rate 4 Mbps

• **Case 3:** Bandwidth 4 MHz; data rate 4 Mbps

We can draw the following conclusions from the preceding discussion. In general, any digital waveform will have infinite bandwidth. If we attempt to transmit this waveform as a signal over any medium, the transmission system will limit the bandwidth that can be transmitted. Furthermore, for any given medium, the greater the bandwidth transmitted, the greater the cost. Thus, on the one hand, economic and practical reasons dictate that digital information be approximated by a signal of limited bandwidth. On the other hand, limiting the bandwidth creates distortions, which makes the task of interpreting the received signal more difficult. The more limited the bandwidth, the greater the distortion, and the greater the potential for error by the receiver. Furthermore, we can generalize these results. If the data rate of the digital signal is W bps, then a very good representation can be achieved with a bandwidth of $2W$ Hz. However, unless noise is very severe, the bit pattern can be recovered with less bandwidth than this. Thus, there is a

direct relationship between data rate and bandwidth: The higher the data rate of a signal, the greater is its effective bandwidth. Looked at the other way, the greater the bandwidth of a transmission system, the higher is the data rate that can be transmitted over that system.

1.2 Analog and Digital data transmission

In transmitting data from a source to a destination, one must be concerned with the nature of the data, the actual physical means used to propagate the data, and what processing or adjustments may be required along the way to assure that the received data are intelligible. For all of these considerations, the crucial point is whether we are dealing with analog or digital entities.

The terms *analog* and *digital* correspond, roughly, to *continuous* and *discrete,* respectively. These two terms are used frequently in data communications in at least three contexts:

• Data

• Signaling

• Transmission

We can define data as entities that convey meaning, or information. Signals are electric or electromagnetic representations of data. Signaling[16] is the physical propagation of the signal along a suitable medium. Finally, transmission is the communication of data by the propagation and processing of signals. In what follows, we try to make these abstract concepts clear by discussing the terms *analog* and *digital* as applied to data, signals, and transmission.

Data

The concepts of analog and digital data are simple enough. Analog data take on continuous values in some interval. For example, voice and video are continuously varying patterns of intensity. Most data collected by sensors, such as temperature and pressure, are continuous valued. Digital data take on discrete values; examples are text and integers.

The most familiar example of analog data is audio, which, in the form of acoustic sound waves, can be perceived directly by human beings. Another common example of analog data is video. Here it is easier to characterize the data in terms of the viewer (destination) of the TV screen rather than the original scene (source) that is recorded by the TV camera. To produce a picture on the screen, an electron beam scans across the surface of the screen from left to right and top to bottom. For black-and-white television, the amount of illumination produced (on a scale from black to white) at any point is proportional to the intensity of the beam as it passes that point. Thus at any instant in time the beam takes on an analog value of intensity to produce the desired brightness at that point on the screen. Further, as the beam scans, the analog value changes. Thus the video image can be thought of as a time-varying analog signal.

A familiar example of digital data is text or character strings. While textual data are most convenient for human beings, they cannot, in character form, be easily stored or transmitted by data processing and communications systems. Such systems are designed for binary data. Thus a number of codes have been devised by which characters are represented by a sequence of bits. Perhaps the earliest common example of this is the Morse code. Today, the most commonly used text code is the International Reference Alphabet (IRA) 4. Each character in this code is represented by a unique 7-bit pattern; thus 128 different characters can be represented. This is a larger number than is necessary, and some of the patterns represent invisible *control characters*. Some of these control characters have to do with controlling the printing of characters on a page. Others are concerned with communications procedures. IRA-encoded characters are almost always stored and transmitted using 8 bits per character (a block of 8 bits is referred to as an octet or a byte). The eighth bit is a parity bit used for error detection. This bit is set such that the total number of binary 1s in each octet is always odd (odd parity) or always even (even parity). Thus a transmission error that changes a single bit, or any odd number of bits, can be detected.

Signals

In a communications system, data are propagated from one point to another by means of electric signals. An analog signal is a continuously varying electromagnetic wave that may be propagated over a *variety* of media, depending on spectrum; examples are wire media, such as twisted pair and coaxial cable, fiber optic cable, and atmosphere or space

[16] Do not confuse signaling as defined here, with network signaling as defined in the previous bloc.

propagation. A digital signal is a sequence of voltage pulses that may be transmitted over a *wire* medium; for example, a constant positive voltage level may represent binary 1 and a constant negative voltage level may represent binary 0.

Data and Signals

In the foregoing discussion, we have looked at analog signals used to represent analog data and digital signals used to represent digital data. Generally, analog data are a function of time and occupy a limited frequency spectrum; such data can be represented by an electromagnetic signal occupying the same spectrum. Digital data can be represented by digital signals, with a different voltage level for each of the two binary digits.

These are not the only possibilities. Digital data can also be represented by analog signals by use of a modem (modulator/demodulator). The modem converts a series of binary (two-valued) voltage pulses into an analog signal by encoding the digital data onto a carrier frequency. The resulting signal occupies a certain spectrum of frequency centered about the carrier and may be propagated across a medium suitable for that carrier. The most common modems represent digital data in the voice spectrum and hence allow those data to be propagated over ordinary voice-grade telephone lines. At the other end of the line, another modem demodulates the signal to recover the original data.

In an operation very similar to that performed by a modem, analog data can be represented by digital signals. The device that performs this function for voice data is a codec (coder-decoder). In essence, the codec takes an analog signal that directly represents the voice data and approximates that signal by a bit stream. At the receiving end, the bit stream is used to reconstruct the analog data.

Transmission

A final distinction remains to be made. Both analog and digital signals may be transmitted on suitable transmission media. The way these signals are treated is a function of the transmission system. We summarize hereafter the methods of data transmission. Analog transmission is a means of transmitting analog signals without regard to their content; the signals may represent analog data (e.g., voice) or digital data (e.g., binary data that pass through a modem). In either case, the analog signal will become weaker (attenuate) after a certain distance. To achieve longer distances, the analog transmission system includes amplifiers that boost the energy in the signal. Unfortunately, the amplifier also boosts the noise components. With amplifiers cascaded to achieve long distances, the signal becomes more and more distorted. For analog data, such as voice, quite a bit of distortion can be tolerated and the data remain intelligible. However, for digital data, cascaded amplifiers will introduce errors.

Digital transmission, in contrast, is concerned with the content of the signal. A digital signal can be transmitted only a limited distance before attenuation, noise, and other impairments endanger the integrity of the data. To achieve greater distances, repeaters are used. A repeater receives the digital signal, recovers the pattern of ls and 0s, and retransmits a new signal. Thus the attenuation is overcome.

The same technique may be used with an analog signal if it is assumed that the signal carries digital data. At appropriately spaced points, the transmission system has repeaters rather than amplifiers. The repeater recovers the digital data from the analog signal and generates a new, clean analog signal. Thus noise is not cumulative.

The question naturally arises as to which is the preferred method of transmission. The answer being supplied by the telecommunications industry and its customers is digital. Both long-haul telecommunications facilities and intra-building services have moved to digital transmission and, where possible, digital signaling techniques. The most important reasons are as follows:

- **Digital technology:** The advent of large-scale-integration (LSI) and very-large scale integration (VLSI) technology has caused a continuing drop in the cost and size of digital circuitry. Analog equipment has not shown a similar drop.

- **Data integrity:** With the use of repeaters rather than amplifiers, the effects of noise and other signal impairments are not cumulative. Thus it is possible to transmit data longer distances and over lesser quality lines by digital means while maintaining the integrity of the data.

- **Capacity utilization:** It has become economical to build transmission links of very high bandwidth, including satellite channels and optical fiber. A high degree of multiplexing is needed to utilize such capacity effectively, and this is more easily and cheaply achieved with digital (time-division) rather than analog (frequency-division) techniques.

- **Security and privacy:** Encryption techniques can be readily applied to digital data and to analog data that have been digitized.

- **Integration:** By treating both analog and digital data digitally, all signals have the same form and can be treated similarly. Thus economies of scale and convenience can be achieved by integrating voice, video, and digital data.

1.3 Data encoding

In the previous section a distinction was made between analog and digital data and analog and digital signals. We develop further the process involved. For digital signaling, a data source $g(t)$, which may be either digital or analog, is encoded into a digital signal $x(t)$. The actual form of $x(t)$ depends on the encoding technique and is chosen to optimize use of the transmission medium. For example, the encoding may be chosen to conserve bandwidth or to minimize errors.

The basis for analog signaling is a continuous constant-frequency signal known as the carrier signal. The frequency of the carrier signal is chosen to be compatible with the transmission medium being used. Data may be transmitted using a carrier signal by modulation. Modulation is the process of encoding source data onto a carrier signal with frequency f_c. All modulation techniques involve operation on one of more of the three fundamental frequency-domain parameters: amplitude, frequency, and phase. The input signal $m(t)$ may be analog or digital and is called the modulating signal or baseband signal. The result of modulating the carrier signal is called the modulated signal $s(t)$ which is a bandlimited (bandpass) signal. The location of the bandwidth on the spectrum is related to f_c, and is often centered on f_c. Again, the actual form of the encoding is chosen to optimize some characteristic of the transmission. Each of the four possible combinations described hereafter is in widespread use. The reasons for choosing a particular combination for any given communication task vary. We list here some representative reasons:

• **Digital data, digital signal:** In general, the equipment for encoding digital data into a digital signal is less complex and less expensive than digital-to-analog modulation equipment.

• **Analog data, digital signal:** Conversion of analog data to digital form permits the use of modern digital transmission and switching equipment.

• **Digital data, analog signal:** Some transmission media, such as optical fiber and unguided media, will only propagate analog signals.

• **Analog data, analog signal:** Analog data in electrical form can be transmitted as baseband signals easily and cheaply. This is done with voice transmission over voice-grade lines. One common use of modulation is to shift the bandwidth of a baseband signal to another portion of the spectrum. In this way multiple signals, each at a different position on the spectrum, can share the same transmission medium. This is known as frequency-division multiplexing.

We now examine in some detail the techniques involved in each of these four combinations.

1.3.1 Digital data, digital signals

A digital signal is a sequence of discrete, discontinuous voltage pulses. Each pulse is a signal element. Binary data are transmitted by encoding each data bit into signal elements. In the simplest case, there is a one-to-one correspondence between bits and signal elements. An example is shown in Figure B2T12, in which binary 0 is represented by a lower voltage level and binary 1 by a higher voltage level. We show in this section that a variety of other encoding schemes are also used.

First, we define some terms. If the signal elements all have the same algebraic sign (that is, all positive or negative), then the signal is unipolar. In polar signaling, one logic state is represented by a positive voltage level, and the other by a negative voltage level. The data signaling rate, or just data rate, of a signal is the rate, in bits per second, that data are transmitted. The duration or length of a bit is the amount of time it takes for the transmitter to emit the bit; for a data rate R, the bit duration is 1/R. The modulation rate, in contrast, is the rate at which the signal level is changed. This will depend on the nature of the digital encoding, as explained later. The modulation rate is expressed in baud, which means signal elements per second. Finally, the terms *mark* and *space*, for historical reasons, refer to the binary digits 1 and 0, respectively.

The encoding scheme is an important factor that can be used to improve performance. The encoding scheme is simply the mapping from data bits to signal elements. A variety of approaches have been tried. In what follows, we describe some of the more common ones; they are defined in B2T11 and depicted in B2T12. Before describing these techniques, let us consider the following ways of evaluating or comparing the various techniques:

- **Signal spectrum:** Several aspects of the signal spectrum are important. A lack of high-frequency components means that less bandwidth is required for transmission. In addition, lack of a direct-current (dc) component is also desirable. With a dc component to the signal, there must be direct physical attachment of transmission components. With no dc component, ac coupling via transformer is possible; this provides excellent electrical isolation, reducing

interference. Finally, the magnitude of the effects of signal distortion and interference depend on the spectral properties of the transmitted signal. In practice, it usually happens that the transfer function of a channel is worse near the band edges. Therefore, a good signal design should concentrate the transmitted power in the middle of the transmission bandwidth. In such a case, a smaller distortion should be present in the received signal. To meet this objective, codes can be designed with the aim of shaping the spectrum of the transmitted signal.

- **Clocking:** There is a need to determine the beginning and end of each bit position. This is no easy task. One rather expensive approach is to provide a separate clock lead to synchronize the transmitter and receiver. The alternative is to provide some synchronization mechanism that is based on the transmitted signal. This can be achieved with suitable encoding.

- **Error detection:** Error-detection is the responsibility of a layer of logic above the signaling level that is known as data link control, However, it is useful to have some error-detection capability built into the physical signaling encoding scheme. This permits errors to be detected more quickly.

- **Signal interference and noise immunity:** Certain codes exhibit superior performance in the presence of noise. This is usually expressed in terms of a BER[17].

- **Cost and complexity:** Although digital logic continues to drop in price, this factor should not be ignored. In particular, the higher the signaling rate to achieve a given data rate, the greater the cost. We will see that some codes require a signaling rate that is in fact greater than the actual data rate.

We now turn to a discussion of various encoding techniques.

Nonreturn to Zero (NRZ)

The most common, and easiest, way to transmit digital signals is to use two different voltage levels for the two binary digits. Codes that follow this strategy share the property that the voltage level is constant during a bit interval; there is no transition (no return to a zero voltage level). For example, the absence of voltage can be used to represent binary 0, with a constant positive voltage used to represent binary 1. More commonly, a negative voltage is used to represent one binary value and a positive voltage is used to represent the other. This latter code, known as Nonreturn to Zero-Level (NRZ-L), is generally the code used to generate or interpret digital data by terminals and other devices. If a different code is to be used for transmission, it is typically generated from an NRZ-L signal by the transmission system.

A variation of NRZ is known as NRZI (Nonreturn to Zero, invert on ones). As with NRZ-L, NRZI maintains a constant voltage pulse for the duration of a bit time. The data themselves are encoded as the presence or absence of a signal transition at the beginning of the bit time. A transition (low to high or high to low) at the beginning of a bit time denotes a binary 1 for that bit time; no transition indicates a binary 0.

NRZI is an example of differential encoding. In differential encoding, the information to be transmitted is represented in terms of the changes between successive data symbols rather than the signal elements themselves. In general, the encoding of the current bit is determined as follows: If the current bit is a binary 0, then the current bit is encoded with the same signal as the preceding bit; If the current bit is a binary 1, then the current bit is encoded with a different signal than the preceding bit. One benefit of differential encoding is that it may be more reliable to detect a transition in the presence of noise than to compare a value to a threshold. Another benefit is that with a complex transmission layout, it is easy to lose the sense of the polarity of the signal. For example, on a multidrop twisted-pair line, if the leads from an attached device to the twisted pair are accidentally inverted, all 1s and 0s for NRZ-L will be inverted. This cannot happen with differential encoding.

The NRZ codes are the easiest to engineer and, in addition, make efficient use of bandwidth. In fact, most of the energy in NRZ and NRZI signals is between dc and half the bit rate. For example, if an NRZ code is used to generate a signal with data rate of 9600 bps, most of the energy in the signal is concentrated between dc and 4800 Hz.

The main limitations of NRZ signals are the presence of a dc component and the lack of synchronization capability. To picture the latter problem, consider that with a long string of 1s or 0s for NRZ-L or a long string of 0s for NRZI, the output is a constant voltage over a long period of time. Under these circumstances, any drift between the timing of transmitter and receiver will result in loss of synchronization between the two.

Because of their simplicity and relatively low-frequency response characteristics, NRZ codes are commonly used for

[17] The BER or *bit error ratio* is the most common measure of error performance on a data circuit and is defined as the probability that a bit is received in error.

digital magnetic recording. However, their limitations make these codes unattractive for signal transmission applications.

Multilevel Binary

A category of encoding techniques known as multilevel binary address some of the deficiencies of the NRZ codes. These codes use more than two signal levels. Two examples of this scheme are bipolar-AMI (alternate mark inversion) and pseudoternary.

In the case of the bipolar-AMI scheme, a binary 0 is represented by no line signal, and a binary 1 is represented by a positive or negative pulse. The binary 1 pulses must alternate in polarity. There are several advantages to this approach. First, there will he no loss of synchronization if a long string of 1s occurs. Each 1 introduces a transition, and the receiver can resynchronize on that transition. A long string of 0s would still he a problem. Second, because the 1 signals alternate in voltage from positive to negative, there is no net dc component. Also, the bandwidth of the resulting signal is considerably less than the bandwidth for NRZ. Finally, the pulse alternation property provides a simple means of error detection. Any isolated error, whether it deletes a pulse or adds a pulse, causes a violation of this property.

The comments of the previous paragraph also apply to pseudoternary. In this case, it is the binary 1 that is represented by the absence of a line signal, and the binary 0 by alternating positive and negative pulses. There is no particular advantage of one technique versus the other, and each is the basis of some applications.

Although a degree of synchronization is provided with these codes, a long string of 0s in the case of AMI or 1s in the case of pseudoternary still presents a problem. Several techniques have been used to address this deficiency. One approach is to insert additional bits that force transitions. This technique is used in ISDN for relatively low data rate transmission. Of course, at a high data rate, this scheme is expensive, because it results in an increase in an already high signal transmission rate. To deal with this problem at high data rates, a technique that involves scrambling the data is used. We examine one example of this technique later in this section.

Thus, with suitable modification, multilevel binary schemes overcome the problems of NRZ codes. Of course, as with any engineering design decision, there is a tradeoff. With multilevel binary coding, the line signal may take on one of three levels, but each signal element, which could represent $\log_2 3 = 1.58$ bits of information, bears only one bit of information. Thus multilevel binary is not as efficient as NRZ coding. Another way to state this is that the receiver of multilevel binary signals has to distinguish between three levels (+A, -A, 0) instead of just two levels in the signaling formats previously discussed. Because of this, the multilevel binary signal requires approximately 3 dB more signal power than a two-valued signal for the same probability of bit error. Put another way, the bit error rate for NRZ codes, at a given signal-to-noise ratio, is significantly less than that for multilevel binary.

Biphase

There is another set of coding techniques, grouped under the term *biphase*, that overcomes the limitations of NRZ codes. Two of these techniques, Manchester and Differential Manchester, are in common use.

In the Manchester code, there is a transition at the middle of each bit period. The midbit transition serves as a clocking mechanism and also as data: a low-to-high transition represents a 1, and a high-to-low transition represents a 0. In Differential Manchester, the midbit transition is used only to provide clocking. The encoding of a 0 is represented by the presence of a transition at the beginning of a bit period, and a 1 is represented by the absence of a transition at the beginning of a bit period. Differential Manchester has the added advantage of employing differential encoding.

All of the biphase techniques require at least one transition per bit time and may have as many as two transitions. Thus, the maximum modulation rate is twice that for NRZ; this means that the bandwidth required is correspondingly greater. On the other hand, the biphase schemes have several advantages:

- **Synchronization:** Because there is a predictable transition during each bit time, the receiver can synchronize on that transition. For this reason, the biphase codes are known as selfclocking codes.

- **No dc component:** Biphase codes have no dc component, yielding the benefits described earlier.

- **Error detection:** The absence of an expected transition can be used to detect errors. Noise on the line would have to invert both the signal before and after the expected transition to cause an undetected error.

The bandwidth for biphase codes is reasonably narrow and contains no dc component. However, it is wider than the bandwidth for the multilevel binary codes.

Biphase codes are popular techniques for data transmission. The more common Manchester code has been specified for the IEEE 802.3 standard for baseband coaxial cable and twisted-pair CSMA/CD bus LANs. Differential Manchester has

been specified for the IEEE 802.5 token ring LAN, using shielded twisted pair.

Modulation Rate

When signal encoding techniques are used, a distinction needs to be made between data rate (expressed in bits per second) and modulation rate (expressed in baud). The data rate, or bit rate, is $1/t_B$ where t_B = bit duration. The modulation rate is the rate at which signal elements are generated. Consider, for example, Manchester encoding. The minimum size signal element is a pulse of one-half the duration of a bit interval. For a string of all binary zeroes or all binary ones, a continuous stream of such pulses is generated. Hence the maximum modulation rate for Manchester is $2/t_B$. In general, $D = R/b$, where D is the modulation rate in baud, R is the data rate in bps and b is the number of bits per signal element.

One way of characterizing the modulation rate is to determine the average number of transitions that occur per bit time. In general, this will depend on the exact sequence of bits being transmitted. Table B2T14 compares transition rates for various techniques. It indicates the signal transition rate in the case of a data stream of alternating 1s and 0s and for the data stream that produces the minimum and maximum modulation rate.

Scrambling Techniques

Although the biphase techniques have achieved widespread use in local area network applications at relatively high data rates (up to 10 Mbps), they have not been widely used in long-distance applications. The principal reason for this is that they require a high signaling rate relative to the data rate. This sort of inefficiency is more costly in a long-distance application.

Another approach is to make use of some sort of scrambling scheme. The idea behind this approach is simple: Sequences that would result in a constant voltage level on the line are replaced by filling sequences that will provide sufficient transitions for the receiver's clock to maintain synchronization. The filling sequence must be recognized by the receiver and replaced with the original data sequence. The filling sequence is the same length as the original sequence, so there is no data rate increase. The design goals for this approach can be summarized as follows:

- No dc component
- No long sequences of zero-level line signals
- No reduction in data rate
- Error-detection capability

We will describe one technique commonly used in Europe and Japan for long-distance transmission services known as high-density bipolar-3 zeros (HDB3) code. It is based on the use of AMI encoding. In this case, the scheme replaces strings of four zeros with sequences containing one or two pulses. In each case, the fourth zero is replaced with a code violation. In addition, a rule is needed to ensure that successive violations are of alternate polarity so that no dc component is introduced. Thus, if the last violation was positive, this violation must be negative and vice versa. This condition is tested for by determining (1) whether the number of pulses since the last violation is even or odd and (2) the polarity of the last pulse before the occurrence of the four zeros.

HDB-3 has no dc component. Most of the energy is concentrated in a relatively sharp spectrum around a frequency equal to one-half the data rate. Thus, this code is well suited to high data rate transmission.

1.3.2 Digital data, Analog signals

We turn now to the case of transmitting digital data using analog signals. The most familiar use of this transformation is for transmitting digital data through the public telephone network. The telephone network was designed to receive, switch, and transmit analog signals in the voice-frequency range of about 300 to 3400 Hz. Digital devices are usually attached to the network via a modem (modulator-demodulator), which converts digital data to analog signals, and vice versa.

For the telephone network, modems are used that produce signals in the voice-frequency range. The same basic techniques are used for modems that produce signals at higher frequencies (e.g., microwave). This section introduces these techniques and provides a brief discussion of the performance characteristics of the alternative approaches.

Encoding Techniques

We mentioned that modulation involves operation on one or more of the three characteristics of a carrier signal:

amplitude, frequency, and phase. Accordingly, there are three basic encoding or modulation techniques for transforming digital data into analog signals.

• Amplitude-shift keying (ASK)

• Frequency-shift keying (FSK)

• Phase-shift keying (PSK)

In all these cases, the resulting signal occupies a bandwidth centered on the carrier frequency.

In ASK, the two binary values are represented by two different amplitudes of the carrier frequency. Commonly, one of the amplitudes is zero; that is, one binary digit is represented by the presence, at constant amplitude, of the carrier, the other by the absence of the carrier. The resulting signal is

$s(t) = A \times \cos(2\pi f_c t)$ for binary 1, and

0 for binary 0

where the carrier signal is $A \times \cos(2\pi f_c t)$. ASK is susceptible to sudden gain changes and is a rather inefficient modulation technique. On voice-grade lines, it is typically used only up to 1200 bps. The ASK technique is used to transmit digital data over optical fiber. For LED[18] transmitters, the preceding equation is valid. That is, one signal element is represented by a light pulse while the other signal element is represented by the absence of light. Laser transmitters normally have a fixed "bias" current that causes the device to emit a low light level. This low level represents one signal element, while a higher-amplitude lightwave represents another signal element.

In FSK, the two binary values are represented by two different frequencies near the carrier frequency. The resulting signal is

$s(t) = A \times \cos(2\pi f_1 t)$ for binary 1, and

$A \times \cos(2\pi f_2 t)$ for binary 0

where f_1 and f_2 are typically offset from the carrier frequency f_c by equal but opposite amounts. FSK is less susceptible to error than ASK. On voice-grade lines, it is typically used up to 1200 bps. It is also commonly used for high-frequency (3 to 30 MHz) radio transmission. It can also be used at even higher frequencies on local area networks that use coaxial cable.

In PSK, the phase of the carrier signal is shifted to represent data. In this system, a binary 0 is represented by sending a signal burst of the same phase as the previous signal burst sent. A binary 1 is represented by sending a signal burst of opposite phase to the preceding one. This is known as differential PSK, because the phase shift is with reference to the previous bit transmitted rather than to some constant reference signal. The resulting signal is

$s(t) = A \times \cos(2\pi f_c t + \pi)$ for binary 1, and

$A \times \cos(2\pi f_c t)$ for binary 0

with the phase measured relative to the previous bit interval.

More efficient use of bandwidth can be achieved if each signaling element represents more than one bit. For example, instead of a phase shift of 180°, as allowed in PSK, a common encoding technique, known as quadrature phase-shift keying (QPSK), uses phase shifts of multiples of $\pi/2$ (90°). The resulting signal is:

$s(t) = A \times \cos(2\pi f_c t + \pi/4)$ for binary 11,

$A \times \cos(2\pi f_c t + 3\pi/4)$ for binary 10,

$A \times \cos(2\pi f_c t + 5\pi/4)$ for binary 00, and

$A \times \cos(2\pi f_c t + 7\pi/4)$ for binary 01.

[18] Light Emitting Diode

Thus each signal element represents two bits rather than one. This scheme can be extended. It is possible to transmit bits three at a time using eight different phase angles. Further, each angle can have more than one amplitude. For example, a standard 9600 bps modem uses 12 phase angles, four of which have two amplitude values.

This latter example points out very well the difference between the data rate R (in bps) and the modulation rate D (in baud) of a signal. Let us assume that this scheme is being employed with NRZ-L digital input. The data rate is $R = 1/t_B$, where t_B is the width of each NRZ-L bit. However, the encoded signal contains $b = 4$ bits in each signal element using L = 16 different combinations of amplitude and phase. The modulation rate can be seen to be $R/4$ because each change of signal element communicates four bits. Thus the line signaling speed is 2400 baud, but the data rate is 9600 bps. This is the reason that higher bit rates can be achieved over voice-grade lines by employing more complex modulation schemes.

In general, $D = R/b = R/\log_2 L$, where D is the modulation rate in baud, R is the data rate in bps, L is the number of different signal elements, and b is the number of bits per signal element. The foregoing is complicated when an encoding technique other than NRZ is used. For example, we saw that the maximum modulation rate for biphase signals is $2/t_B$. Thus D for biphase is greater than D for NRZ. This to some extent counteracts the reduction in D that occurs using multilevel signal modulation techniques.

1.3.3 Analog data, digital signals

In this section we examine the process of transforming analog data into digital signals. Strictly speaking, it might be more correct to refer to this as a process of converting analog data into digital data; this process is known as digitization. Once analog data have been converted into digital data, a number of things can happen. The three most common are as follows:

1. The digital data can be transmitted using NRZ-L In this case, we have in fact gone directly from analog data to a digital signal.

2. The digital data can be encoded as a digital signal using a code other than NRZ-L Thus an extra step is required.

3. The digital data can be converted into an analog signal, using one of the modulation techniques discussed in Section 1.3.2.

This last, seemingly curious, procedure may be applied to (analog) voice data that are digitized and then converted to an analog ASK signal. This allows digital transmission in the sense defined in section 1.2. The voice data, because they have been digitized, can he treated as digital data, even though transmission requirements (e.g., use of microwave) dictate that an analog signal be used. The device used for converting analog data into digital form for transmission, and subsequently recovering the original analog data from the digital, is known as a codec (coder-decoder). In this section we examine the two principal techniques used in codecs pulse code modulation and delta modulation. The section closes with a discussion of comparative performance.

Pulse Code Modulation

Pulse code modulation (PCM) is based on the sampling theorem, which states the following:

If a signal *f(t)* is sampled at regular intervals of time and at a rate higher than twice the highest signal frequency, then the samples contain all the information of the original signal. The function *f(t)* may be reconstructed from these samples by the use of a lowpass filter.

If voice data are limited to frequencies below 4000 Hz, a conservative procedure for intelligibility, 8000 samples per second would be sufficient to characterize the voice signal completely. Note, however, that these are analog samples, called pulse amplitude modulation (PAM) samples. To convert to digital, each of these analog samples must be assigned a binary code. Each sample can be approximated by being "quantized" into one of N different levels. Each sample can then be represented by $\log_2 N$ bits. But because the quantized values are only approximations, it is impossible to recover the original signal exactly. By using an 8-bit sample, which allows 256 quantizing levels, the quality of the recovered voice signal is comparable with that achieved via analog transmission. Note that this implies that a data rate of 8000 samples per second × 8 bits per sample = 64 kbps is needed for a single voice signal.

Typically, the PCM scheme is refined using a technique known as nonlinear encoding, which means, in effect, that the quantization levels are not equally spaced. The problem with equal spacing is that the mean absolute error for each sample is the same, regardless of signal *level.* Consequently, lower amplitude values are relatively more distorted. By using a greater number of quantizing steps for signals of low amplitude, and a smaller number of quantizing steps for signals of large amplitude, a marked reduction in overall signal distortion is achieved. Two choices of quantization levels are in common use, μ-law in the USA and Japan and A-law in the rest of the world.

Delta Modulation (DM)

A variety of techniques have been used to improve the performance of PCM or to reduce its complexity. One of the most popular alternatives to PCM is delta modulation (DM).

With delta modulation, an analog input is approximated by a staircase function that moves up or down by one quantization level (δ) at each sampling interval (T_s). The important characteristic of this staircase function is that its behavior is binary: At each sampling time, the function moves up or down a constant amount δ. Thus, the output of the delta modulation process can be represented as a single binary digit for each sample. In essence, a bit stream is produced by approximating the derivative of an analog signal rather than its amplitude: A 1 is generated if the staircase function is to go up during the next interval; a 0 is generated otherwise.

The transition (up or down) that occurs at each sampling interval is chosen so that the staircase function tracks the original analog waveform as closely as possible. For transmission, the following occurs: At each sampling time, the analog input is compared to the most recent value of the approximating staircase function. If the value of the sampled waveform exceeds that of the staircase function, a 1 is generated; otherwise, a 0 is generated. Thus, the staircase is always changed in the direction of the input signal. The output of the DM process is therefore a binary sequence that can be used at the receiver to reconstruct the staircase function. The staircase function can then be smoothed by some type of integration process or by passing it through a low-pass filter to produce an analog approximation of the analog input signal.

There are two important parameters in a DM scheme: the size of the step assigned to each binary digit, δ, and the sampling rate. δ must be chosen to produce a balance between two types of errors or noise. When the analog waveform is changing very slowly, there will be quantizing noise. This noise increases as δ is increased. On the other hand, when the analog waveform is changing more rapidly than the staircase can follow, there is slope overload noise. This noise increases as δ is decreased.

It should be clear that the accuracy of the scheme can be improved by increasing the sampling rate. However, this increases the data rate of the output signal. The principal advantage of DM over PCM is the simplicity of its implementation. In general, PCM exhibits better SNR characteristics at the same data rate.

1.3.4 Analog data, analog signals

Modulation has been defined as the process of combining an input signal $m(t)$ and a carrier at frequency f_c to produce a signal $s(t)$ whose bandwidth is (usually) centered on f_c. For digital data, the motivation for modulation should be clear: When only analog transmission facilities are available, modulation is required to convert the digital data to analog form. The motivation when the data are already analog is less clear. After all, voice signals are transmitted over telephone lines at their original spectrum (referred to as baseband transmission). There are two principal reasons for analog modulation of analog signals:

- A higher frequency may be needed for effective transmission. For unguided transmission, it is virtually impossible to transmit baseband signals; the required antennas would be many kilometers in diameter.

- Modulation permits frequency-division multiplexing, an important technique explored in the previous bloc.

In this section we look at the principal techniques for modulation using analog data: amplitude modulation (AM), frequency modulation (FM), and phase modulation (PM). As before, the three basic characteristics of a signal are used for modulation.

Amplitude Modulation

Amplitude modulation (AM) is the simplest form of modulation. Mathematically, the process can be expressed as $s(t) = [1 + n_a x(t)]\cos 2\pi f_c t$, where $\cos 2\pi f_c t$ is the carrier and $x(t)$ is the input signal (carrying data), both normalized to unity amplitude. The parameter n_a, known as the modulation index, is the ratio of the amplitude of the input signal to the carrier. Corresponding to our previous notation, the input signal is $m(t) = n_a x(t)$. The "1" in the preceding equation is a dc component that prevents loss of information, as explained subsequently. This scheme is also known as double sideband transmitted carrier (DSBTC).

Angle Modulation

Frequency modulation (FM) and phase modulation (PM) are special cases of angle modulation. The modulated signal is expressed as $s(t) = A_c \cos[2\pi f_c t + \phi(t)]$.

For phase modulation, the phase is proportional to the modulating signal: $\phi(t) = n_p m(t)$, where n_p *is* the phase

modulation index.

For frequency modulation, the derivative of the phase is proportional to the modulating signal: $\phi'(t) = n_f m(t)$, where n_f *is* the frequency modulation index.

Quadrature Amplitude Modulation

QAM is a popular analog signaling technique that is used in the asymmetric digital subscriber line (ADSL). This modulation technique is a combination of amplitude and phase modulation. QAM takes advantage of the fact that it is possible to send two different signals simultaneously on the same carrier frequency, by using two copies of the carrier frequency, one shifted by 90° with respect to the other. For QAM each carrier is ASK modulated. The two independent signals are simultaneously transmitted over the same medium. At the receiver, the two signals are demodulated and the results combined to produce the original binary input.

1.3.5 Summary

Both analog and digital information can be encoded as either analog or digital signals. The particular encoding that is chosen depends on the specific requirements to he met and the media and communications facilities available.

- Digital data, digital signals: The simplest form of digital encoding of digital data is to assign one voltage level to binary one and another to binary zero. More complex encoding schemes are used to improve performance, by altering the spectrum of the signal and providing synchronization capability.

- Digital data, analog signal: A modem converts digital data to an analog signal so that it can be transmitted over an analog line. The basic techniques are amplitude-shift keying (ASK), frequency-shift keying (FSK), and phase-shift keying (PSK). All involve altering one or more characteristics of a carrier frequency to represent binary data.

- Analog data, digital signals: Analog data, such as voice and video, are often digitized to be able to use digital transmission facilities. The simplest technique is pulse code modulation (PCM), which involves sampling the analog data periodically and quantizing the samples.

- Analog data, analog signals: Analog data are modulated by a carrier frequency to produce an analog signal in a different frequency band, which can be utilized on an analog transmission system. The basic techniques are amplitude modulation (AM), frequency modulation (FM), and phase modulation (PM).

2.0 Multiple Access

2.1 Introduction

Imagine an audioconference where voices get garbled if two or more participants speak simultaneously. How should the participants coordinate their actions so that people are not always talking over each other? A simple *centralized* solution is for a moderator to poll each participant in turn, asking whether he or she wants to speak. Unfortunately, with this solution, a participant who wants to speak must wait for the moderator's poll, even if nobody else has anything to say. Moreover, if the moderator's connection fails, no one can speak until the problem is detected and corrected. A better *distributed* solution is for a participant to start speaking as soon as someone else stops, hoping that no one else is about to do the same. This avoids the time wasted in polling and is robust against failed connections, but two speakers waiting for a third speaker to stop speaking are guaranteed to *collide* with each other. If collisions are not carefully resolved, the meeting will make no progress, because the two speakers could repeatedly collide. Thus, both solutions have their faults. Designing coordination schemes that simultaneously minimize collisions and a speaker's expected waiting time is surprisingly hard. In the literature, this is called the *multiple-access* problem.

The multiple-access problem arises when a transmission medium *is broadcast*, so that messages from one endpoint (usually called a *station* in the literature) can be heard by every other station in its listening area. Thus, if two stations in each other's listening area send a message simultaneously, both messages are garbled and lost. Our goal is to devise a scheme (or *protocol)* that maximizes the number of messages that can be carried per second, simultaneously minimizing a station's waiting time. This abstract problem manifests itself in many systems that outwardly appear different. Thus, before we study the abstract multiple-access problem, it is worth looking at some physical contexts in which it arises (see B2T23 and B2T24).

2.1.1 Contexts for the multiple-access problem

The multiple-access problem appears in five main contexts (in each network, the medium is broadcast, so stations in the same listening area must coordinate with each other to share access to the medium) :

- *Wired local area networks:* The most common context for multiple access is in a local area network, where multiple stations share the "ether" in a coaxial cable or a copper wire. Two common solutions to the problem are Ethernet and FDDI (Fiber Distributed Data Interface).

- *Wireless local area networks:* These are increasingly popular replacements for wired LANs, where we replace the coaxial cable with an infrared or radio-frequency link. As with a coaxial cable, a radio link is inherently broadcast, thus requiring a multiple-access protocol. A good example of a wireless LAN is the one described in the IEEE 802.11 standard.

- *Packet radio:* Packet-radio networks replace one or more links in a traditional wired network with radio links. They are the wide-area analogs of a wireless LAN. Metricom's Ricochet service is an example of an operational packet-radio network in the USA.

- *Cellular telephony:* Cellular telephones share the radio spectrum when transmitting to a base station and, therefore, must coordinate access with other cellular telephones in the same service area. Analog and digital cellular telephony service is provided by most major telephone companies around the world.

- *Satellite communication:* Low-earth and geostationary satellites typically rebroadcast messages that they receive from any of the earth stations. Data rates of up to 500 Mbps are possible, though the station-to-station propagation delay for geosynchronous satellites is at least 240 ms. Satellite data service and telephony is provided by networks such as INMARSAT.

In this section, we will study both the abstract multiple-access problem and its instantiation in the five networks just described.

2.1.2 Rest of the section

We usually solve the multiple-access problem in two steps. The first step is to choose a "base technology" to isolate traffic from different stations. This isolation can be in the time domain (stations go one after another), or in the frequency domain (different stations use different frequencies), or in both. Unfortunately, we rarely have enough time slots or enough frequencies to be able to exclusively dedicate one or the other to each station. Thus, in the second step, the multiple access-scheme determines the allocation of transmission resources (in the time or frequency domain) to individual stations. We will study base technologies for multiple access in subsection 2.3 and multiple access schemes

in subsections 2.4 and 2.5.

Before we study base technologies, we make a small diversion, in Section 3.2, to discuss the fundamental choices in designing multiple-access schemes, and some constraints that might affect the design.

2.2 Choices and constraints

2.2.1 Choices

The designer of a multiple-access scheme has two fundamental design choices. These are between a centralized and a distributed design, and between a circuit-mode and a packet-mode design. We discuss these choices next.

Centralized versus distributed communication

In many situations, one station is a "master" station, and all others are "slaves" in the sense that the master decides when a slave is allowed to send data. This is also called a *centralized* design. The master-to-slave downlink may have different physical properties than the slave-to-master uplink. An example is in cellular telephony, where the base station acts as a master to coordinate the actions of the cellular mobile telephones.

With a *distributed* design, no station is a master, and every station is free to talk to every other station. An example is in wired LANs such as Ethernet, where all stations are peers. Multiple-access algorithms are usually optimized to exploit one or the other situation.

Circuit-mode versus packet-mode transfer

The second choice in the design depends on the workload being carried. Note that the multiple-access problem occurs both in the telephone network and in local area networks that carry data traffic. Thus, solutions to the problem must deal not only with smooth, continuous traffic streams, characteristic of voice, but also with *bursty* data traffic streams, where the transmission rate in some time intervals is much higher than in others. If stations generate a steady stream of packets, then it makes sense to allocate part of the link to the source for its exclusive use, thus avoiding the need to negotiate link access for every packet in the stream. If, on the other hand, sources generate bursty traffic, negotiating link access on a packet-by-packet basis is more efficient. We call the first mode of link operation *circuit-mode* and the second *packet-mode*. Circuit and packet-mode access are linked in an interesting way: packets containing circuit reservation requests contend for link access using packet mode. Once this packet-mode access succeeds, subsequent data transmission on the circuit uses circuit mode.

In a meeting, circuit mode corresponds to the situation where each participant makes a long speech. Thus, it makes sense to spend some time determining the order and duration of each speech. Packet mode is more like a free-for-all brainstorming session, where participants have only a one- or two-sentence thought to share. It therefore does not make sense to choose any particular order in which participants speak.

We will study centralized schemes in Section 2.4 and distributed-access schemes in Section 2.5.

2.2.2 Constraints

The design of a multiple-access scheme is often highly constrained by its implementation environment. In this subsection, we discuss three common constraints: spectrum scarcity, radio link impairments, and the parameter *"a"*. The first two are important for radio links, which are used in wireless LANs, packet radio, and cellular telephony. The third plays an important role in packet-mode networks.

Spectrum scarcity

The electromagnetic spectrum, which is used for all radio-frequency wireless communication, is a scarce resource. National and international laws allow only a few frequencies to be used for radio links that span 1 to 10 miles. For example, in the United States, the FCC allows unlicensed data communication in this range only in the 902-928 MHz and 2.40-2.48 GHz bands (also called the Industrial, Scientific, and Medical or *ISM* bands). Thus, multiple-access schemes using radio-frequency links are tightly constrained by the available spectrum.

Radio link properties

A radio link is subject to many impairments. *Fading is* a spatial phenomenon that refers to the degradation of a received signal because of the environment, such as a hill, dense foliage, or a passing truck. *Multipath interference* occurs when a receiver's antenna receives the same signal along multiple paths, which mutually interfere. These two effects cause long periods of bit errors, which must be compensated for using error-control schemes, such as those discussed in bloc

4.

Besides these two common impairments, two other properties of a radio link affect multiple-access protocols. The first problem, called the *hidden-terminal* problem, occurs when a station can be heard only by a subset of receivers in the listening area. For example, in a master-slave configuration, the master may hear all the slaves, but transmissions from some slaves may not be heard by other slaves. Thus, a slave sensing the transmission medium may think that the medium is idle, while, in fact, the medium is occupied by another slave. Multiple-access schemes must deal with the hidden-terminal problem.

The second problem with radio links, called the *capture* or the *near-far* problem, occurs when a station receives messages simultaneously from two other stations, but the signal from one of them drowns out the other, so that no collision occurs. The station with the higher received power is said to have captured the receiver. Capture is good in that it reduces the time wasted on resolving collisions. On the other hand, a transmitter whose signals happen to be received weakly at other stations may be completely shut out of the medium. To prevent starvation, a multiple-access scheme should give such transmitters a chance to transmit by somehow shutting off competing sources.

The parameter "a"

The performance of a packet-mode multiple-access scheme is heavily influenced by a parameter known in the literature as *"a"* and defined as follows. Let:

D = maximum propagation delay between any two stations (in seconds)

T = time taken to transmit an average size packet (in seconds)

Then, $a = D/T$

Intuitively, a is the number of packets (or fraction of a single packet) that a transmitting station can place on the medium before the station farthest away receives the first bit. When a is small ($<<1$), propagation delay is a small fraction of the packet transmission time. Thus, every station in the network receives at least part of a packet before the transmitting station finishes its transmission. a is usually small (around 0.01) for wired and wireless LANs, cellular telephony, and packet radio. However, as the speed of a link increases, T decreases, increasing a. Thus, with faster LANs, a can be on the order of 1 (in other words, a sender may send a whole packet before some receiver sees the first bit of the packet). When a is large, a source may have sent several packets before a receiver sees the first bit. This is true mostly for satellite links, where a can be as large as 100.

The value of a determines what happens when two stations send a message simultaneously. When a is small, their messages collide soon and get garbled. Thus, if both sources listen to the medium, they soon realize that the other is active. If they pause, somehow resolve who goes first, then try again, they will not waste much time. When a is large, a collision affecting a packet from a station happens substantially *after* its transmission (Figure B2T29[19]). Thus, a station does not know for a fairly long time if its transmission was successful. In this case, just sensing the medium is not sufficient to recover from collisions. It makes sense to impose some more structure on the problem, so that the stations *avoid* collisions. We will see the influence of a on the design of multiple-access solutions throughout this bloc.

2.2.3 Performance metrics

We measure the performance of a multiple-access scheme in several ways.

- Normalized throughput or *goodput:* This is the fraction of a link's capacity devoted to carrying nonretransmitted packets. *Goodput* excludes time lost to protocol overhead, collisions, and retransmissions. For example, consider a link with a capacity of 1 Mbps. If the mean packet length is 125 bytes, the link can ideally carry 1000 packets/s. However, because of collisions and protocol overheads, a particular multiple-access scheme may allow a peak throughput of only 250 packets/s. The goodput of this scheme, therefore, is 0.25. An ideal multiple-access protocol allows a goodput of 1.0, which means that no time is lost to collisions, idle time, and retransmissions. Real-life protocols have goodputs between 0.1 and 0.95.

[19] In the B2T29 figure, time increases going down the vertical axis, and each vertical line corresponds to a station. Pairs of parallel slanted lines represent packets, where the distance between a pair is the transmission time, and the vertical drop represents the propagation delay. The upper portion of the figure shows a network where "a" is small, so that the transmission time is larger than the propagation delay. We see that a collision between packets from A and B happens after each has transmitted only a part of a packet. The lower portion of the figure shows the situation when "a" is large. The stations detect that their transmissions interfered only several transmission times after sending the packets.

- Mean delay*:* This is the mean time that a packet at a station waits before it is successfully transmitted (including the transmission time). The delay depends not only on the multiple access scheme, but also on the load generated by the stations and characteristics of the medium.

- Stability*:* When a shared link is underloaded, it can carry all the traffic offered to it. As the offered load increases, the chances of a collision increase, and each collision results in wasted time on at least two retransmissions. These retransmissions may, in turn, collide with newly generated packets, causing further retransmissions. When the load increases beyond a threshold, a poorly designed multiple-access scheme becomes unstable, so that practically every access causes a collision, and stations make slow progress. Thus, we define stability to mean that the throughput does not decrease as we increase the offered load. We can show that if an infinite number of uncontrolled stations share a link, no multiple-access scheme is stable. However, if control schemes are carefully chosen that reduce the load when an overload is detected, a finite population can share a link in a stable way even under heavy load.

- Fairness*:* It is intuitively appealing to require a multiple-access scheme to be "fair." There are many definitions for fairness. A minimal definition, also called no-starvation, is that every station should have an opportunity to transmit within a finite time of wanting to do so. A stricter metric is that each contending station should receive an equal share of the transmission bandwidth.

These performance metrics must be balanced by the cost of implementing the scheme. This depends on the hardware and software costs, which depend on the scheme's complexity.

2.3 Base technologies

A "base technology" serves to isolate data from different stations on a shared medium.

The three common base technologies are frequency division multiple access (FDMA), time division multiple access (TDMA), and code division multiple access (CDMA). We study these in subsections 2.3.1, 2.3.2, and 2.3.3, respectively.

2.3.1 FDMA

The simplest base technology, and the one best suited for analog links, is frequency division multiple access, or FDMA. In FDMA, each station is allocated its own frequency band, and these bands are separated by guard bands. Continuing with the meeting metaphor, assume that each participant speaks in a different frequency or pitch, such as bass, contralto, or soprano. Thus, tuning to a frequency band corresponds to tuning to a sentence spoken at a particular pitch. With FDMA, a participant is assigned a speaking pitch. Receivers who want to listen to a particular participant "tune in" to sentences spoken at the corresponding pitch.

FDMA is common in the radio and television broadcast industry, where each radio or TV station has its own channel. FDMA is adequate for these purposes because a listening area, such as a large city, has at most a few hundred transmitters. With cellular telephony, however, a single listening area may contain hundreds of thousands of simultaneously active transmitters. If we were to assign each transmitter a separate frequency, we would need hundreds of thousands of frequency bands, which is infeasible because the electromagnetic spectrum is a scarce resource. Thus, a naive implementation of FDMA cannot support a large user population. To work around the problem, we must reduce transmitter power and spatially reuse frequencies in different geographical zones called *cells*. A direct consequence is that mobile stations that cross cell boundaries must change their transmission frequency, thus requiring a *handoff*. Handoffs are complicated because they require synchronization between a mobile station and two geographically separated base stations. Thus, cellular telephony trades off an increase in system complexity for the ability to support many more users. See B2T36 for an example.

2.3.2 TDMA

In time division multiple access or TDMA, all stations transmit data on the same frequency, but their transmissions are separated in time. Continuing with our meeting metaphor, a TDMA meeting is where all participants speak at the same pitch, but take turns to speak. In TDMA, we divide time into fixed-length or variable-length *slots*, and each station limits its transmission to a single slot. If all stations are time-synchronized, and stations somehow resolve contention for access to a time slot, the single frequency can be shared among many users. TDMA is universally used in computer communication networks, where a slot duration corresponds to the transmission time required for a single packet. Time synchronization is usually achieved by having one of the stations emit a periodic synchronization signal.

Advantages

Compared with an FDMA system, where we partition a single, large frequency band among many stations, a TDMA station uses the entire frequency band, but only for a fraction of the time. Thus, given the same frequency band, a TDMA system and an FDMA system can be shared by roughly the same number of stations. It can be shown, however, that the mean delay to transmit an *N*-bit packet over a TDMA system is lower than in an FDMA system. Besides this, TDMA has several advantages over FDMA. First, unlike FDMA, users can be dynamically allocated different bandwidth shares by giving them fewer or more time slots. Second, a mobile station can use the time slots in which it is not transmitting to measure the received power from multiple base stations, allowing it to request a handoff to the base station with the most received power. Finally, TDMA mobile stations typically require less power than with FDMA, because they can switch their transmitters off except during their time slot. For these reasons, TDMA is a better solution for multiple access than FDMA.

Problems

TDMA, however, is not without its problems. Every station must synchronize to the same time base. Thus, we must devote a significant portion of the bandwidth to transmitting synchronization signals, and even so, we must surround each time slot by guard times to deal with out-of-synch stations. It has been estimated that 20 to 30% of the bandwidth in a TDMA system is wasted on synchronization and guard bit overhead. TDMA does *not* do away with the need for handoffs, since battery-powered mobiles have limited transmission power. Thus, as a mobile moves away from a base station, its transmitted signal weakens, and the mobile must be handed off to a nearer base station. Finally, since a station transmits data in a wider frequency band, the duration of the transmission is smaller than in FDMA. This leads to greater problems with multipath interference, which must be countered with an adaptive equalization circuit in each receiver.

2.3.3 CDMA

Frequency-hopping CDMA

The easiest way to understand code division multiple access or CDMA is to first consider a variant of CDMA called frequency-hopping CDMA or FH/CDMA. An FH/CDMA station transmits at a particular frequency for a short duration, then hops to another frequency, where it transmits for a fixed duration, and then it hops again. If the receiver and the transmitter synchronize hop schedules, the receiver can decipher the transmitter's signal. Continuing with the meeting metaphor, with FH/CDMA, each speaker speaks each sentence in a different pitch. The first sentence may be in a low pitch, the second in a high pitch, and the third in a medium pitch. If a receiver knows the sequence of pitches that the speaker plans to use, he or she can decode the transmission.

With FH/CDMA, two stations can share the same large frequency band if they agree to use a different bandlimited portion of the spectrum each time they hop. A receiver can choose to receive data from one or the other station by choosing the appropriate hopping sequence. Thus, unlike FDMA or TDMA systems, where stations are separated either in time or in frequency, in CDMA, stations can be distinguished only by simultaneously considering *both* time and frequency.

Direct-sequence CDMA

A second technique for CDMA is more complicated, but uses the same general principle of taking the user signal and spreading it over a wider frequency spectrum. In this technique, called direct sequence CDMA or DS/CDMA, each bit of the transmitter is replaced by a *sequence* of bits that occupy the same time duration. For example, if the transmitter wants to send the bit "l," it may normally send a signal corresponding to "1" for 1 second, thus requiring a bandwidth of 1 Hz. With DS/CDMA, the transmitter might send the *codeword* "10110110" during the same second instead, thus requiring a bandwidth of at least 8 Hz. We call each bit in the codeword a *chip,* and measure the bit-rate of the codeword in *chips/second.* We can view this replacement of the single bit by a codeword as "smearing" or *spreading* the single bit over a wider frequency spectrum. The receiver receives the eight bits at the higher frequency, and retrieves the original information using a decorrelator that "unsmears" according to the given codeword. Thus, if a narrowband noise signal corrupts some part of the 8 Hz spectrum, enough information is carried in the rest of the spectrum that the original signal ("1 ") can still be correctly retrieved.

If a second user wanted to use the same 8 Hz spectrum, it could do so by spreading its bits with a different codeword. For example, it might represent a "1" by the code "00101101." We can show that even if both stations sent their codewords simultaneously, a decorrelator tuned to one of the codewords (and time-synchronized with the transmitter) can extract the signal corresponding to that codeword even in the presence of the other codeword. Of course, the codewords must be chosen in a special way to prevent too much confusion at a receiver. These sets of compatible codewords are said to be *orthogonal,* and a great deal of research has gone into creating optimal orthogonal codewords (see an example on B2T40).

As we mentioned before, both FH/CDMA and DS/CDMA systems spread a transmitter's signal over a larger frequency

spectrum. In some sense they use a wider spectrum than is strictly necessary for carrying the transmitter's signal. Thus, we also call them *spread-spectrum* communication techniques.

Advantages

Why would a transmitter want to hop from frequency to frequency or use an entire codeword when a single bit will do? An obvious answer, for those in the spy business, is that it is very hard to eavesdrop on a frequency-hopping transmitter. If the receiver does not know the exact sequence of hops that the transmitter intends to make, or the codeword it uses to spread its bits, it cannot decipher the transmitter's message. Indeed, the first use for CDMA was in the military, presumably for use in the battlefield. However, spread-spectrum communication has other interesting properties that make it attractive for civilians.

First, if the spectrum is subjected to narrowband noise, such as a jamming signal from an enemy, then it does not greatly affect the transmitted signal, since only part of the transmission is in any given frequency band. A transmitter can make its signal immune to narrowband noise by using sufficiently long interleaved error-correcting codes that span over multiple narrowband frequencies (see bloc 4 for more details on interleaving and error-correcting codes). Moreover, it can be shown that a CDMA receiver is less susceptible to errors due to multipath interference than FDMA or TDMA.

Second, unlike TDMA, where all stations must share the same time base, in CDMA, stations can use different time bases. It is sufficient for the receiver and the sender to agree on a time base: because other stations see the effect of the transmission only as noise, they do not need to synchronize with the sender.

Third, CDMA allows adjacent base stations to communicate with the same mobile station if they share a CDMA code or FH sequence[20]. Thus, handoffs between cells can be made "soft" by asking more than one base station to transmit (or receive) using the same CDMA FH sequence or the same CDMA code. This reduces the interruption during a handoff. We cannot do this using FDMA, where adjacent cells are not allowed to use the same frequency, or TDMA, where soft handoff would require time synchronization between cells.

Fourth, unlike a TDMA or FDMA system, a CDMA system has no hard limit on capacity. Each station appears as a source of noise for every other station. Thus, the number of stations can be increased without limit, at the expense of decreasing the effective bit-rate per station.

Fifth, frequency planning in a CDMA system is much easier than in FDMA. Recall that in FDMA, adjacent cells may not use the same set of frequencies. In CDMA, the same frequency band is used in all cells; thus, no frequency planning is needed. This feature is often called universal frequency reuse.

Sixth, unlike FDMA and TDMA, higher-powered CDMA transmitters can be used in the unlicensed ISM bands in the United States. Thus, all products operating in the ISM bands (other than low-power devices) use CDMA.

Finally, in an FDMA or TDMA system, if a station is inactive, the signal-to-noise ratio of the other stations does not increase. In CDMA, a station's silence directly benefits other stations. Thus, a transmitter can increase overall system performance (and decreases its own power consumption) by switching itself off when it has nothing to send. CDMA allows us to take advantage of this common situation.

Problems

CDMA's benefits come at a price. The three major problems with CDMA are (a) its implementation complexity, (b) its need for power management (also called power control), and (c) its need for a large, contiguous frequency band. We discuss these in more detail next.

First, CDMA requires receivers to be perfectly synchronized with transmitters, and introduces substantial computational complexity either for tracking hops (with FH/CDMA) or for decorrelation (with DS/CDMA). This increases the cost of receivers (though this cost is dropping rapidly).

A more insidious problem is power management. If a CDMA receiver receives a low-power signal from an intended. but remote, transmitter, and a higher-power noise signal from a nearer transmitter, it has a hard time recovering the intended signal. Unless nearby transmitters reduce their transmission power, they can completely shut out distant transmitters. The solution is to require each transmitter to vary its power in inverse proportion to its distance from its intended receiver. If each cell has only a single CDMA receiver (as is the case in telephony), and if the path from the

[20] This is allowed if some conditions hold.

transmitter to the receiver is symmetric, then a simple power management technique is for the transmitter to ensure that the sum of its transmission and reception power is constant. This automatically ensures that when a receiver is far away (so that its transmissions are faint), the transmit power is increased (so that the receiver can hear). Conversely, when the receiver is near (so that its transmissions are loud) the transmit power is reduced. If the system has multiple rapidly moving CDMA receivers, the power management problem is much more complicated.

Finally, introducing DS/CDMA into the existing infrastructure is hard because it needs a large, contiguous frequency band. In contrast, FDMA, TDMA, and FH/CDMA can use multiple, non-contiguous frequency bands.

On balance, CDMA seems to have more advantages than disadvantages. As we will see in Section 2.4, many commercial systems are moving from FDMA to TDMA and CDMA.

2.3.4 FDD and TDD

In many situations a pair of stations must simultaneously communicate with each other. For example, a person speaking on a cellular phone may simultaneously want to hear a signal from the other end. Two standard techniques to establish two-way communication between a pair of stations are called frequency division duplex (FDD) and time division duplex (TDD). FDD is analogous to FDMA. in that the two directions use different frequencies. In TDD, we slice a single frequency band into time slots, and each end takes turns in using the time slots.

TDD, FDD, TDMA, FDMA, and CDMA can be simultaneously used to provide several interesting combinations. For example, combining FDD with TDMA, one frequency band is time shared between multiple stations that want to transmit to a base station, and another frequency band is time shared for communication from a base station to several other stations. If the system is master-slave, FDD is very common, with the master-to-slave channel using TDMA (since synchronizing time bases with a single master is easy), and the slave-to-master channel shared using either TDMA or CDMA.

Second-generation cordless phones use a TDD/FDMA scheme, where different phone/base-station pairs use different frequencies, but in a given frequency band, the base station and the phone share time slots for communication. Digital cellular phones use FDD/TDMA/FDMA, where each cell has some number of frequency bands, each frequency band is time-shared using TDMA, and phone-to-base and base-to-phone communications use different frequencies. Digital cordless phones (often advertised as 900MHz phones) use a TDMA/TDD/FDMA scheme, where each phone/base-station pair is assigned not only a particular frequency band, but also a time slot in the frequency band, and the base station and phone take turns in sharing time slots in a single frequency band.

2.4 Centralized access schemes

We now turn our attention to multiple-access schemes that build on the base technologies described in Section 2.3. We will first study schemes where one of the stations is a *master*, and the others are *slaves.* By this, we mean that the master can prevent a slave from transmitting until told to do so. This mode of operation makes multiple access straightforward, since the master provides a single point of coordination. On the other hand, a master-slave system can be less robust because the master is also a single point of failure. Moreover, since the master is involved in every transmission, it adds delay to every message exchange. The reliability issue can be solved by allowing slaves to reelect a master in case the master goes down. However, this complicates the system, something we wanted to avoid with a master-slave arrangement in the first place!

Physical constraints often lead naturally to a master-slave configuration. For example, in a wireless LAN, the base station is the only station guaranteed to be in direct communication with every other station (since it is usually mounted on a ceiling, where it has a clear infrared or radio path to every station). Similarly, in a cellular phone network, the base station is the only one with a wired connection to the network, and it can have a transmission power much greater than any mobile. In these situations, it often makes sense to have the base station also act as the master for a multiple-access scheme.

In the following subsection, we will study three centralized schemes. In subsection 2.4.1, we study circuit-mode schemes, primarily for cellular telephony. In subsections 2.4.2 and 2.4.3, we study two packet-mode schemes: polling and reservation.

2.4.1 Circuit mode

The most common centralized multiple-access scheme for circuit-mode communication is cellular telephony. As we saw in Section 2.3, partitioning a service area into cells allows a provider to increase the number of simultaneous voice calls it can support. Each geographically separated cell has a *base station* that provides access to the wired telephone infrastructure and manages access to the broadcast medium (as was shown in B2T24 (d)).

When a cellular phone is turned on, it sends a registration message to the base station on a control channel. This message contends for access with other phones accessing the control channel using the ALOHA packet-mode multiple access protocol described in subsection 2.5.5. Once the base station correctly receives the registration message, it allocates the cellular phone either a frequency (FDMA), time slot (TDMA), or code (CDMA) that the phone uses for subsequent voice transfer. Since the voice call is carried in circuit mode, once the cellular phone acquires its channel, it never has to contend for access again.

We now outline three common cellular telephone standards, which combine centralized control and circuit-mode operation with one of the base technologies described in Section 2.2. These are EAMPS, the analog telephone technology common in the United States; GSM, a digital cellular technology common in Europe; and IS-95, a proposal for CDMA cellular telephony.

EAMPS

The earliest cellular telephones, and the ones most common in the United States at the time of this writing (1996), use *analog* FDD/FDMA. In the United States, this is called the Extended Advanced Mobile Phone Service or EAMPS (other parts of the world use similar systems, but with different acronyms-for example, TACS in Europe and JTACS in Japan). In EAMPS, mobiles use a 25-MHz band from 824 to 849 MHz, and the base station uses a 25MHz band from 869 to 894 MHz. The large difference in frequency bands (45 MHz) minimizes interference between the transmitter and the receivers. The base station uses the higher of the two bands, since higher-frequency bands are subject to higher propagation losses, and the base station has more transmission power than the mobiles.

GSM

The second generation of cellular phones, currently very popular in Europe, use *digital* TDMA with FDD. The standard system in Europe is the Global System for Mobile Communication or GSM. In GSM, the uplink occupies the 935-960 MHz band, and the downlink the 890-915 MHz band. Each band is divided into a number of 200-KHzwide channels, and each channel is shared among eight users using TDMA. GSM is being adopted around the world, as usual, under a variety of names and frequency allocations. In the United States, a variant of GSM with six-slot 20-ms frames and 30-KHz channels is called IS-54, and it occupies the same frequency bands as EAMPS. In Japan, the GSM variant is called Personal Digital Cellular. A second system that was also introduced in Europe has the same technology as GSM, but is centered around the 1800-MHz band and is, therefore, called DCS-1800 (except in the United Kingdom, where it is called Personal Communications Network).

IS-95

The latest technology for digital cellular telephony is CDMA, which has the advantages outlined in Section 2.3.3. The U.S. standard for CDMA is called Interim Standard-95 or IS-95. In IS-95, each user channel at 9.6 Kbps is spread to a rate of 1.2288 Mchips/s (a spreading factor of 128). On the downlink, user data is encoded using a rate 1/2 convolutional code, interleaved, and spread with one of 64 orthogonal spreading codes called the Walsh spreading codes. Adjacent base stations coordinate their use of the codes to minimize interference. On the uplink, the data stream is encoded with a rate 1/3 convolutional coder and six-way interleaved. The interleaved bits are then spread using a Walsh code. The power of the uplink is tightly controlled (to within 1dB) to avoid the near-far problem. To ensure compatibility, IS-95 mobile stations can also interoperate with EAMPS base stations, and the IS-95 frequency bands are the same as the EAMPS frequency bands. IS-54, IS-95, GSM, and other digital cellular telephony services are also called Personal Communication Services or PCS.

2.4.2 Polling and probing

In the previous subsection, we studied centralized control for circuit-mode communication for cellular telephony (similar schemes are also used for cordless telephones). In this and the next subsection, we will study two classes of schemes with centralized control of *packet-mode* communication. Recall that in packet mode, stations generate bursts of traffic that cannot be predicted in advance. Thus, a station must contend for medium access for each packet. A central controller mediates this contention.

One of the simplest schemes for central control in packet-mode is roll-call *polling*. In this scheme, the master asks each station in turn whether it has data to send. If the station has data, it sends it to the master (or directly to another station). Otherwise, the master continues polling. The main advantage of roll-call polling is that it is simple to implement. It is inefficient if (a) the time taken to query a station is long (due to station or propagation delays), (b) the overhead of polling messages is high, or (c) the system has many terminals. Roll-call polling can lead to high mean message delays. In the worst case, each station has data to send just after it has been passed over during a poll, so that it has to wait for every other station to send data before it has a chance to send anything.

A variant of roll-call polling that is more intelligent in its polling order is called *probing.* In this scheme, stations are

given consecutive addresses. We assume that each station can program its host interface to be able to receive data addressed not only to itself, but also to "multicast" or group addresses. Here follows an example detailing how it works.

Example 2.1

Suppose a network has eight stations numbered 0 - 7 (000 to 111 in binary). In the first time slot, the master sends a message to the multicast address 0*, asking for data. This query is received by all stations that have a 0 in the first bit of their address (that is, stations 000, 001, 010, and 011). If one of them has data to send, it replies and is given permission to send data in the next time slot. If more than one station in this range has data to send, then both reply, and their replies collide. On seeing the collision, the master restricts the query to 00* and polls again. This time, only stations 000 and 001 may answer. If there is another collision, stations 000 and 001 are polled in turn, followed by a poll to the multicast address 01 *. Continuing in this manner, the master can skip over large chunks of the address space that have no active stations, at the expense of repeated polls in sections of the address space that have more than one active station. In the worst case, when all stations are active, this results in doubling the number of poll messages. However, if many stations share the medium, of which only a few are active, probing is quite efficient.

2.4.3 Reservation-based schemes

A different approach to centralized control becomes necessary for packet-mode transmission when a is large, so that collisions are expensive, and the overhead for polling is too high. This is common in satellitebased networks, where the round-trip propagation delay between stations varies between 240 and 270 ms (depending on whether the satellite is near the zenith or the horizon, respectively), and the packet transmission time is roughly a few milliseconds, leading to an a value of around 100. Then, it is more efficient for a master (which can be located at either the satellite or a ground station) to coordinate access to the medium using a reservation-based scheme.

The essential idea in a reservation-based scheme is to set aside some time slots for carrying reservation messages. Since these messages are usually smaller than data packets, reservation time slots are smaller than data time slots and are called *reservation minislots.* When a station has data to send, it requests a data slot by sending a reservation message to the master in a reservation minislot. In some schemes, such as in fixed priority oriented demand assignment or FPODA, each station is assigned its own minislot. In other schemes, such as in packet-demand assignment multiple access or PDAMA, slaves contend for access to a minislot using one of the distributed packet-based contention schemes described in Section 2.5.5 (such as slotted ALOHA). When the master receives the reservation request, it computes a transmission schedule and announces the schedule to the slaves.

In a reservation-based scheme, if each slave station has its own reservation minislot, collision can be completely avoided. Moreover, if reservation requests have a priority field, the master can schedule slaves with urgent data before slaves with delay-insensitive data. Packet collisions can happen only when stations contend for a minislot. Therefore, the performance degradation due to collisions is restricted to the minislots, which use only a small fraction of the total bandwidth. Thus, the bulk of the bandwidth, devoted to data packets, is efficiently used.

2.5 Distributed schemes

In Section 2.4, we studied several multiple-access schemes where one of the stations plays the role of a master. As we saw, the presence of a master simplifies a multiple-access scheme since it provides a single point of coordination. However, we are often interested in a distributed access scheme, because these are more reliable, have lower message delays, and often allow higher network utilization. The price for these benefits is increased system complexity, for example, to synchronize stations to the same time base.

Most distributed multiple access schemes support only packet-mode access. The reason for this, perhaps, is that with circuit-mode transfer, the overhead of negotiating medium access for each packet in a stream is unacceptable. It makes more sense to use a master-slave configuration for circuit-mode access, since the one-time coordination overhead is amortized over many packets. Circuit-mode access is not ruled out with a distributed scheme, but it is rare.

2.5.1 Decentralized polling and probing

Perhaps the simplest distributed access schemes are variants of the centralized polling and probing schemes described in Section 2.4.2. Both schemes can work without a master, if all the stations agree on a single time base. In the centralized version of polling, the master polls each slave in turn. In the distributed version, a station sends data in its time slot, or else is idle (this is just TDMA). For this to work correctly, we assume that a station is statically associated with its own private time slot, and that there are at least as many time slots as possible stations.

We will study the decentralized version of probing, also called tree-based multiple access, continuing with the example introduced in Section 2.4.2.

Example 2.2

Recall that in Example 2.1, we have eight stations, addressed 000-111, trying to access the shared medium. In the tree-based scheme, in the first time slot, every station with a 0 in its high-order address bit "places" a packet on the medium. If there is a collision, in the second time slot the stations with an address of the form 01* become idle, and the two stations with the address 00* try again. If another collision happens, station 000 goes first, followed by station 001. The two stations with address 01* now contend for access, and so on. Thus, if stations have the same time base, they can carry out what looks like centralized probing, but without a master.

This approach works well when the parameter *a* is small, so that stations can detect collisions quickly and can easily establish a common time base. When *a is* large, a station detects a collision only several time slots *after* it has transmitted a packet. This requires the station either to introduce an idle time after each transmission while it waits for a possible collision, making the scheme inefficient, or to roll back its state if a collision is detected, making the scheme complex. Thus, tree-based multiple access is best suited only for networks with small *a*.

2.5.2 CSMA and its variants

The problem with polling and tree-based algorithms is that they waste time when the number of stations is large, but the number of simultaneously *active* stations is small. Consider a system with 1024 stations, of which at most two are simultaneously active. With polling, in the worst case, a station may need to wait 1023 slots before it has a chance to send a packet. Even with a tree-based algorithm, the station may need to wait up to ten slots while the medium lies unused. A clever idea that makes better use of the medium is to introduce a carrier-sensing circuit in each base station. This allows the station to detect whether the medium is currently being used. This is like a participant in a meeting keeping one ear open to find out if another participant is speaking. Schemes that use a carrier-sense circuit are classed together as carrier-sense multiple access or CSMA schemes. In this subsection we will study CSMA and two of its variants, CSMA/CD and CSMA/CA.

An important parameter in all CSMA schemes is the time taken for a message sent from one station to be heard by the station furthest away. This is just the maximum propagation delay in the system. CSMA schemes assume that this value is small compared with a packet transmission time, that is, the parameter *a is* small. Usually, *a is* assumed to be 0.1 or smaller. In slotted versions of CSMA, where packet transmissions are aligned to a slot boundary, the slot time is chosen to be the maximum propagation delay.

The simplest CSMA scheme is for a station to sense the medium, sending a packet immediately if the medium is idle. If the station waits for the medium to become idle, we call it *persistent;* otherwise we call it *nonpersistent.* With nonpersistent CSMA, when a station detects that the medium is busy, it sets a timer for a random interval, then tries again. Thus, a nonpersistent CSMA source probes the medium at random intervals, transmitting a packet immediately after detecting that the medium is free.

With persistent CSMA, what happens if two stations become active when a third station is busy? Both wait for the active station to finish, then simultaneously launch a packet, guaranteeing a collision! There are two ways to handle this problem: *p-persistent* CSMA and *exponential backoff.*

_P_-persistent CSMA

The first technique is for a waiting station not to launch a packet immediately when the channel becomes idle, but first toss a coin, and send a packet only if the coin comes up heads. If the coin comes up tails, the station waits for some time (one slot for slotted CSMA), then repeats the process. The idea is that if two stations are both waiting for the medium, this reduces the chance of a collision from 100% to 25%. A simple generalization of the scheme is to use a biased coin, so that the probability of sending a packet when the medium becomes idle is not 0.5, but p, where $0 < p \leq 1$. We call such a scheme p-persistent CSMA. The original scheme, where $p = 1$, is thus called 1-persistent CSMA.

The choice of p represents a trade-off between performance under heavy load and mean message delay. Note that if n stations are waiting for the end of a packet transmission, then the mean number of stations that will send a packet at the end of the transmission is just np. If $np > 1$, then a collision is likely, so we must choose $p < 1/n$. Since n increases with system load, a smaller p leads to good behavior for higher offered loads. On the other hand, as p decreases, a station is more likely to wait instead of sending a packet, though the medium is idle. Thus, the smaller the value of p, the greater the mean message delay. In a given system design, p must be chosen to balance the message delay with the required performance under heavy offered loads.

Exponential backoff

The second technique to deal with collisions between CSMA stations is to use *exponential backoff.* The key idea is that each station, after transmitting a packet, checks whether the packet transmission was successful. Successful

transmission is indicated either by an *explicit* acknowledgment from the receiver or by the absence of a signal from a *collision detection* circuit (this circuit is similar to the carrier-sensing circuit, and detects a collision by noticing that the signal energy in the medium is greater than the energy placed on the medium by the local station). If the transmission is successful, the station is done. Otherwise, the station retransmits the packet, simultaneously realizing that at least one other station is also contending for the medium. To prevent its retransmission from colliding with the other station's retransmission, each station backs off (that is, idles) for a random time chosen from the interval [0, 2×max-propagation-delay] before retransmitting its packet. If the retransmission also fails, then the station backs off for a random time in the interval [0, 4× max-propagation-delay], and tries again. Each subsequent collision doubles the backoff interval length, until the retransmission finally succeeds (thus, the expected duration of a backoff interval increases exponentially fast)[21]. On a successful transmission, the backoff interval is reset to the initial value. Intuitively, sources rapidly back off in the presence of repeated collisions, thus drastically reducing the load on the medium, and ideally averting future collisions. We call this type of backoff *exponential backoff.* With exponential backoff, even 1-persistent CSMA is stable, thus freeing network designers from the onerous task of choosing an optimal *p*.

We mentioned earlier that a station determines that its transmission is successful either using an explicit acknowledgment or using a collision detection circuit. CSMA with collision detection is common enough to merit its own acronym, CSMA/CD It is the scheme used in Ethernet, to which we now turn our attention.

Ethernet

Ethernet is undoubtedly the most widely used local-area networking technology. Actually, "Ethernet" is a trademark that refers loosely to a variety of products from many manufacturers. Network cognoscenti prefer to use the term IEEE 802.3, which is the international standard describing the physical and datalink-layer protocols used in Ethernet-like LANs. In this book, for the sake of convenience, we use "Ethernet" when we mean IEEE 802.3.

Ethernet uses a variant of 1-persistent CSMA/CD with exponential backoff on a wired LAN. Moreover, if a station detects a collision, it immediately places a "jam" signal (a sequence of 512 bits) on the medium, ensuring that every active station on the network knows that a collision happened and increments its backoff counter. Since Ethernet is used for networks where *a* is small, collisions can be detected within a few bit-transmission times. Thus, the time wasted in each collision is small (about 50 microseconds), which increases the effective throughput of the system. To keep things simple, the Ethernet standard requires a packet to be large enough that a collision is detected before the packet transmission completes. Given the largest allowed Ethernet segment length and speed-of-light propagation times, the minimum packet length in Ethernet turns out to be 64 bytes. If the largest distance between any pair of stations is longer, the minimum packet size is correspondingly higher.

The first version of Ethernet ran at 3 Mbps and used "thick" coaxial cable for the physical medium. Subsequent versions have increased the speed to 10 Mbps and, more recently, to 100 Mbps. The physical medium has also diversified to include "thin" coaxial cable, twisted-pair copper, and optical fiber. In an attempt to keep things straight, the IEEE classifies Ethernet using a code of the form *<Speed> <Baseband or Broadband> <Physical medium>*. The first part, speed, is 3, 10, or 100 Mbps. The second part describes the infrastructure over which the network is run. Ethernet can be used not only within a building between computers, but also within a frequency band allocated to it in the cable-television infrastructure. The former is called *baseband,* and the latter *broadband.* The third part, the physical medium, is either a number, which refers to the longest allowed segment, in hundreds of meters, that the medium supports, or a letter, which represents a particular medium type. For example, 10Base2 is a 10 Mbps Ethernet that uses the baseband, and therefore is confined to a single building or campus. The "2" means that this physical medium can run at most 185 meters before a repeater must be installed. "2" usually refers to cheap, "thin" 50ohm cable that is a poor substitute for the "thick" coaxial cable used in 10Base5 Ethernet. 10BaseT is 10 Mbps Ethernet over unshielded twisted-pair copper. Finally, 10Broad36 is Ethernet over the cable TV plant, with at most 3600 meters between repeaters. This is the technology used for some so called "cable modems" (and has turned out to be a commercial failure). A wealth of practical details on Ethernet can be found by searching google for Ethernet FAQ. One URL is http://www.faqs.org/faqs/LANs/ethernet-faq/.

Three recent developments give LAN administrators even more options in managing their infrastructure. The first of these, *switched* Ethernet, continues to use 10-Mbps links, but each station is connected to a switch (also called an *Ethernet switch)* by a separate wire, as in 10BaseT. However, unlike 10BaseT, each line card in the switch has a buffer to hold an incoming frame. A fast backplane allows packets to be transferred from one line card to another. Since packets arriving to the switch simultaneously do not collide, switched Ethernet has a higher intrinsic capacity than

[21] Most real systems give up after backing off a certain number of times, typically 16.

10BaseT. This comes at the expense of memory in the line card.

The other two developments increase Ethernet speed from 10 to 100 Mbps. The first variant, called IEEE 802.3u or *Fast Ethernet, is* conceptually identical to 10BaseT, except that the line speed is increased to 100 Mbps. Like 10BaseT, it requires a point-to-point connection from a station to a hub. Fast Ethernet is not supported on bus-based Ethernets such as 10Base5 and 10Base2. It is interesting because it can reuse telephone wiring, just like 10BaseT.

Commercially available hubs are rapidly erasing the distinctions among 10BaseT, switched Ethernet, Fast Ethernet, and switched Fast Ethernet. These hubs support line cards in all four formats interchangeably, allowing customers to upgrade their line cards one at a time.

CSMA/CA

Unlike wired LANs, where a transmitter can simultaneously monitor the medium for a collision, in many wireless LANs the transmitter's power overwhelms a co-located receiver. Thus, when a station transmits a packet, it has no idea whether the packet collided with another packet or not until it receives an acknowledgment from the receiver. In this situation, collisions have a greater effect on performance than with CSMA/CD, where colliding packets can be quickly detected and aborted. Thus, it makes sense to try to avoid collisions, if possible. A popular scheme in this situation is CSMA/Collision Avoidance, or CSMA/CA. The IEEE has standardized CSMA/CA as the IEEE 802.11 standard. CSMA/CA is basically *p*-persistence, with the twist that when the medium becomes idle, a station must wait for a time called the interframe spacing or IFS before contending for a slot. A station gets a higher priority if it is allocated a smaller interframe spacing.

When a station wants to transmit data, it first checks if the medium is busy (see B2T56). If it is, it continuously senses the medium, waiting for it to become idle. When the medium becomes idle, the station first waits for an interframe spacing corresponding to its priority level, then sets a contention timer to a time interval randomly selected in the range [0, CW], where CW is a predefined contention window length. When this timer expires, it transmits a packet and waits for the receiver to send an ack. If no ack is received, the packet is assumed lost to collision, and the source tries again, choosing a contention timer at random from an interval twice as long as the one before (binary exponential backoff). If the station senses that another station has begun transmission while it was waiting for the expiration of the contention timer, it does not reset its timer, but merely freezes it, and restarts the countdown when the packet completes transmission. In this way, stations that happen to choose a longer timer value get higher priority in the next round of contention.

2.5.3 Dealing with hidden terminals: BTMA and MACA

CSMA/CA works well in environments where every station can hear the transmission of every other station. Unfortunately, many environments suffer from the *hidden terminal* and *exposed terminal* problems. In the hidden terminal problem, station B can hear transmissions from stations A and C, but A and C cannot hear each other (see Figure B2T67 left). Thus, when A sends data to B, C cannot sense this, and thinks that the medium is idle. C's transmission after an IFS, therefore, causes a collision at B. Continuing with the meeting metaphor, a hidden terminal situation occurs when a participant can hear speakers on either side, but these speakers cannot hear each other speak.

The exposed terminal problem is shown in Figure (B2T67 right). Here, we have two local areas, with station B talking to station D in its local area, and station C with a packet destined to A in its own local area. Station B is "exposed," perhaps because it is on a hilltop, or mounted on the ceiling of a large hall. Thus, unlike D, it can be heard by A when transmitting. Since A can hear B, A defers to B when B is active, though A and B can be simultaneously active. Thus, because of B's exposed location, it defers to faraway transmitters[22] even when it need not. Again, CSMA/CA does not work, because the exposed station senses more than it ought to. We now study two solutions to these problems, busy tone multiple access and multiple access collision avoidance.

Busy tone multiple access

In busy tone multiple access (BTMA), we assume that for any two stations A and B, if station A can hear station B, then station B can hear station A (that is, the wireless links are symmetrical). We divide the available frequency band into a message channel and a smaller "busytone" channel. While a station is receiving a message, it places a tone on the busy-tone channel. Other stations that want to send a message to this receiver (or, any other receiver that is in range of the receiver) thus know to avoid placing a message on the channel. When sending a message, a transmitter ignores its carrier-sense circuit, and sends a message if and only if the busy-tone channel is idle. BTMA provides protection against the hidden terminal and exposed terminal problems. For example, in Figure (B2T67 left), when B receives data

[22] In this example we assume symmetrical links.

from A, it places a busy tone on the busy-tone channel. Since C can hear B, C can hear B's busy tone, and it does not send data to B. In the exposed terminal situation (Figure B2T67 right), if B does not interfere with C, B does not hear C's busy tone, so it goes ahead with its transmission to D (even if it senses carrier from A sending to C).

Multiple access collision avoidance

The problem with BTMA is that it requires us to split the frequency band into two parts, making receivers more complex (because they need two separate tuners). The two bands need to be well separated to prevent crosstalk. Unfortunately, the propagation characteristics of a radio link depend on the frequency. Thus, a station may hear another station's busy tone even if it cannot hear that station's data, or vice versa, causing a problem for the BTMA protocol. We can avoid these problems by using a single frequency band for all messages, and replacing the busy tone with an explicit message that informs all stations in the area that the receiver is busy. This is done in the multiple access collision avoidance or MACA scheme.

In MACA, before a station sends data, it sends a *request to send* (RTS) message to its intended receiver. If the RTS succeeds, the receiver returns a *clear to send* (CTS) reply. The sender then sends a packet to the receiver. If a station overhears an RTS, it waits long enough for a CTS message to be sent by a receiver before it tries to send any data. If a station hears a CTS (which carries the length of the data packet in its body), it waits long enough for the data packet to be transmitted before it tries sending a packet.

The RTS and CTS messages allow intended receivers and transmitters to overcome the hidden terminal and exposed terminal problems. Consider, first, the hidden terminal problem (Figure B2T67 left). Suppose C wants to transmit data to A, but C cannot hear B is sending to A. When A sends a CTS to B, C hears the CTS, and realizes that A is busy. It, therefore, defers its RTS until B's transmission completes. This solves the hidden terminal problem. In the exposed terminal scenario (Figure B2T67 right), B hears A's RTS, but not C's CTS. Thus, it assumes that D cannot hear it either, and sends D an RTS after waiting for the CTS to reach A. If D cannot hear A, it replies immediately with a CTS. The B - D transmission is not unnecessarily affected by the A - C transmission. Thus, the RTS/CTS messages solve the hidden and exposed terminal problems.

2.5.4 Token passing and its variants

Recall that in the distributed polling network (Section 2.5.1), all stations share the same time base, and each station uses its time slot to place a message on the medium. Ensuring that all stations acquire and maintain time synchronization is nontrivial and requires some bandwidth for training and synchronization signals. If a station could inform the "next" station in the polling sequence when it was done with its transmission, stations no longer need precise time synchronization. This is the basic idea used in *token-ring* networks.

In a token-ring network, stations are placed in a fixed order along a ring. This does not necessarily correspond to actual physical connectivity: the key idea is that each station has a well-defined predecessor and successor. A special packet called a *token* gives a station the right to place data on the shared medium. If a station has data to send, it waits till it receives the token from its predecessor, then holds on to the token when it transmits one or more packets. After packet transmission, it passes the token to its successor. The token mechanism allows the medium to be shared fairly among all the stations (no station will ever starve, as might happen with Ethernet). The obvious analogy in a meeting context is to require a participant to obtain a "right-to-speak" marker before speaking, which is passed on to a successor when she or he is done.

We mentioned earlier that a token ring does not necessarily require stations to be connected in a ring. To stress this point, we show four topologies in Figure B2T71 that all form logical rings. The first topology connects stations in a single physical ring (Figure B2T71a). Each station needs only one transmitter and one receiver. During normal operation, a station copies packets from its receive buffer to its transmit buffer, additionally copying the packet to a local buffer if the packet is addressed to it. Thus, packets eventually return to the sender, who removes them from the ring. The returning packets are an implicit acknowledgment that the receiver has seen them (or, the receiver may explicitly mark a packet when it copies it into its local buffers). A variant of this scheme requires a busy/idle flag in the token. When a station that has data to send sees an idle token, it changes the flag to busy and then starts sending data. When it is done, it waits for the token to return, then resets the flag to idle.

The major problem with a single-ring configuration is that if one link, transmitter, or receiver fails, the entire ring fails. We can mitigate this problem by using a second topology, which is the dual counterrotating ring shown in Figure B2T71b. In normal operation, only one ring is used, and the other serves as a backup. A single failure causes the ring to go into "wrap" mode, where the two rings are collapsed to form a single ring, thus allowing traffic to be carried without pause while the fault is corrected. However, each node requires two transmitters and receivers, and stations must constantly monitor their neighbors, checking to see if the network should change from regular mode to "wrap" mode and vice versa. A good example of this topology is FDDI, which is described at the end of this subsection.

The third topology, also called a *token bus*, dispenses with the ring altogether and stations are connected to a single bus, as in Ethernet (Figure B2T71c). A station gets to speak when it gets a token from its predecessor that is explicitly addressed to it. Having spoken, it passes the token to its logical successor. Thus, a token bus forms a single logical ring.

The fourth topology, shown in Figure B2T71d, is called a *star-ring* or *hub* topology. A *passive* hub acts only as a wiring concentrator and does not participate in the token-ring protocols. It simplifies wiring because, to add a new station, we need only run a pair of wires from the station to the hub, instead of between the new station, its predecessor, and its successor. Thus, this is identical to the single ring shown in Figure B2T71a. With an *active* hub, also called a *bridge*, the hub is both a predecessor and successor to every station, thus fully participating in the token-ring protocol. An active hub can monitor the ring for link and station failures, eliminating failed components from the ring. This is very important in practice, when machines, interfaces, and links fail with tedious regularity. For these practical reasons, most token rings in the field are configured as either active or passive star-rings.

Advantages and problems with token rings

The main virtue of a token ring is that its medium access protocol is explicit, and therefore simple. If a station has a token, it can send data, otherwise, it cannot. Stations do not need carrier sensing, time synchronization, or complex protocols to resolve contention. Moreover, the scheme guarantees zero collisions and can give some stations priority over others. Unfortunately, these virtues are tempered by a major vice: the token represents a single point of failure in the system. If the token gets lost or corrupted (for example, if the station holding the token suddenly crashes), the network becomes unusable. Thus, the token must be carefully protected with checksums and error-correction, as described in bloc 4. Moreover, stations must actively monitor the network to detect token loss and duplication. Usually, one of the stations is elected as a *monitor*. If the monitor finds that the ring has been idle for some time and no token has passed by, it assumes that the token has been lost or corrupted and generates a new token. A station can decide that a token has been duplicated if it receives a packet from another station when it holds the token. In this case, it purges the ring of all data and tokens, and the monitor eventually generates a new token.

A second problem with a token ring is that the token scheme requires a station to cooperate with the others, at least to the extent of forwarding a token if it does not need it. If a station crashes, refusing to accept or forward the token, it can hold up every other station. Thus, the network must monitor and exclude stations that do not cooperate in token passing. These two requirements (to monitor tokens and stations) reintroduce complexity back into the system. Indeed, a major reason for the initial unpopularity of token rings was the cost and added complexity of the *station management* functions, which are responsible for monitoring the health of the token and the stations .

Fiber distributed data interface (FDDI)

The most popular token ring is FDDI, which uses a dual counterrotating token-ring LAN. Each link in FDDI runs at 100 Mbps over an optical fiber medium. Cheaper variants over copper wires are called *copper-DDI* or CDDI.

An FDDI network uses the token-passing mechanism discussed earlier. In addition, it supports real-time applications by ensuring that a station *I* can send data for at least *Synchronous_Allocation(I)* seconds once every target token rotation time (TTRT). To guarantee this, each station maintains a timer called the Token Rotation Timer or TRT. This timer roughly measures the time since the station last forwarded the token. Thus, if a station receives the token when TRT is smaller than TTRT, it has TTRT - TRT seconds to send data without violating the target rotation time.

What actually happens is that when a station receives a token, it sends synchronous packets up to its allocation, regardless of the value of TRT (we must somehow ensure that the sum of the synchronous allocations is smaller than the target token rotation time). It then resets TRT to zero, saving the previous value of TRT in the Token Holding Time (THT). If THT < TTRT, the station can send TTRT - THT worth of non-real-time packets without violating the TTRT. If the station does not have enough time to send a non-real-time packet, it immediately forwards the token, waiting for the next opportunity to transmit. In this way, a real-time application that must receive the token within a bounded time period can be guaranteed access (see TD 7 exercice 9 for an example).

2.5.5 ALOHA and its variants

The ALOHA multiple access protocol is one of the simplest multiple access protocols and was the earliest to be developed and analyzed. In ALOHA, when a *station* wants to send data, it just sends it, without checking to see if any other station is active. After sending the packet, the station waits for an implicit or explicit acknowledgment. If the station receives no acknowledgment, it assumes that the packet got lost, and tries again after a random waiting time. ALOHA is useful when the parameter a is large, so that carrier-sensing and probing techniques are impractical.

Unlike the schemes we studied earlier in this bloc, stations implementing ALOHA do not need carrier sensing, time-base synchronization, token passing, or any other contention-resolution mechanism. Moreover, ALOHA's performance is independent of the value of *a*. Thus, ALOHA has the clear advantage of simplicity. On the other hand,

its performance, compared with the more sophisticated schemes, is quite poor. Under some simplifying mathematical assumptions, we can show that the peak achievable goodput of the scheme is only 18%. This figure has often been used to "prove" that ALOHA is an unusable multiple-access scheme. In fact, if the workload does not obey the simplistic assumptions, much higher goodput is achievable (consider, for example, the case of a single user accessing the medium, who can achieve a goodput of 100%). Moreover, if channel capacity is sufficiently high, so that the normalized offered load is low, the sheer simplicity of ALOHA makes it very attractive. Thus, ALOHA can still be found as a component in many multiple-access schemes. For example, in the cellular telephone network, when a cellular telephone is turned on, it must contact the base station and request a frequency (or time slot) on which to carry a voice call eventually. It uses ALOHA to send this initial request. Since the offered load from frequency allocation requests is small, ALOHA is sufficient for this purpose. Several variants of ALOHA, such as slotted ALOHA and reservation ALOHA, are widely used, and we study them next.

Slotted ALOHA

In ALOHA a newly emitted packet can collide with a packet already in progress. If all packets are the same length and take *L* time units to transmit, then it is easy to see that a packet collides with any other packet transmitted in a time window of length *2L* (Figure B2T79a). We call this the *window of vulnerability.* If we somehow reduce this window, then the number of collisions decreases, and throughput increases. One way to achieve this is for stations to share the same time base by synchronizing with a master that periodically broadcasts a synchronization pulse. We divide time into equal *slots* of length *L.* When a station wants to send a packet, it waits till the beginning of the next slot. This reduces the window of vulnerability by a factor of two, doubling the peak achievable throughput, under the same set of mathematical assumptions as with ALOHA, to 36% (Figure B2T79b). This version of ALOHA is called slotted ALOHA or S-ALOHA. This scheme has a clear advantage in throughput, but introduces complexity in the stations and bandwidth overhead because of the need for time synchronization. It is best suited in an environment where time synchronization is already needed for another purpose. For example, in the TDMA GSM cellular telephone system, stations use slotted ALOHA to request a TDMA time slot. Since stations need to synchronize for TDMA anyway, slotted ALOHA poses little additional overhead.

Loss detection

We remarked earlier that a station in a multiple-access scheme can detect packet loss either implicitly or explicitly. Satellite ALOHA networks use an implicit loss-detection scheme. Usually, a satellite simply broadcasts whatever signal it receives from any station. Thus, a station knows that it should receive whatever it sent to the satellite one round-trip time later. If what it receives at this time is garbled, its packet suffered from a collision and must be retransmitted (Figure B2T80). Cellular ALOHA networks use an explicit acknowledgment scheme. When the base station successfully receives a channelrequest message on an uplink, it places an acknowledgment on the downlink (this acknowledgment can be a busy tone on an acknowledgment channel, or setting a busy/ idle bit on a downlink frame to "busy"). If a station does not see an acknowledgment in a reasonable amount of time, it times out, waits for a random time, and retransmits the packet.

Problems with stability

Besides the low achievable goodput, a second major problem with ALOHA and slotted ALOHA is that of stability. Since ALOHA is essentially uncontrolled, a sudden burst of traffic leads to numerous collisions. Each lost packet is retransmitted, and may, in turn, collide with newly generated packets that also require access to the medium. If the offered load is sufficiently high, the network can be put in an unstable state where every packet is a retransmission, and the goodput drops to zero. Luckily, this grisly fate can be avoided by using a sufficiently aggressive backoff policy, such as binary exponential backoff, described in Section 2.5.2. Designers of all ALOHA-like multiple access systems must include reasonable backoff mechanisms to assure system stability.

Reservation ALOHA

Reservation ALOHA or *R*-ALOHA combines the notion of slot reservation in schemes such as FPODA, described in Section 2.4.3, with the slotted ALOHA multiple-access scheme. Unlike a centralized reservation-oriented scheme (such as FPODA), R-ALOHA and its variants do not have a master that can resolve channel access priorities. Instead, stations independently examine reservation requests and come to consistent conclusions about which station owns which transmission slot. Thus, in R-ALOHA, all stations have the same priority (though streams within a station can get different priorities).

In the simplest version of R-ALOHA, we divide time into *frames,* where each frame consists of a fixed number of time slots. A station that wins access to a slot using S-ALOHA is automatically assured ownership of the same slot in the next frame. A station that has data to send keeps track of which slots are idle in the current frame: these slots are fair game for contention in the next frame. Thus, a station that has a steady stream of packets to send (as in circuit mode) needs to contend for access only once, while stations that want to access the medium in packet-mode are not shut out.

Since a station does not suffer from collisions once it has acquired a slot, we can show that the performance of R-ALOHA is usually better than that of S-ALOHA.

In R-ALOHA, a station determines which data slots in a frame are available to it by using information from a previous frame. Thus, the previous frame must be long enough so that a station receives at least the start of the previous frame before the frame's transmission ends. In other words, the frame length must be at least as long as *a*. For example, in a network where $a = 200$, a station may send 200 packets before another station sees the first one. If the first of these contained the reservation minislots, for example, the station ought not start the next frame till at least 200 slot times have passed. Thus, the smallest possible frame length is 200 slots. This restriction on the size of a frame also bounds the delay performance of a reservation scheme. Since a packet that arrives to a previously idle station must wait for a reservation before it can be sent, the packet will see a delay of at least a slots.

Besides this, R-ALOHA suffers from two other problems. First, a station that gains access to the medium, and always has something to send, cannot be preempted. Thus, a station that initiates a low-priority bulk transfer cannot be evicted in favor of a high priority transfer. Second, if all stations want to use packet mode, the automatic reservation of a slot in the next frame means that packets that need only one slot still use two. This doubles the packet-mode bandwidth overhead. We can avoid this overhead if a station appends an "end-of-transmission" flag to its last packet, which signals that its slot in the next frame is up for grabs. Variants of R-ALOHA that allow a certain amount of preemption were also proposed.

2.6 Summary

The section 2 of this bloc discusses multiple-access schemes that arise in five main contexts: wired LANs, wireless LANs, packet radio, cellular telephony, and satellite communications. Despite their differences, they all face a similar problem, which is to coordinate access to a shared medium. In solving this problem, we can choose to build a centralized or a distributed system and use circuit-mode or packet-mode transfer. Designers must try to maximize the throughput and minimize the mean delay; they must also try to achieve stability, at least at low loads. It is useful to divide the multiple-access problem into providing base technologies and multiple-access schemes.

The three main base technologies are FDMA, TDMA, and CDMA. Each has its own problems, and all three are used in real-life systems. TDMA represents a compromise between FDMA and CDMA in terms of complexity and flexibility, and is popular for both circuit-mode and packet-mode transmission.

Centralized schemes use a master station to coordinate the actions of the slave stations. Circuit-mode centralized schemes, which are used in cellular telephony, can be categorized according to the base technology they use. In the United States, the most popular cellular technology is EAMPS, which uses FDMA, and in Europe, the most common scheme is GSM, which uses TDMA. Other, centralized packet-mode schemes are polling, for wired LANs, and reservation schemes, which are used primarily for satellites.

Distributed schemes are usually packet mode and are common in wired and wireless LANs. The simplest of these schemes is distributed polling, which is similar to the centralized version. For networks where the propagation delay is small compared with the packet transmission time, carrier sensing is effective and is used in the CSMA scheme. CSMA's variants, particularly CSMA /CD, which is used for Ethernet, dominate multiple access for wired local-area networks. Token-ring schemes, such as the double counterrotating ring design used for FDDI, are also popular choices for these networks. Wireless LANs use either CSMA/CA, or, when hidden and exposed terminals are common, BTMA or MACA. Finally, the ALOHA scheme and its variants, S-ALOHA and R-ALOHA, are common in satellite networks.

A network designer should choose a scheme that best satisfies the requirements posed by link characteristics, station complexity, and achievable performance.

Bloc 3

Adressage et routage point à point dans l'Internet

1.0 Naming and addressing

1.1 Introduction

Consider a user who walks up to a computer connected to the Internet and types *ftp research.att.com*, thus asking for a file transfer session to be initiated between the local computer and a computer with the name *research.att.com.* The

local computer must first translate from a human-understandable *name* (research.att.corn) to a numerical identifier called the destination's *address* (here, 135.104.117.5). Understandably, both names and addresses must be globally unique. *Naming is* the process of assigning unique names to endpoints, and *addressing is* the process of assigning unique addresses. We call the process of translating from a name to an address *resolution.*

After figuring out the address, the network must find a *route,* that is, a way to get from the source to the destination. A route may be found before data transfer, as in a connection-oriented network, or may be found on-the-fly, when a packet arrives at a router, as in a datagram network. In either case, given a destination address, switches or routers need a way to find the next hop on which to forward a call-setup packet or a datagram. This is the function of *routing.* In section 1, we study naming and addressing, and in section 2 we study routing.

1.2 Naming and addressing

If names and addresses both serve to identify a destination uniquely, why do we need them both? There are two reasons. First, names are usually human-understandable, therefore variable-length, and potentially rather long. For example, you could name your computer *very- long-name-and-I-dare-you-to-change-it,* and that would be perfectly acceptable. However, every packet in a datagram network must carry an indication of its destination. If we use names in packet headers, the source and destination fields in the packet header must be variable-length, and possibly quite long. Not only does this waste bandwidth, but also it would be more complicated for a router receiving the packet to look up the name in a routing table. Using fixed-length identifiers (that is, addresses) in packet headers is more efficient. (This restriction is less applicable in connection-oriented networks, where the destination identification is carried only once during connection establishment.)

The second reason to separate names and addresses is that this separation provides a level of indirection, giving network administrators some leeway in independently reorganizing names and addresses. For example, if a computer moves from one location to another within the same building or campus, an administrator can change its address without changing its name. If we inform the name-to-address translation mechanism about the change, users of the computer's name are not affected.

1.3 Hierarchical naming

Suppose you were given the job of uniquely naming every computer in the Internet. One way to do this would be to give each machine a name from a dictionary, choosing names in some order, and crossing off names as you assign them. This is easy enough when only a single naming authority (that is, you) is allowed to choose names. However, if many authorities were to choose names in parallel, conflicts are likely. For example, you may choose to name a computer *rosebud* exactly at the same time as another authority. You could come up with a protocol where you circulate a proposed name to every other naming authority before finalizing it. This would ensure global uniqueness, but can be somewhat time-consuming, especially if there are many widely separated naming authorities. A better solution is possible if we invoke a higher authority that assigns each naming authority a unique prefix. For example, you may be allowed to name machines with names starting with *a,* and another authority may be allowed to name machines with names starting with *b.* If both of you obey this rule, you can choose any name you wish, and conflicts will never arise.

Partitioning the set of all possible names (the *name space)* into mutually exclusive portions (or *domains)* based on a unique prefix simplifies distributed naming. The unique prefix introduces a *hierarchy* in the name space, as shown in Figure B3T6. Here, we use the rule that names starting with *a* are written in the form *a.name,* where the "." is a special character showing the domain boundary. We can generalize the name assignment rule to multiple levels of hierarchy. For example, you may choose to give your subordinates portions of the name space prefixed by *a.a, a.b, a.c,* etc. If they start their names with these prefixes, the names they generate are guaranteed to be universally unique.

We call the first level in the hierarchy the *top-level domain.* The rule for naming is, therefore, that a global authority assures uniqueness of names in the top level. Authorities in a given domain can give away part of the name space to lower-level naming authorities, if the prefixes handed to these authorities are unique within that domain. This rule applies recursively at all levels, and therefore we can make the naming hierarchy arbitrarily deep.

A hierarchically organized name space scales without bound, yet allows names to be chosen without consulting every other naming authority on the planet. Because of these wonderful properties, it is used in every large network, including the Internet, the telephone network, and ATM networks.

- Names on the Internet follow the conventions established by the *domain name system,* or DNS. A global authority assigns top-level domains with unique names, such as *edu, com, cz, in,* or *net.* Naming authorities in charge of each domain then parcel out names within that domain to lowerlevel authorities.

- The telephone network identifies endpoints with hierarchical telephone numbers. Since these are not

understandable by normal human beings, the telephone network, arguably, has no naming, only addressing.

Example 1.1

The name administrator at UC Berkeley might be given the authority to give names in the domain *berkeley.edu.* (Note that Domain Name System uses a unique suffix rule, instead of a unique prefix rule. This is an equally valid way to create a hierarchy.) The campus naming administrator, might, in turn, give the naming authority in the Mechanical Engineering Department fiefdom over the domain *mech.berkeley.edu.* A lowly graduate student lording it over a roomful of PCs might receive the domain *pclab. mech.berkeley.edu*, and may name the machines, prosaically, *pc1.pclab.mech.berkeley.edu, pc2.pclab.mech.berkeley.edu*, etc. By construction, these names are unique.

1.4 Addressing

Addresses are numerical identifiers that, like names, are globally unique. Therefore, just like names, we usually organize them as a hierarchy. Another important reason for hierarchical addresses is that they allow aggregation in routing tables, as shown in Figure B3T8. We now study the relationship between hierarchical addressing and aggregation.

Example 1.2

In Figure B3T8 left, we see a 10-node network where the addresses have been chosen from a nonhierarchical (or flat) name space. Suppose computer 3 wanted to send a message to computer 5. Since 3 and 5 are not directly connected, 3 must choose to send the message to either 2 or 4, which, in turn will forward it to 5. It turns out that if 3 sends it to 2, the packet will take at least four hops, but if it sends it to 4, it can take as few as two hops. Thus, 3 should send it to 4. Usually, 3 needs to maintain a routing table that shows the next hop for every destination in the network. Here, because 3 has nine possible destinations, the routing table will have nine entries.

Although having one entry per destination is reasonable for networks with hundreds, or even thousands of destinations, it is impractical for networks with tens of millions of destinations, such as the Internet. To work around this problem we must aggregate addresses into clusters by using a hierarchical address space.

Example 1.3

In Figure B3T8 right, we have renamed each node with a two-part address. We now call node 5 node 2.3, which we interpret as computer 3 in subnetwork 2. Certain nodes in each subnetwork are shaded, to mark them as special nodes. We call these *border routers*, and they carry all the traffic into and out of the subnetwork. Now, when node 3 in Figure B3T8 left (now called 1.2) wants to contact 2.3, it simply sends the packet to its border router, 1.1. The routing table in 1.1 shows that the shortest path to subnetwork 2 is via computer 2.1, which, in turn, routes the packet through either 2.4 or 2.2 to 2.3. Router 1.1 only stores a route to *subnetwork* 2, instead of to every router in subnetwork 2. Thus the router *aggregates* routes to subnetwork 2, making its routing table smaller.

Note that the route from 1.2 to 2.3 is four hops long, which is longer than the two-hop shortest path. On the other hand, every nonborder router has only one entry in its routing table, the address of its border router. Each border router has an entry for every computer in its subnetwork, but only one entry for every other subnetwork. Thus, we have traded off some inefficiency in routing for a dramatic reduction in the number of routing entries. This reduction is possible because the addresses are hierarchical, and the network is partitioned along the same lines as the addresses. If computers in subnetwork 3 could have arbitrary addresses, border routers in other subnetworks would need to have one routing entry for each computer in 3. We can reduce the number of entries in a border router only because every node in subnetwork 3 has a prefix of 3. We refer to addresses in subnetwork 3, and the subnetwork itself, as 3*.

1.5 Addressing on the Internet

The Internet addresses host interfaces instead of endpoints. Thus, a computer with multiple interfaces on the Internet has an address for each interface. Internet addresses are used by the Internet Protocol (IP), so they are usually called IP

addresses. There are two versions of IP addresses: version 4, also called IPv4 addresses, and version 6, called IPv6. IPv4 addresses are 4 bytes long and are divided in a two-part hierarchy. The first part is called the *network* number, and the second part is a *host* number (although it addresses an interface). A *subnet mask* describes the partition between the two (Figure 1.3). The logical AND of the IP address and the associated subnet mask is the network number, and the remaining portion is the host number. The subnet mask allows arbitrary partitioning of the address into the network and host number. The Internet Numbers Authority guarantees that subnet numbers are globally unique, and within each subnet, individual network administrators guarantee that IP addresses are unique.

Since IP addresses are partitioned, they can be aggregated, and routing tables in the core of the network need only store a specific route to the router responsible for a particular network. For example, routers in the core of the Internet store a route only to the network number 135.104.*. The router in charge of this network routes the packet to the final destination within 135.104, such as 135.104.53.100.

1.5.1 Address classes

The first cut at IP addresses had a fixed network-host partition with only 8 bits of network number. 2 The designers did not envisage more than 256 networks ever joining the Internet "experiment"! They soon generalized this to allow the address to be partitioned in one of three ways:

- Class A addresses have 8 bits of network number and 24 bits of host number
- Class B addresses have 16 bits of network and host number
- Class C addresses have 24 bits of network number and 8 bits of host number

The classes are distinguished by the leading bits of the address. If the address starts with a 0, it is a Class A address. It has 7 bits for a network number, and 24 bits for a host number. Thus, there can be 128 Class A networks, each with 2^{24} hosts (actually, there can be only 126 networks, since network numbers 0 and 127 have special significance 3) . If the address starts with a 10, it is a Class B address; if it starts with a 110, it is a Class C address. Two special classes of addresses are those that start with 1110 (Class D), which are used for multicast, and those that start with 1111 (Class E), which are reserved for future use.

This solution proved adequate until 1984, when the growth of the Internet forced the first of three changes: subnetting, described in Section 1.5.2; CIDR, described in Section 1.5.3; and *dynamic host configuration*, described in Section 1.5.4.

1.5.2 Subnetting

Owners of Class B addresses had always wanted to further partition their set of host numbers into many smaller *subnets* Then, routing tables within the network need only store routes to subnets, instead of to individual hosts. The class structure is too coarse to deal with this, because the Class B address administrator cannot use the class structure to further partition its address space. The solution is to introduce the subnet mask that we studied at the beginning of this section. Note that the subnet mask chosen within a network is not visible outside the network. Core routers route packets to the border router responsible for the network, where the incoming packet's IP address is interpreted according to the subnet mask valid in that particular network.

Example 1.4

Suppose you are the administrator for the Class B address 135.104.*. If you do not partition the address, every router in your network has to know the route to every host, because you have no way to describe aggregates of hosts within your network. For instance, hosts 135.104.5.0, 135.105.5.6, and 135.105.5.24 may lie in the same physical LAN segment and may all be reachable from router 135.105.4.1. There is no way to express this using the class notation, because these computers all have the same Class B address. You need some extra information to describe this aggregation, which is the subnet mask. Suppose you decide that no subnet is likely to contain more than 256 hosts. Then, you could partition the 65,536 addresses in your domain into 256 subnets, each with 256 addresses. This is expressed by the subnet mask 255.255.255.0 (1111 1111 1111 1111 1111 1111 0000 0000), as shown in Figure B3T12. Addresses within your local Class B network would then be treated as if the network number were the first 24 bits (instead of the first 16 bits), and the host number were the last 8 bits (instead of the last 16 bits). Because the hosts 135.104.5.0, 135.104.5.1, and 135.104.5.2 all lie in the same subnet, that is, 135.104.5.*, routing tables within your Class B network need only have an entry for 135.104.5.* (expressed as 135.104.5.* plus the subnet mask 255.255.255.0) pointing to 135.105.4.1 (the next hop), instead of entries to each of the individual computers. This saves routing table space and route table computation time.

1.5.3 CIDR

As the Internet grew, most networks were assigned Class B addresses, because their networks were too large for a Class C address, which can hold only 256 hosts, but not large enough for a Class A address, which can hold more than 4 million hosts. In 1991, it became clear that the 16,382 Class B addresses would soon run out, which resulted in a crisis in the Internet community. Some network engineers noted that addresses from the enormous Class C space were rarely allocated, since most networks have more than 256 hosts. Their solution was to allocate new networks contiguous subsets of Class C addresses instead of a single Class B address (the allocated set of Class C addresses usually spans a smaller portion of the address space than a single Class B address). To aggregate routes to sets of Class C addresses, routers in the core of the network must now carry a *prefix indication*, just as routers within a network carry subnet masks. The prefix indication is the number of bits of the network address that should be considered part of the network number. Routing protocols that substitute multiple Class C addresses for a Class B address are said to obey *classless interdomain routing* or *CIDR* (pronounced "cider").

Example 1.5

With CIDR, a network could be allocated eight Class C networks, spanning the 2048 addresses from 201.10.0.0 to 201.10.7.255, instead of a single Class B network, with 65,536 addresses. Since the network administrator is allocated eight Class C networks, which use three bits of the Class C space, the remaining 21 bits must be the network number. The address and prefix describing the network are, therefore, 201.10.0.0 and 21, usually written 201.10.0.0/21 (Figure B3T14).

1.5.4 Dynamic host configuration

A third technique to extend the life of the v4 address space is to dynamically allocate hosts with IP addresses. In many situations, a computer, such as a laptop, may access the Internet only once in a while. These hosts need an IP address only when they are active.

Thus, a clever network operator can share the same IP address among different computers, as long as they are not simultaneously active. The protocol to do so is called the *Dynamic Host Configuration Protocol*, or *DHCP*.

In DHCP, a newly booted computer broadcasts a *DHCP discover* packet on its local LAN. This packet contains host-specific information, such as its hardware address. DHCP servers that receive this packet reply with a *DHCP offer* packet that contains an offered IP address and other configuration parameters. A host that receives one or more replies selects a server and IP address, and confirms its selection by broadcasting a *DHCP request* packet that contains the name of the selected server. The server confirms receipt of this message and confirms its offer with a *DHCP ack*. Other servers, on hearing a DHCP request, automatically withdraw their offers. When a host is done, it sends a *DHCP release* message to its selected server, which releases its IP address for use by other hosts.

DHCP has the notion of a *lease*, which is the time for which a host's IP address is valid. A host can guess the time for which it wants an address and put this request in its request packet, can ask for an infinitely long lease, or can periodically renew its lease. When a lease expires, the DHCP server is free to reassign the IP address to other hosts. A wise server should reuse least-recently-used addresses first, to deal with forgetful hosts that may retain their IP addresses past their lease.

A similar technique is used to dynamically allocate IP addresses to computers that access the Internet on a dial-up line. The widely-used *Point-to-Point Procotol (PPP)* allocates a temporary IP address to a host when it dials in. Since only a fraction of dial-up hosts are simultaneously active, an Internet Service Provider can share a pool of IP addresses among a larger set of dial-up hosts.

1.5.5 IPv6

Although CIDR bought the Internet Numbering Authority some breathing space, the 32-bit IP address space will eventually run out. This problem is being rectified in IP version 6, which uses 128-bit addresses. It has been calculated that, even in the most pessimistic scenario, this will result in more than 1500 IP addresses available for each square meter of surface area on the planet. Like v4, v6 distinguishes between multicast and unicast addresses based on a well-known prefix. Version 6 has address prefixes, as in CIDR, to allow aggregation without reference to classes. Subnetting with subnet masks is also allowed.

IP version 6 distinguishes among three types of addresses: unicast, anycast, and multicast. Unicast addresses are defined in RFC2374. Multiple levels of aggregation are defined (top-level aggregation, next-level aggregation and site-level aggregation).

Anycast addresses correspond to more than one interface. The idea is that a packet sent to an anycast address is sent to

one of the interfaces sharing the address.

Multicast addresses correspond to multicast groups. They start with the bit sequence FF. A packet sent to a multicast address is routed to every member of the corresponding multicast group. Multicast addresses contain a flag to show whether the group is well known and therefore permanent, or whether it is transient.

To ease the transition from IPv4 to IPv6 addresses, a special form of the v6 address allows v4 addresses to be encapsulated. In these v6 addresses, the first 80 bits are zero and the next 16 bits are all ones. A complicated set of rules allows interoperation between v6 and v4 hosts through intermediate routers that may or may not understand v6 addresses.

1.6 Name resolution

Users typically supply applications with a destination's name. Applications use *name servers* to resolve a name to an address. A name server is like a telephone directory that stores name-to-address translations. An application resolves a name by sending a query to the name server, which responds with the translation.

The simplest design for a name server is to have a single name server for the entire network. This has the advantage that the resolution is always guaranteed to be consistent, because only one copy of the translation exists. On the other hand, the central name server is not only a single point of failure, but also a choke point, since every endpoint directs its name-resolution queries to it. Thus, we usually compose name servers from a set of distributed agents that coordinate their actions to provide the illusion of a single translation table. Perhaps the most successful distributed name service is the Internet's *Domain Name System (DNS),* which we will use as an example of good name server design.

Recall that DNS partitions names, such as *pc1.pclab.mech.berkeley.edu,* into several hierarchically organized domains, such as *mech, berkeley,* and *edu.* The hierarchy can span arbitrarily many levels. The DNS consists of many name servers, each responsible for a subtree of the name space corresponding to a domain boundary (Figure B3T18). Each server may delegate part of the name space to other servers. For example, the name server responsible for the *berkeley.edu* domain gives responsibility for names ending with **.mech.berkeley.edu* to the *mech.berkeley.edu* name server. This delegation of authority allows domains to match administrative boundaries, considerably simplifying namespace management. DNS administrators arrange things so that every DNS name is guaranteed to be correctly translated by at least one *Authoritative Server* for that name.

When an endpoint wants to translate a name, it sends a query to the server serving the root of the name space. The root parses the name right-to-left, determines the server responsible for the name (the name after the last "."), and forwards the query to that server. For example, if the query is for the name *pc1.pclab.mech.berkeley.edu,* the root server forwards the query to the server responsible for the *edu* domain. This server, in turn, hands off the query to the *berkeley* name server, and so on.

Although the scheme described so far allows each authoritative server to independently update its translation table, it still places a heavy load on the root name server. Moreover, if the root server fails, the entire name resolution process will come to a halt. DNS uses two techniques to combat these problems: *replication* and *caching.*

- *Replication:* DNS allows more than one server to handle requests for a domain, since these servers coordinate among themselves to maintain consistency. In particular, the root name server itself is highly replicated, so that a single failure will not cause any problems. A name resolution query can be made to any of the replicated servers, and an end-system typically chooses the nearest or least loaded one.

- *Caching:* When an endpoint (or an agent acting on its behalf) resolves a name, it stores the result in a cache. Thus, if the query is repeated, it is answered without recourse to the root name server. Endpoints can cache not only the results of a query, but also the addresses of the Authoritative Servers for commonly queried domains. Thus, future queries need not go through the root server. This has been found to reduce name-resolution traffic dramatically. Cached entries are flushed after a time specified in the name-translation table at authorized servers, so that changes in the table are eventually reflected in the entire Internet.

1.7 Datalink-layer addressing

Although the bulk of our discussion of routing deals with network-layer addresses, we will delve briefly into datalink-layer addressing for two common datalink layers-Ethernet and FDDI- because they are commonly used for routing and bridging within a local-area network (LAN).

Both FDDI and Ethernet follow an addressing specification laid down by the 802 committee of the Institute of Electrical and Electronics Engineers (IEEE). This committee is responsible for most LAN standards, and every IEEE 802 LAN obeys the 802 addressing scheme (FDDI is not an IEEE 802 standard, but it uses 802 style addressing

anyway).

An 802 address is 6 bytes long (Figure B3T20 top). The IEEE globally assigns the first 3 bytes, and host-adaptor manufacturers uniquely assign the next 3 bytes, imprinting the resulting address on a ROM on the host-adaptor card. The last two bits of the first byte of the address are used for the *global/local* and *group/individual* flags. If the global flag bit is set, then the address is guaranteed to be globally unique; otherwise, it has only local significance. Thus, an experimental prototype card can be assigned a temporary (but still valid) 802 address, if the local bit is set. The *group* bit marks the address as a datalinklayer broadcast or multicast address. Packets with this bit set can be received by more than one destination (but they must be previously configured to accept packets from this address). The group bit allows efficient datalink-level multicast and broadcast, since an 802 host adaptor needs to match the address on an incoming packet with a set of active multicast addresses only when the bit is set.

1.8 Finding datalink-layer addresses

As we saw in Section 1.6, distributed name servers allow us to translate from a name to a network-level address. This network-level address is used to locate a unique destination in the network. However. the destination's host-interface card may only recognize and accept packets labeled with its datalink-layer address (this problem only arises in broadcast LANs; in point-to-point LANs, a destination does not require a datalink-layer address). Thus, the last router along the path to the destination must encapsulate an incoming IP packet in a datalink-layer header that contains a datalink-layer address corresponding to the final destination (Figure B3T20 bottom). Symmetrically, if a source connects to a router within an 802 LAN, it must encapsulate the outgoing packet with a datalink-layer header containing the datalink layer address of the router's host-interface. We usually call the datalink-layer address the *Medium Access Control* or *MAC* address.

If both the source and destination of a packet are on the same broadcast LAN, we can easily translate from an IP address to a MAC. This is because the authoritative translation from a network-layer (IP) address to a MAC address is always available from at least one of the hosts on the LAN. When a source wants a translation, it broadcasts a query on the LAN, and the computer that owns the network-layer address replies with its MAC address. The Internet protocol that carries out this operation is called the *Address Resolution Protocol* or *ARP*.

An ARP packet contains the MAC address of the sender and the IP address that the sender wants to resolve. The packet is addressed with the LAN's broadcast address. Every host on the LAN is required to listen to ARP broadcasts and respond to an ARP query containing its own IP address with its MAC address. The reply is sent to the querier's MAC address, available from the ARP request. The originator of an ARP reply saves the reply in an *ARP cache so* that future translation will not require a broadcast. Cache entries are discarded after a while, so that if the MAC address of a host changes, this will eventually be noticed by every other host on the LAN.

ARP works well only on broadcast LANs. In point-to-point LANs, ARP queries are replied to by an *ARP server,* which is essentially a name server. When a host boots up, it registers its MAC address with the ARP server. When a host wants to find out a destination's address, it asks the ARP server, instead of initiating a broadcast. This technique *was* used when carrying IP over ATM LANs.

1.9 Summary

Naming and addressing are essential to the operation of any network. Although names can be arbitrarily long and are (usually) human understandable, addresses are usually fixed-length and are meant to be easily parsed by routers. Both names and addresses must be unique, yet should not require a naming authority to consult with every other naming authority in the network. For these reasons, both names and addresses are organized hierarchically. Moreover, hierarchical addressing allows us to aggregate sets of computers when describing routes to them in a routing table. Naming and addressing are closely tied to routing, which we study in the next section.

The Internet uses IP addresses, which come in two flavors, version 4 and version 6. Version 4, which is the current standard, divides the address into two parts, called the network number and the host number. A variable number of bits can be assigned to each portion by using a subnet mask. Sets of addresses can also be grouped together to form a larger address space by specifying an address prefix. Moreover, a host can be dynamically assigned an IP address when it joins the Internet. These three innovations allow us to manage the v4 address space more effectively, but it is still running out of addresses. Version 6 addresses, which are four times the size of version 4 addresses, promise to solve the address scarcity problem once and for all.

Name resolution is the process by which names are associated with addresses. A common solution is to distribute this functionality among a set of name servers. Replicating servers and caching replies are two techniques to increase reliability and to avoid excessive name-resolution traffic.

In broadcast LANs, we need to translate a network-layer address to a datalink-layer address. This is done in the Internet with the Address Resolution Protocol. A host broadcasts a resolution request on its LAN and receives a reply from any system that knows the answer. The telephone and ATM networks do not need address resolution because they are carried over point-to-point networks that do not need a datalink-layer address.

2.0 Routing

2.1 Introduction

Routing is the process of finding a path from a source to every destination in the network. It allows users in the remotest part of the world to get to information and services provided by computers anywhere in the world. Routing is what makes networking magical: allowing telephone conversations between Botswana and Buenos Aires, and video clips from the space shuttle to be multicast to hundreds of receivers around the world! How does a network choose a path that spans the world? How does the routing system scale to describe paths to many millions of endpoints? How should the system adapt to a failed link? What if the user wants to choose a path that has the least delay, or the least cost, or the most available capacity? What if the users are themselves mobile, attaching to the network from different wired access points? These are the sorts of questions we will study in this section.

Routing is accomplished by means of *routing protocols* that establish mutually consistent *routing tables* in every router (or switch controller) in the network (Figure B3T24). A routing table contains at least two columns: the first is the address of a destination endpoint or a destination network, and the second is the address of the network element that is the next hop in the "best" path to this destination. When a packet arrives at a router (or when a call-setup packet arrives at a switch controller), the router or switch controller consults the routing table to decide the next hop for the packet.

Example 2.1

An example of a routing table for a toy network is shown in Figure B3T24. We see that the routing table for node 1 has one entry for every other node in the network. This allows it to choose the next hop for every possible destination. For example, packets that arrive at node 1 destined for node 10 are forwarded to node 2.

Notice that node 1 has only two choices: to forward a packet to 2 or to forward it to 3. This is a *local* routing choice. Yet this choice depends on the *global* topology, because the destination address by itself does not contain enough information to make a correct decision. For example, the shortest path from node 1 to node 6 is through 2, and the shortest path to 11 is through 3. Node 1, just by looking at the destination address "6," cannot decide that node 2 should be the next hop to that destination. *We conclude that any routing protocol must communicate global topological information to each routing element to allow it to make local routing decisions.* Yet global information, by its very nature, is hard to collect, subject to frequent change, and voluminous. How can we summarize this information to extract only the portions relevant to each node? This lies at the heart of routing protocols.

A routing protocol asynchronously updates routing tables at every router or switch controller. For ease of exposition, in the remainder of the bloc, we will refer to both routers and switch controllers as "routers." Note that switch controllers are called upon to route packets only at the time of call setup, so that they route connections, instead of packets.

2.2 Routing protocol requirements

A routing protocol must try to satisfy several mutually opposing requirements:

- *Minimizing routing table space:* We would like routing tables to be as small as possible, so that we can build cheaper routers with smaller memories that are more easily looked up. Moreover, routers must periodically exchange routing tables to ensure that they have a consistent view of the network's topology: the larger the routing table, the greater the overhead in exchanging routing tables. We usually require a routing table to grow more slowly than the number of destinations in the network.

- *Minimizing control messages:* Routing protocols require control message exchange. These represent an overhead on system operation and should be minimized.

- *Robustness:* The worst thing that a router can do is to misroute packets, so that they never reach their destination. (They are said to enter a *black hole.)* Routers in error may also cause loops and *oscillations* in the network. Black holes, loops, and oscillations are rare under normal conditions, but can show up if routing tables are corrupted, users specify incorrect information, links break or are restored, or routing control packets are corrupted. A robust routing protocol should protect itself from these types of problems by periodically running consistency tests, and by careful use of checksums and sequence numbers as described in bloc 4.

Example 2.2

If routing tables are inconsistent, loops can easily be formed. For example, router A may think that the shortest path to C is through B, and B may think that the shortest path to C is through A. Then, a packet to C loops back and forth between A and B until some other procedure (such as a "time to live" reaching zero, as described in bloc 4) detects the

loop and terminates the packet.

Oscillations can be caused if the routing protocol chooses paths based on the current load. Consider routers A and B connected by paths P1 and P2. Suppose P1 is heavily loaded and P2 is idle. The routing protocol may divert all traffic from P1 to P2, thus loading P2. This makes P1 more desirable, and traffic moves back to P1! If we are not careful, traffic oscillates from P1 to P2, and the network is always congested.

- *Using optimal paths: To* the extent possible, a packet should follow the "best" path from a source to its destination. The "best" path may not necessarily be the shortest path: it may be a path that has the least delay, the most secure links, or the lowest monetary cost, or one that balances the load across the available paths. Routers along the entire path must collaborate to ensure that packets use the best possible route to maximize overall network performance.

As always, these requirements represent trade-offs in routing protocol design. For example, a protocol may trade off robustness for a decrease in the number of control messages and routing-table space. Many common protocols trade off a dramatic reduction in routing-table space for slightly longer paths.

2.3 Choices

Designers of routing protocols have many mechanisms available to them. In this section, we will describe some commonly available choices for routing. These choices also represent a rough taxonomy to categorize routing protocols.

- *Centralized versus distributed routing:* In centralized routing, a central processor collects information about the status of each link (up or down, utilization, and capacity) and processes this information to compute a routing table for every node. It then distributes these tables to all the routers. In distributed routing, routers cooperate using a distributed routing protocol to create mutually consistent routing tables. Centralized routing is reasonable when the network is centrally administered and the network is not too large, as in the core of the telephone network. However, it suffers from the same problems as a centralized name server: creating a single point of failure, and the concentration of routing traffic to a single point.

- *Source-based versus hop-by-hop:* A packet header can carry the entire route (that is, the addresses of every router on the path from the source to the destination), or the packet can carry just the destination address, and each router along the path can choose the next hop. These alternatives represent extremes in the degree to which a source can influence the path of a packet. A source route allows a sender to specify a packet's path precisely, but requires the source to be aware of the entire network topology. If a link or a router along the path goes down, a sourcerouted packet will not reach its destination. Moreover, if the path is long, the packet header can be fairly large. Thus, source routing trades off specificity in routing for packet-header size and extra overhead for control messages. An intermediate solution is to use a *loose source route.* With loose source routes, the sender chooses a subset of routers that the packet should pass through, and the path may include routers not included in the source route. Loose source routes are supported in the IP version 4 and 6 headers.

- *Stochastic versus deterministic:* With a deterministic route, each router forwards packets toward a destination along exactly one path. In stochastic routing, each router maintains more than one next hop for each possible destination. It randomly picks one of these hops when forwarding a packet. The advantage of stochastic routing is that it spreads the load among many paths, so that the load oscillations characteristic of deterministic routing are eliminated. On the other hand, a destination may receive packets along the same connection out of order, and with varying delays. Consequently, modern networks usually use deterministic routing.

- *Single versus multiple path:* In single-path routing, a router maintains only one path to each destination. In multiple-path routing, a router maintains a *primary* path to a destination, along with *alternative* paths. If the primary path is unavailable for some reason, routers may send packets on the alternative path (with stochastic routing, routers may send packets on alternative paths even if the primary path is available). Single-path routing is used on the Internet, because maintaining alternative paths requires more routing table space. Telephone networks usually use multiple-path routing, because this reduces the call blocking probability, which is very important for customer satisfaction.

- *State-dependent versus state-independent:* With state-dependent or *dynamic* routing, the choice of a route depends on the current (measured) network state. For example, if some links are heavily loaded, routers may try to route packets around that link. With state-independent or *static* routing, the route ignores the network state. For example, a shortest-path route (where we measure the path length as the number of hops) is state independent. State-dependent routing usually finds better routes than state-independent routing, but can suffer from problems caused by network dynamics (such as the routing oscillations described earlier). It also requires more overhead for monitoring the network load. The Internet uses both state-dependent and stateindependent routing. Telephone network routing used to be state independent, but statedependent routing with multiple paths is now the norm.

Having broadly considered the choices in routing protocol design, the rest of the bloc deals with specific routing protocols that make a selection from the choices described earlier. The literature on routing (both in the telephone network and in the Internet) is vast. We focus on the study of the routing in the Internet.

2.4 Distance-vector routing

Telephone network routing is specialized to take advantage of the unique features of the telephone network, such as a predictable traffic flow, and a relatively small network core. Large packet networks, such as the Internet, present a very different environment. In the Internet, links and routers are unreliable, alternative paths are scarce, and traffic patterns can change unpredictably within minutes. It is not surprising that routing in the Internet, and in ATM networks, which are likely to have Internet-like characteristics, follows a different path. The two fundamental routing algorithms in packet-switched networks are *distance-vector* and *link-state.*

Both algorithms assume that a router knows (a) the address of each neighbor, and (b) the cost of reaching each neighbor (where the cost measures quantities like the link's capacity, the current queuing delay, or a per-packet charge). Both algorithms allow a router to find *global* routing information, that is, the next hop to reach every destination in the network by the shortest path, by exchanging routing information with only its neighbors. Roughly speaking, in a distance-vector algorithm, a node tells its *neighbors* its distance to *every* other node in the network, and in a link-state algorithm, a node tells *every* other node in the network its distance to its *neighbors.* Thus, both routing protocols are *distributed* and are suitable for large internetworks controlled by multiple administrative entities. In this section, we will focus on distance vector algorithms. We will study link-state algorithms in Section 2.5.

2.4.1 Distance-Vector Algorithm

In distance-vector routing, we assume that each router knows the *identity* of every other router in the network (but not necessarily the shortest path to it). Each router maintains a *distance vector,* that is, a list of <*destination, cost*> tuples, one tuple per destination, where *cost* is the current estimate for the sum of the link costs on the shortest path to that destination. Each router initializes the cost to reach all nonneighbor nodes to a value higher than the expected cost of any route in the network (commonly referred to in the routing literature as *infinity*[23]*). A* router periodically sends a copy of its distance vector to all its neighbors. When a router receives a distance vector from a neighbor, it determines whether its cost to reach any destination would decrease if it routed packets to that destination through that neighbor (Figure B3T31). It can easily do so by comparing its current cost to reach a destination with the sum of the cost to reach its neighbor and its neighbor's cost to reach that destination.

Example 2.3

In Figure B3T31, if router A has an initial distance vector of (<A, 0>, <B, 1>, <C, 4>, <D,-> we see that the arrival of a distance vector from B results in A updating its costs to C and D. If a neighbor's distance vector results in a decrease in a cost to a destination, that neighbor is chosen to be the next hop to get to that destination. For example, in Figure B3T31, the distance vector from B reduced A's cost to D. Therefore, B is the next hop for packets destined for D. A router is expected to advertise its distance vector to all its neighbors every time it changes.

We can show that even if nodes asynchronously update their distance vectors, the routing tables will eventually converge. The intuition behind the proof is that each router knows the true cost to its neighbors. This information is spread one hop with the first exchange of distance vectors, and one hop further on each subsequent exchange. With the continued exchange of distance vectors, the cost of every link is eventually known throughout the network. The distance-vector algorithm is also called the Bellman-Ford algorithm.

2.4.2 Problems and solutions with distance-vector routing

The distance-vector algorithm works well if nodes and links are always up, but it runs into many problems when links go down or come up. The root cause of problems is that when a node updates and distributes a distance vector, it hides the sequence of operations it used to compute the vector. Thus, downstream routers do not have sufficient information to figure out whether their choice of a next hop will cause loops to form. This will become clear when we look at the *count-to-infinity* problem.

Count-to-infinity

We illustrate this problem with the next example.

---

[23] Infinity = 16.

Example 2.4

Consider the simple network shown in Figure B3T33. Initially, A routes packets to C via B, and B uses its direct path. Now, suppose the BC link goes down. B updates its cost to infinity, and tells this to A. Suppose, in the meantime, A sends its distance vector to B. B notices that A has a two-hop path to C. Therefore, it updates its routing table to reflect this information, and tells A that it has a three-hop path to C. In the previous exchange, A discovered that B did not have a path to C any more, and had updated its table to reflect that. When B joyfully announces that it does indeed have a path, A updates its routing table to show a four-hop path to C. This process of increasing the hop count to C continues until the hop count reaches infinity, when both nodes realize that no route to C exists after all. Note that during the process of counting-to-infinity, packets from A or B destined to C are likely to loop back and forth between A and B. Thus, if the counting process takes a while, many packets may wander aimlessly in the network, making no progress, and causing congestion for everyone else. It makes sense to try to avoid counting to infinity.

Path vector

The reason for count-to-infinity is that when B updated its path to C to go through A, it did not realize that A's path to C was through B. In other words, the distance vector that A sent B hid the fact that B was on A's route to C. There are several ways to add information to the distance vector to solve this problem. One solution is to annotate each entry in the distance vector with the path used to obtain this cost. For example, in Step 2 of Figure B3T33, A can tell B that its cost to C is 2, and the path to C was C-B. When B sees this, it realizes that no route to C exists, and the count-to-infinity problem goes away. This solution is also called the *path-vector* solution, since routers annotate the distance vector with a path. The path-vector approach is used in the *border gateway protocol (BGP)* in the Internet core. Note that path vectors trade off a larger routing table and extra control overhead for robustness.

Split horizon

The problem with path vectors is that the vectors require large table sizes, which can prove expensive. Several other solutions to the count-to-infinity problem avoid this overhead. In one solution, called *split-horizon routing,* a router never advertises the cost of a destination to its neighbor N, if N is the next hop to that destination. For example, in Figure B3T33, this means that A does not advertise a cost for C to B because it uses B as its next hop to C. This trivially solves the count-to-infinity problem in Figure B3T33. However, split horizon works only when two adjacent routers count to infinity: it is ineffective when three routers mutually do so.

A variant of split-horizon, called *split horizon with poisonous reverse,* is used in the *Routing Information Protocol (RIP)* on the Internet. When A routes to C via B, it tells B that it has an *infinite* cost to reach C (with normal split horizon, A would not tell B of a path to C at all). Though this sometimes accelerates convergence, it does not prevent three-way counting to infinity.

Triggered updates

While the classical distance-vector algorithm prescribes that a router should advertise its distance vector every time it changes, this can lead to a flurry of updates every time a link cost changes. If, for example, the cost measures link delay, a router may update its distance vector quite often. To prevent this, most distance vector algorithms prescribe that distance vectors be advertised only once in about 30 seconds. This adversely affects the time taken to recover from a count-to-infinity situation. Consider the situation in Figure B3T33, where each node must count from 1 to infinity. If we define infinity to be 16, then it will converge only 15 * 30 seconds later = 7.5 minutes. The network will be in an unstable situation during this entire interval. To avoid this, we can trigger distance vector changes immediately after a link is marked down. This rapid propagation removes some race conditions required for count-to-infinity and is adopted in the Internet RIP protocol.

Source tracing

The key idea for source tracing is to augment a distance vector so that it carries not only the cost to a destination, but the router *immediately preceding* the destination. We can show that this information is sufficient for a source to construct the entire path to the destination.

When a router updates its distance vector, if its cost to a destination decreases, it replaces the preceding-router field for that destination in its routing table with the corresponding value in the incoming distance vector. Distance vector with source tracing is guaranteed to be loop free if routers follow the rule that if a router changes its notion of the next hop for a destination D, then it should use the same neighbor for all destinations for which D lies along the shortest path.

Example 2.5

Consider Figure B3T35 which shows the routing table for router 1. Suppose we want to trace the path to router 6. First, we locate 6 in the routing table and see that the preceding router on the path to 6 is 5. We now look for 5, and find 4 as the preceding router. Continuing in this fashion, it is easy to compute the path as 1-2-4-5-6. This allows us to get the same information as a path vector, but with very little additional table space.

DUAL

The *distributed update algorithm (DUAL) is* a technique to assure loop-free routing tables even in the presence of rapid changes in network topology. With DUAL, a router maintains a pared down version of the network topology by storing the distance reported by each of its neighbors to each destination (this is just the union of their distance vectors). If the cost from a particular router R to a destination D *decreases* because of the receipt of a distance vector from a neighbor N, then it is impossible for a loop to form if R updates its tables based on that distance vector. The reason is that if a loop forms, the reported cost to D from N in the incoming distance vector C2 must *include* the previously reported cost from R to D, Cl. Thus, C2 must be larger than Cl. If R updates its tables only for distance vectors such that C2 < Cl, then loops will not form.

Now, suppose R receives an update such that the distance to a destination increases because of an increase in a link's cost or because of a link failure. Then, R first checks if it can find a shorter path to this destination through another neighbor using its topology table. If not, *it freezes its* routing table and distributes the new distance vector to all its neighbors. The neighbors check whether this increases their cost to D. If so, they freeze their tables in turn and spread the vector to their neighbors. The computation *expands* in this manner until all the routers affected by the change (that is, all routers whose distance to any endpoint increases because of this change) know of it. If all the neighbors of a router already know of the change or are unaffected, they inform the router that they are done. The router unfreezes its state and informs the router that previously informed it of the change, which, in turn, propagates this information. Thus, the computation *contracts* until the router that first detected the change knows that the effect of the change has propagated to every router that ought to know of it. This is called a *diffusion computation.* It can be shown that the DUAL algorithm results in loop-free routing. DUAL is implemented in the Extended Interior Gateway Routing Protocol (EIGRP), a proprietary routing protocol from Cisco Systems.

2.5 Link-state routing

In distance-vector routing, a router knows only the cost to each destination or, sometimes, the path to the destination. This cost or path is partly determined on its behalf by other routers in the network. This hiding of information is the cause of many problems with distance-vector algorithms.

In contrast, the philosophy in link-state routing is to distribute the topology of the network and the cost of each link to all the routers. Each router independently computes optimal paths to every destination. If each router sees the same cost for each link and uses the same algorithm to compute the best path, the routes are guaranteed to be loop free. Thus, the key elements in link-state routing are a way to distribute knowledge of network topology to every router in the network, and a way to compute shortest paths given the topology. We will study these in turn.

2.5.1 Topology dissemination

Each router participating in the link-state algorithm creates a set of *link-state packets (LSPs)* that describe its links. An LSP contains the router's ID, the neighbor's ID, and the cost of the link to the neighbor. The next step is to distribute a copy of every LSP to every router using *controlled flooding*. The idea is that when a router receives a new LSP, it stores a copy of the LSP in an *LSP database,* and forwards the LSP to every interface other than the one on which it arrived. It can be shown that an LSP is never transferred over the same link twice in the same direction. Thus, if a network has *E* edges, flooding requires at most *2E* transfers.

Example 2.6

In Figure B3T39, A creates two LSPs, <A, B, 1> and <A, C, 4>. The other nodes in the network create similar LSPs. Let us trace the path taken by the LSP <A, B, 1> that originates from A. In the first step, the LSP reaches B, which in turn forwards it to C and D, but not to A, because it arrived from A. When C gets the LSP, it forwards it to A and D. A does not forward the LSP further, because its database already contains the LSP. If D got the LSP from B before it got it from C, D detects that the LSP is a duplicate and does nothing. Otherwise, it forwards it to B, who does nothing. Thus, in a few short steps, the LSP reaches every router in the network.

Sequence numbers

Although flooding is easy to understand when links and nodes stay up, as with distance vector algorithms, complexity creeps in when links or routers can go down. For example, in Figure B3T39, suppose link AB goes down. We would like the LSP corresponding to AB to be removed from all the other routers. Router B detects that link AB is down and

sends an LSP with an infinite cost for AB to all the other routers. The other routers must somehow determine that this LSP overrides the information already existing in their databases. Therefore, every LSP must have a sequence number, and LSPs with newer sequence numbers override LSPs with older sequence numbers. This allows us to purge the old LSPs from each router's database.

Wrapped sequence numbers

Unfortunately, every sequence number has finite length and therefore is subject to wraparound. We have to ensure that a new LSP that has a numerically lower (but wrapped-around) sequence number still overrides an old LSP that has a numerically higher sequence number. For example, if sequence numbers are three bits long, thus spanning the space from 0 to 7, we would like a newer LSP with sequence number 0 to override an older LSP with sequence number 7. We can solve this problem by using very large sequence numbers. Then it is almost certain that if the difference between the sequence numbers of an existing and an incoming LSP is large, so that the numerically smaller LSP is actually newer. For example, if sequence numbers are 32 bits long, then they span the space from 0 to 4,294,967,295. If an old LSP in the database has a sequence number toward the end of the space, say, 4,294,967,200, and a new LSP has a sequence number toward the beginning of the space, say, 20, then the difference is 4,294,967,180. We therefore declare the LSP with sequence number 20 to be the newer one. More precisely, if the LSP sequence number space spans *N* sequence numbers, sequence number *a* is older than sequence number *b* if:

$$a < b \text{ and } |b-a| < N/2,$$

$$\text{or } a > b \text{ and } |b-a| > N/2.$$

Initial sequence number

When a router starts, it must choose a sequence number such that its LSPs in other routers' databases are overridden. If the router does not know what LSPs it used in the past, it may risk flooding new LSPs that are always ignored. For example, with the 0 to 2^{32}-1 sequence space, the LSPs in the databases may have a sequence number 5. If the router comes back up and chooses to start numbering LSPs with sequence number 0, other routers ignore the new LSPs. There are two ways to solve this problem: *aging* and a *lollipop sequence space*.

Aging

With *aging*, the creator of an LSP sets a field in the LSP header to a maximum age (MAX_AGE). A router receiving this LSP copies the current age to a per-LSP counter in its database and periodically decrements it. If decrementing an LSP counter makes it zero, the router purges the LSP from its database. To preserve a consistent view of the network topology, the router should quickly request the rest of the network to discard this LSP. It does so by initiating flooding with the zero-age LSP. When a router gets an LSP with zero age and the latest sequence number, it purges the LSP and floods the zeroage LSP to its neighbors. This quickly restores consistency. After a purge of an LSP from a particular router, any subsequent LSP from that router will automatically enter the LSP database. Thus, if a newly booted router waits for a while before sending new LSPs, it knows that its old LSPs will be purged, and its new LSPs will override all LSPs from its previous incarnation.

Although this scheme does work, the choice of initial LSP age is problematic. We would like the latest LSP from a router to be flooded throughout the network before its previous one times out. Otherwise, some routers may purge the LSP from their databases before the new LSP reaches them, leading to inconsistent routes. To minimize the overhead of sending LSPs frequently, we should use a fairly large initial LSP age, on the order of an hour or so[24]. However, to allow purging of old LSPs, after rebooting, a router must wait for a time on the order of the initial LSP age before it can start sending new LSPs. So, we cannot simultaneously minimize control overhead and the dead time after a router reboots.

Lollipop sequence space

A better solution is for newly booted routers to use a sequence number that uniquely differentiates it from every other sequence number that it could have used in the past. Although this is impossible with a circular sequence space, we can achieve this using a *lollipop sequence space* (Figure B3T42). Here, we have partitioned the sequence space of size *N* into three parts: a negative space from *-N/2* to 0, the sequence number 0, and a positive space of size *N/2 - 1*. When a router comes up, it uses the sequence number *-N/2* for its LSPs, and subsequent LSPs use *-N/2 + 1*, *-N/2 + 2*, etc. When the sequence number becomes positive, subsequent sequence numbers wrap around in the circular part of the space. An LSP with sequence number *a* is older than an LSP with sequence number *b* if:

[24] If the initial LSP age is small, then the time interval between the creation of two LSPs must be small; otherwise, a distant router may time out an LSP before the next LSP reaches it. This increases routing overhead.

- $a < 0$ and $a < b$, or
- $a > 0$, $a < b$, and $b - a < N/4$, or
- $a > 0$, $b > 0$, $a > b$, and $a - b > N/4$

Note that $-N/2$ is therefore the oldest sequence number.

We add the rule that if a router gets an LSP from another router that has an older sequence number than the one in its database, it informs the other router of its sequence number. Because a newly booted router always generates a packet with the oldest sequence number, it is guaranteed to be told by its neighbors of the sequence number it had used before it crashed. It then jumps to a sequence number 1 larger than this, so that subsequent LSPs override its past LSPs. For example, as shown in Figure B3T42, a newly booted router starts with sequence number -4. If existing routers have an LSP from this router with a sequence number *2*, they inform the newly booted router of this. The newly booted router then uses 3 for its newer LSPs and continues to number packets as 3, 0, 1, 2, 3, 0, etc., until it boots again. This solution does not require the newly booted router to wait for its LSPs to require. Intuitively, the neighbors of a router act as a distributed memory recording its actions. By sending a unique packet (with sequence number $-N/2$), a newly booted router can access this memory to regain its past state.

Recovering from a partition

LSP databases remain coherent if the network does not partition into two or more fragments. However, when recovering from a partition, databases may become inconsistent. We illustrate this with the next example.

Example 2.7

Consider the network shown in Figure B3T44. Assume that at the start of time, all routers have consistent LSP databases. Now, suppose link 4-5 breaks. This partitions the network into two independent fragments. If links 7-8 and 1-2 break later, the databases in each fragment evolve independently of each other. For example, node *2* is unaware of the break in link 7-8. This does not pose a problem if 4-5 stays down. However, when it comes up, routers on each side of the partition must update their view of the other side; otherwise, routing loops are possible (for example, 2 may route packets to 8 via 7, not knowing that link 7 - 8 is down).

Routers on each side of the newly restored link cooperate to restore LSP databases. Each LSP in the database is associated with a link ID and a version number. Routers increment the version number each time the LSP value changes. A set of *database descriptor records*, also maintained in the database, describe the link IDs and version numbers in the database. Database descriptor records are like link-state packets, except that they store far less information, so that exchanging these records is less expensive than exchanging LSPs. When a link comes up, routers at each end exchange a complete set of database descriptor records. By comparing them, each determines the set of records that are either nonexistent or out-of-date in their database. They request their peer router to send them these LSPs, which they then flood into their respective fragments. This restores a uniform view of the topology to the entire network.

Link or router failure

When a link fails, the routers on either side notice this and can flood the network with this information. Thus, link failures are relatively easy to recover from. However, when a router fails, there is no direct way to detect this. Most link-state protocols require routers to exchange HELLO packets with their neighbors. If a router does not respond to a series of HELLOs, it is likely to be down. The neighbors should immediately flood this information.

It is possible to construct scenarios where, because of a series of failures, the HELLO protocol does not detect a dead router (for example, an undetectable corruption in the source address may make a HELLO packet from a router that is alive look like a HELLO from a dead router). To prevent databases from becoming corrupted without explicit failure detection, LSP records are usually aged (even with lollipop-space sequence numbers). When an LSP times out at some router, the router immediately floods the network with a special packet that informs every other router that the LSP timed out, and that they should delete this (stale) LSP from their LSP database. This allows the network eventually to recover from almost every possible sequence of failures.

Securing LSP databases

Loop-freeness in link-state routing requires that all routers share a consistent view of the network. If a malicious agent injects spurious LSP packets into a router, routing becomes unstable. Thus, routers must actively protect their LSP database not only from corruption, but also from malicious interference. Several techniques for securing LSP databases are well known. First, link-state packets are protected by a checksum, not only when sent over a transmission link, but

also when stored in the database. This detects corruption on the link or on a disk. Second, the receiver acknowledges LSP exchanges, so that a sender can recover from link losses using timeouts and retransmissions. Third, LSP exchanges are authenticated by a password known only to routing administrators. This makes it harder for malicious users to inject LSPs into a database. Several other techniques to ensure security in LSP exchange are described in reference.

2.5.2 Computing shortest paths

Thus far, we have seen how every router in the network obtains a consistent copy of the LSP database. We now study how a router can use this database to compute optimal routes in the network. A router typically uses Dijkstra's shortest-path algorithm to do so.

Dijkstra's algorithm

Dijkstra's algorithm computes the shortest path from a *root* node (corresponding to the router where the algorithm is being run) to every other node in the network. The key idea is to maintain a set of nodes, P, for which the shortest path has already been found. Every node outside P must be reached by a path from a node already in P. We find out every way in which an "outside" node *o* can be reached by a one-hop path from a node already in P, and choose the shortest of these as the path to *o*. Node *o* can now be added to P, and we continue in this fashion until we have the shortest path to all the nodes in the network.

More precisely, we define two sets P and T (standing for permanent and temporary). Set P is the set of nodes to which shortest paths have been found, and set T is the set of nodes to which we are considering shortest paths. We start by initializing P to the current node, and T to null. The algorithm repeats the following steps:

1. For the node *p* just added to P, add each of its neighbors *n* to T such that (a) if *n* is not in T, add it, annotating it with the cost to reach it through *p* and *p*'s ID, and (b) if *n* is already in T and the path to *n* through *p* has a lower cost, then remove the earlier instance of *n* and add the new instance annotated with the cost to reach it through *p* and *p*'s ID.

2. Pick the node *n* that has the smallest cost in T and, if it is not already in P, add it to P. Use its annotation to determine the router p to use to reach *n*. If T is empty, we are done.

When the algorithm stops, we have, for each router, the router on the shortest path used to reach it. As we did with source tracing for distance-vector routing, this allows us to compute the next hop on the shortest path for every destination in the network. Figure B3T48 shows an example of Dijkstra's algorithm.

2.5.3 Link state versus distance vector

Given a choice between link-state and distance-vector routing, which style should we prefer? Arguments on this issue among experts often parallel the medieval discussion of how many angels could dance on the head of a pin! Conventional wisdom is that link state algorithms are more stable because each router knows the entire network topology. On the other hand, transient routing loops can form while the new topology is being flooded. If the network is so dynamic that links are always coming up or going down, then these transients can last for a long time, and the loop-free property is lost. Moreover, as we have seen, simple modifications to the vanilla distance-vector algorithm can prevent routing loops. Thus, one should not prefer link-state protocols for loop-freeness alone.

A second argument in favor of link-state algorithms is that they allow multiple routing metrics. The idea is that each LSP can carry more than one cost. Thus, each router can compute multiple shortest-path trees, one corresponding to each metric. Packets can then be forwarded on one of the shortest-path trees, which they can select with a flag in the header. For example, an LSP may carry a delay cost and a monetary cost. This would allow every router to compute a shortest-delay tree and a lowest-monetary-cost tree. Incoming packets that prefer lower delays (and, perhaps, are willing to pay for it) would be routed according to the shortest-delay path.

Although this sounds attractive at first, it assumes that every router will agree to report the same set of metrics. If some routers do not report some metrics, this is not a disaster if all the other routers assign it a consistent default. However, the benefits from multiple metrics seem more tenuous if a considerable fraction of the routers along the path choose not to report one or more metrics of interest. Moreover, the benefits of multiple metrics can be realized by path-vector-type distance-vector algorithms.

Third, we prefer link-state algorithms because, after a change, they usually converge faster than distance-vector algorithms. It is not clear that this holds if we use a distance-vector algorithm with triggered updates and one of the several algorithms to ensure loop-freeness (and therefore, absence of counting to infinity). Convergence depends strongly on the network topology, the load on the routing protocol, and the exact sequence of link failure and recovery. Thus, it is impossible to argue convincingly for either link state or distance vector.

Distance-vector algorithms do seem to have two advantages over link-state algorithms. First, much of the overhead in link-state routing is in the elaborate precautions necessary to prevent corruption of the LSP database. We can avoid these in distance-vector algorithms because we do not require that nodes independently compute consistent routes. Second, distance-vector algorithms typically require less memory for routing tables than do link-state protocols. Again, this is because they do not need to maintain an LSP database. On the other hand, this advantage disappears when we use path-vector-type distance-vector algorithms.

Because there is no clear winner, both distance-vector and link-state algorithms are commonly used in packet-switched networks. For example, in the Internet, the two "modern" routing protocols are Open Shortest Path First (OSPF), which is a link-state protocol, and Border Gateway Protocol (BGP), which is a path-vector protocol (more about these in Section 2.9). Examples of both algorithms will probably exist in datagram networks for many years to come.

2.6 Choosing link costs

Thus far, we have assumed that network administrators somehow assign a reasonable cost to each link in the network, which they then distribute to other routers in the network. We have not really considered how the choice of a link cost affects the flow of traffic in the network. As we see next, the cost of a link and the load on it are coupled in a manner reminiscent of the Erlang map, which couples routing strategies and blocking probabilities in the telephone network. The key idea is that the choice of link costs implicitly defines the way in which traffic load is distributed in the network. The lower the cost of a given link, the higher the probability that it is a part of a shortest path to some destination, and the higher the expected load on it. Therefore, if link costs depend on the current load on the link (which is usually a good idea), a high cost lowers the load on the link, which, in turn, lowers its cost. Our goal is to choose an appropriate cost function so that the load and cost converge on a desirable fixed point. A poor cost function leads to routing oscillations, which are highly undesirable.

2.6.1 Static metrics

For the moment, let us ignore the dynamics of routing and focus on the simplest possible way to assign weights, the *hop-count* metric. Here, we give every link a unit weight, so that the shortest-cost path is also the path with the smallest hop count. Allocating all links a unit weight is reasonable when the links are homogeneous. However, it makes little sense if some links run at DS3 speeds (45 Mbps), while others are DS1 (1.5 Mbps). Here, we should probably give links with lower bandwidth a higher cost, so that the load is mostly carried on high-capacity links. This is illustrated in the next example.

Example 2.8

Consider the network in Figure B3T52. Here, links AB, AC, and BD are T3 links, and BC and CD are T1 links. If we assign all links a unit cost, then traffic from B to C will go over the BC link. All other things being equal, it is a better idea to route B-C traffic on the path B-A-C, because it has nearly thirty times the capacity. Therefore, we could assign T1 links a weight of 10, and T3 links a weight of 1. Then, links BC and CD are never used (unless one or more of the T3 links goes down). Unfortunately, even if link AB is highly congested and BC is idle, traffic will still take the B-A-C path instead of the BC path. This points out the inherent problems of statically assigning weights to links. It *may* be a better idea to assign link costs *dynamically*, based on the current load on the link.

2.6.2 Original ARPAnet dynamic metrics

One of the earliest dynamic cost allocation techniques was used in the original ARPAnet. In this scheme, the cost of a link is directly proportional to the length of a router's output queue at the entrance to that link. If a link has a long queue, no matter the link's capacity, it is considered overloaded and given a higher cost. Continuing with Example 2.8 and Figure B3T52, assume that link A-B was heavily loaded in the A-to-B direction. Then, the queue at router A for that link would be long. If A therefore advertises a higher cost for A-B, this would divert the C-to-B traffic to the path C-B from C-A-B, reducing the load on A-B.

Although the idea of a dynamic link cost is a good one, the original ARPAnet implementation is a case study in the unintended consequences of a complex design. In its defense, the scheme did work well when the network was lightly loaded. However, many problems appeared under a heavy load. First, the link cost depended on the queue length averaged over 10 seconds. Since the backbone ran at only 56 Kbps, this represented too small a time granularity at which to measure queue lengths. Thus, tran sient spikes in the queue length could trigger major rerouting in the network. Second, link costs had a wide dynamic range (that is, they could be very low or very high). Consequently, it turned out that the network completely ignored paths with high costs. Although we should avoid high-delay links, they should not be unused! Third, the queue length was assumed to be a predictor for future loads on the link. In other words, if the queue length was long, the link cost was increased in the expectation that the link would continue to be overloaded in the future. In fact, the opposite was true. When a link's cost was large, it was avoided, so that when routes

were recomputed, the link's load dramatically decreased. Fourth, there was no restriction on the difference between successively reported costs for a link. This allowed link costs to oscillate rapidly. Finally, all the routers tended to recompute routing tables simultaneously. Thus, links with low costs would be chosen to be on the shortest path simultaneously by many routers, flooding the link.

2.6.3 Modified ARPAnet metrics (or Dynamic metrics II)

The modified version of the ARPAnet link-cost function avoided many errors made in the first version and was much more successful. In this scheme, link costs are a function not only of the measured mean queue length, but also of the link's capacity. When the link's load is low, its cost depends entirely on the link's capacity, and the queue length comes into play only at higher loads. Thus, at low loads, network routing essentially uses static costs, making it stable. Moreover, link costs are hop *normalized*, that is, the weight of a link is measured in "hops". Traversing a link with a weight c is as expensive as traversing c links with unit weight. The higher the advertised hop-normalized cost, the greater the barrier to using the link, but the barrier is not overwhelmingly high.

Two schemes were also added to dampen the oscillation in link costs. First, the dynamic range of link costs was reduced from a range of 127:1 to 3:1 (the worst cost a link can advertise is that it equals 3 hops). Second, a router was allowed to change the link cost by only half a hop in successive advertisements. With these changes, and a few others, routing oscillations were nearly eliminated even under heavy load.

2.6.4 Routing dynamics

We mentioned earlier that the load on a link and the probability of its use in shortest-path routes are tightly coupled. We illustrate this by using two functions called the *metric map* and the *network response map*. The metric map translates the load on a link to its link-cost metric and is the link-cost function we described in the previous paragraph. The network response map translates from a given link metric to the expected load on that link, given the current topology and traffic load. Although the metric map is precisely defined, the network response map is empirically determined by modifying the cost of one link at a time and measuring the additional traffic due to that change, then averaging this over all links. We show the general form of these maps in Figure B3T55. In Figure B3T55 top (a) we see that as the load increases, the link cost first is flat (as explained earlier), and then rises linearly to 3, where it saturates. In Figure B3T55 top (b), we show a family of curves, each corresponding to an overall network load, which plot the load on an "average" link as a function of the cost of that link. We see that as the link cost increases, the mean load on a link decreases from 1.2, when the cost is 0, to nearly 0, when the cost is 5.

We can envision the dynamics of routing in the network by putting these two maps together, as shown in Figure B3T55 bottom. Paths in this map show the evolution of the system. We choose an arbitrary initial load in the system, and compute the corresponding cost metric by drawing a horizontal line and choosing its intercept on the metric curve. The corresponding load in the next time step can now be obtained by drawing a vertical line through that metric and noting the intercept on the load curve. By repeating these steps, we can determine the dynamics of the system starting from an arbitrary initial load.

For example, consider the path marked *a* in the figure. This represents a link at equilibrium, where the link load and its cost metric suffer from a bounded oscillation. Note that the link cost is allowed to change by only half a hop in successive advertisements, which tightly bounds the range of oscillations. This fact is dramatically illustrated by the path marked *b*. Here, we see the effect of introducing a new link into the system. We artificially start the link off with a high cost, reducing the cost by half a hop each time step. Because of the high initial cost metric, the initial load on the link is low. Each subsequent advertisement therefore reduces the metric by half a hop, gradually increasing the load. The network eventually stabilizes with a small, bounded oscillation. If link costs were allowed to change by larger amounts in successive advertisements, link loads and metrics would suffer from large oscillations. System evolution diagrams such as these are very useful in evaluating heuristics for link-cost metrics.

2.7 Hierarchical routing

If a network with N nodes and E edges uses link-state routing, it can be shown that computing shortest paths takes $O(E \log E)$ computation at each router, and the routing table requires $O(N)$ storage. E is at least the same size as N, because even for a tree-shaped graph, which requires the smallest number of edges for a given number of nodes $E = N - 1$. Clearly, the computation and space requirements for a routing protocol become excessive when N is large. Because both the Internet and the telephone network are expected to grow to several billion endpoints, we must use *hierarchical routing* to rein in routing costs.

We alluded to hierarchical routing when we discussed hierarchical addressing and address aggregation in Section 1. The idea is to partition the network into multiple hierarchical levels. A handful of routers in each level are responsible for communication between adjacent levels. Thus, at each level, only a few hundred routers need to maintain shortest-path

routes to each other. A router that spans a hierarchy boundary agrees to route packets from the rest of the network to every router in its "area", and from every router in its area to the rest of the network.

2.7.1 Features of hierarchical routing

Figure B3T59 shows a detailed view of how a nation-wide Internet Service Provider might put together a hierarchically routed network. First, note that we have partitioned the network into four routing levels. Each level contains only a few routers, thus making it easy to compute routing tables.

Second, the network is not a strict hierarchy, because more than one router may advertise reachability to the same part of the address space. For example, both routers in Los Angeles, named LA0 and LA1, advertise that they can carry traffic for addresses of the form 6.*. When a packet with an address in 6.* arrives at San Francisco, it should forward the packet to LA0, but the router at Atlanta should forward it to LA1. Making the hierarchy looser lets the network survive faults more easily. For example, if LA0 went down, traffic for destination in 6.* could be routed via LA1.

Third, routers that span levels, such as 6.0.0.0 and 6.4.0.0, participate in routing protocols at both levels. For example, router 6.4.0.0 discovers, using a level-3 routing protocol, that the shortest path to networks advertised by router 6.3.0.0 is its direct link to 6.3.0.0. If the link goes down, level-3 routing informs 6.4.0.0 that it should use router 6.1.0.0 instead. 6.4.0.0 also participates in level-2 routing to find, for instance, that the shortest path to 6.4.2.0 is through 6.4.3.0. The routing protocols at the two levels may be completely different (one may be link-state, and the other distance-vector). Therefore, routers that route between levels must be prepared to speak multiple protocols.

Finally, note that we have a router in level 3 marked 21.1.2.3. Why would a router with address 21.* be placed under a router with address 6.*? This might be because of address-space exhaustion in the 6.* space. Or, a company that had obtained the address space 21.1.2.* for its computers from a service provider with authority over 21.*, then moved to a new location, might want to retain its old addresses (because renumbering computers on the Internet requires manual reconfiguration of every end-system and router). In any case, router 6.2.0.0 must advertise reachability to 21.1.2.3. Moreover, routers 6.0.0.1 and 6.0.0.2 at the network core must also advertise that they can route to 21.1.2.3, and every other router in the core must know that packets for 21.1.2.3 must be forwarded to one of these two routers. The lesson is that if we introduce a router at a lower level whose address cannot be aggregated into an existing address space, then each router in the core of the network must contain a routing table entry for it.

In the current Internet, addresses obey a three-level hierarchy (network number, subnet number, and host number). Because the highest possible degree of aggregation of addresses is at the network level, routers in the network core, which benefit most from aggregation, advertise routes to networks. For example, routers at the core will usually advertise routes to network 135.104.*, instead of to 135.104.53, 135.104.52, etc., which are subnets within 135.104. This approach to aggregation works well when the number of networks is small and routers at a lower level can handle routing within a network. Unfortunately, because of exhaustion of Class B addresses, many networks received multiple Class C network addresses instead. Consequently, routers in the core need to store table entries for routes to thousands of Class C networks. Each core router carried routes to more than 80,000 networks. Thus, even if addresses are hierarchical, they must be carefully managed, or routing tables and route computation can still be expensive. The CIDR scheme for addressing, discussed in Section 1, alleviates some of these problems.

2.7.2 External and summary records

Consider the four level-3 routers in Figure B3T59 with addresses 6.1.0.0, 6.2.0.0, 6.3.0.0, and 6.4.0.0. Suppose they use link-state routing to compute routes. What should be the next hop for a packet arriving at 6.4.0.0 that is destined to an address in 5.*? From the topology of the network, note that if the network uses a least-hop cost metric, then the next hop should be 6.3.0.0. Thus, we want router 6.4.0.0 to discover that there is a 3-hop path through router 6.3.0.0 to 5.0.0.0, whereas the path through 6.2.0.0 is at least 4 hops long. Unfortunately, since router 5.0.0.0 is not part of the level-3 network, 6.4.0.0 would not ordinarily be aware of its existence. We need a mechanism that allows routers that participate in level-4 routing to advertise paths to routers that are external to the level-3 network. This is done by using *external records* in the LSP database.

For example, router 6.0.0.0, which knows that it has a 1-hop path to 5.0.0.0 using level-4 routing, creates an *external LSP* that advertises a link to 5.0.0.0 with a link cost of 1 and floods this within the level-3 network. Similarly, 6.0.0.1 uses level-4 routing to learn that its least-cost path to 5.0.0.0 is 2 hops and floods this information in the level-3 network. Thus, routers 6.0.0.1 and 6.0.0.2 pretend that 5.0.0.0 is a level-3 router that happens to be connected to them with a 1- and 2-hop path, respectively. When this information is propagated within the level-3 network, 6.4.0.0 automatically discovers that its shortest path to 5.0.0.0 is through 6.3.0.0, as we wanted. External LSPs, therefore, allow optimal routes to be computed despite the information-hiding inherent in hierarchical routing.

Summary records

An external record allows routers within level 3 to discover shortest paths to external networks. The symmetrical problem is for external networks to discover shortest paths to level-3 networks that are not visible at level 4. This is done using summary records. For example, 6.0.0.0 advertises to level-4 routers that it has a path to 6.1.0.0 that is of length 2, to 6.2.0.0 of length 3, to 6.3.0.0 of length 1, and to 6.4.0.0 of length 2. It is as if these are single links with higher costs. Level-4 routers do not need to know the exact topology within the level-3 network, just the costs. Note that cost information in a summary record is functionally equivalent to a distance vector, because it summarizes the distance from a level-4 router to every level-3 router connected to it.

The network uses summary records to compute optimal paths. For example, the Atlanta router knows from 6.0.0.1's summary records that it has a 1-hop path to 6.2.0.0. It also knows (from level-4 routing) that it has a 1-hop path to 6.0.0.1. Therefore, its cost to reach 6.2.0.0 through 6.0.0.1 is 2 hops. In contrast, its path to 6.2.0.0 via 6.0.0.0 is (from these same sources of information) 5 hops. Therefore, it routes packets destined to 6.2.0.0 through 6.0.0.1, as we wanted.

Continuing with our example, the LSP database at router 6.0.0.0 therefore contains the following:

- LSP records for every link in its level-3 network
- LSP records for every link in its level-4 network
- External records that summarize its cost to reach every router in the level-4 network
- Summary records for virtual links that connect it to every level-3 router in its area
- Summary records received from other level-4 routers for virtual links to their level-3 routers

These records allow it to compute optimal paths not only to other level-4 routers, but also to level-3 routers within other level-4 networks.

2.7.3 Interior and exterior protocols

In the Internet, we distinguish between three levels of routing (corresponding roughly to the three-level address hierarchy), where we allow each level to use a different routing protocol. The highest level is the Internet backbone, which interconnects multiple autonomous systems (ASs) (Figure B3T63). Routing between autonomous systems uses the exterior gateway protocol. The name reflects the history of the Internet, when a gateway connected university networks to the ARPAnet. The protocol that gateways spoke to each other therefore was the exterior gateway protocol. Symmetrically, the protocol that the gateway spoke to routers within a campus (and now, within an AS) is called the interior gateway protocol. At the lowest level, we have routing within a single broadcast LAN, such as Ethernet or FDDI, In this section, we will discuss the requirements for interior and exterior protocols, and problems with their interconnection.

Exterior protocols

Although all the routers within an AS are mutually cooperative, routers interconnecting two ASs may not necessarily trust each other. Exterior protocols determine routing between entities that can be owned by mutually suspicious domains. An important part of exterior protocols, therefore, is configuring *border gateways* (that is, gateways that mediate between interior and exterior routing) to recognize a set of valid neighbors and, valid paths. This is illustrated in Figure B3T63.

Example 2.9

In Figure B3T63, assume that border routers A and B belong to AT&T, and router D belongs to MCI. Say that the AB link goes down. A can still reach B through D, and a generic link-state routing protocol will easily find this path. However, the thought that internal AT&T packets traverse MCI's router may upset both MCI's and AT&T's managements! Therefore, the exterior protocol must allow A and B to state that if the A-B link goes down, the A-D-B path is unacceptable. Of course, for packets destined to D, the A-D link is perfectly valid, and, similarly, the D-B link may also be independently valid. It is only their combination, A-D-B, that is prohibited. Accounting for administrative issues such as these complicates the design of exterior routing protocols, and these protocols often require manual configuration and intervention.

A related problem is that of *transit.* Suppose autonomous system A and autonomous system C set up a backdoor link between A.2 and C.1 for their own purposes. Since B knows from its interior routing protocol that C.1 is reachable through the backdoor link, it might advertise this to the rest of the Internet. This might cause A's facility to be used for packets destined for neither A nor C, which might annoy their administrators. Therefore, B should know that some links

advertised by an interior protocol are special and should not be advertised (summarized) externally. This is another problem that usually requires manual intervention.

Exterior gateway protocols must be suspicious of routing updates. It should not be possible for malicious users to bring down the Internet by sending spurious routing messages to backbone gateways. Typically, every routing exchange is protected by a link password. Routing updates that fail the password check are rejected.

Interior protocols

Interior protocols are largely free of the administrative problems that exterior protocols face. just as autonomous systems hierarchically partition the Internet at the top level, interior routing protocols typically hierarchically partition each AS into *areas.* However, the same interior protocol routes packets both within and among areas. The issues in generating external and summary records, which we studied in Section 2.7.2, apply to routing among areas, in the same way as they do to routing between autonomous systems.

Issues in interconnecting exterior and interior routing protocols

The key problem in interconnecting exterior and interior protocols is that they may use different routing techniques and different ways to decide link costs. For example, the exterior protocol may advertise a 5-hop count to another AS. However, each of these hops may span a continent and cannot be compared with a 5-hop path in the interior of the AS. How is a router to decide which is the shortest path when routers use link costs that cannot be compared? The solution is to use the least common denominator, usually a hop-count metric, when computing routes outside the AS. This is not necessarily the optimal path, but at least it is a path that works!

A similar problem arises if the interior and exterior routing protocols use different routing schemes. For example, the exterior protocol may use path-vector routing, and the interior may use link-state routing. Thus, the border gateway must convert from an LSP database to a set of distance vectors that summarize paths to its interior. In the other direction, it must convert from distance-vector advertisements to external records for the interior routing protocol. Things are easier if both the interior and exterior routing protocols use the same basic routing scheme.

The bottom line is that interconnecting a given interior and exterior protocol requires a fair amount of manual intervention, and frequent monitoring to ensure that the network stays up. This is a direct consequence of the heterogeneity in the administration of the Internet, and of its decentralized control.

2.8 Common routing protocols

This section presents a highly abbreviated introduction to Internet routing protocols. Details on Internet routing can be found in references famous book wrote by Christian Huitema, Le routage dans l'Internet.

The Internet distinguishes between interior and exterior routing protocols because of the different demands that they pose on the routing system. Two protocols are commonly used as interior protocols. These are the Routing Information Protocol (RIP) and the Open Shortest Path First protocol (OSPF). The protocols commonly used for exterior routing are the Exterior Gateway Protocol (EGP) and the Border Gateway Protocol (BGP).

2.8.1 RIP

RIP, a distance-vector protocol, was the original routing protocol in the ARPAnet. It uses a hop-count metric, where infinity is defined to be 16. Peer routers exchange distance vectors every 30 s, and a router is declared dead if a peer does not hear from it for 180 s. The protocol uses split horizon with poisonous reverse to avoid the count-to-infinity problem. RIP is useful for small subnets where its simplicity of implementation and configuration more than compensates for its inadequacies in dealing, with link failures and providing multiple metrics.

2.8.2 OSPF

OSPF, a link-state protocol, is the preferred interior routing protocol on the Internet. It uses the notion of *areas* to route packets hierarchically within an AS. It also uses all the techniques for achieving LSP database consistency described in Section 2.5. Consequently, it is rather complex to describe and implement.

2.8.3 EGP

The original exterior protocol in the Internet was the distance-vector-based *Exterior Gateway Protocol* or EGP. EGP allows administrators to pick their neighbors in order to enforce inter-AS routing policies. To allow scaling, EGP allows address aggregation in routing tables.

EGP routers propagate distance vectors that reflect a combination of preferences and policies. For example, in the NSFnet backbone, a router advertises the distance to another AS as 128 if the AS is reachable, and 255 otherwise. This reduces EGP to a reachability protocol rather than a shortest-path protocol. Therefore, unless we structure the network backbone as a tree, EGP leads to routing loops! The reason for choosing 128 as the standard inter-AS distance is that it enables *backdoors.* Backdoors between autonomous systems are always given a cost smaller than 128, so that the two ASs sharing the backdoor will use it while keeping the backdoor invisible to outside systems. EGP is no longer widely used because of many deficiencies, particularly its need for loop-free topologies.

2.8.4 BGP

The preferred replacement for EGP is the Border Gateway Protocol, version 4, commonly referred to as BGP4. BGP4 is a path-vector protocol, where distance vectors are annotated not only with the entire path used to compute each distance, but also with certain policy attributes. An exterior gateway can usually compute much better paths with BGP than with EGP by examining these attributes. Since BGP uses true costs, unlike EGP, it can be used in non-tree topologies. Its use of a path-vector guarantees loopfreeness, at the expense of much larger routing tables. BGP routers use TCP to communicate with each other, instead of layering the routing message directly over IP, as is done in every other Internet routing protocol. This simplifies the error management in the routing protocol. However, routing updates are subject to TCP flow control, which can lead to fairly complicated and poorly understood network dynamics. For example, routing updates might be delayed waiting for TCP to time out. Thus, the choice of TCP is still controversial.

If an AS has more than one BGP-speaking border gateway, path vectors arriving at a gateway must somehow make their way to all the other gateways in the AS. Thus, BGP requires each gateway in an AS to talk to every other gateway in that AS (also called *internal peering).* BGP4 is hard to maintain because of the need to choose consistent path attributes from all the border routers, and to maintain clique connectivity among internal peers.

2.9 Routing within a broadcast LAN

Thus far we have looked at the routing problem for the network as a whole. In this section, we view the routing problem from the perspective of an endpoint-that is, how should the routing module at an endpoint decide where to forward a packet that it receives from an application?

An endpoint connected to a router by a point-to-point link (as in an ATM network) simply forwards every packet to that router. However, if the endpoint is part of a broadcast LAN, we can exploit the LAN's inherent routing capacity to reduce the load on routers. Specifically, the routing module must make four decisions:

- Is the packet meant for a destination on the same LAN?
- If so, what is the datalink-layer (MAC) address of the destination?
- If not, to which of the several routers on the LAN should the packet be sent?
- What is the router's MAC address?

Example 2.10

Consider host H1 shown in Figure B3T73 If it wants to send a packet to H2, it should determine that H2 is local, then figure out H2's MAC address. If it wants to send a packet to H3, it should find out that the next hop should be R1. If R1 goes down, it should send the packet to R2. Similarly, packets for H4 should go to R2, unless it is down, in which case it should be sent to R1.

These decisions typically require a combination of addressing conventions, explicit information, and exploiting the broadcast nature of the LAN. In the Internet, the first problem is solved by agreeing that all hosts that have the same network number must belong to the same broadcast LAN. (Although a single physical LAN may carry more than one IP subnet, hosts on different subnets on the same LAN communicate only through a router.) Thus, a host can determine whether the destination is local simply by using its subnet mask to extract the network number of the destination and comparing this with its own network number. For example, if host 135.104.53.100, with a subnet mask of 255.255.255.0, wants to send a packet to 135.104.53.12, it uses the subnet mask to determine that the destination's network number is 135.104.53. Since this matches its own network number, the destination must be on the local LAN.

The sending host must next determine the MAC address of the host to which it wants to send. It does so using the Address Resolution Protocol described in Section 1. It installs the MAC address in a local ARP cache and uses it for further transmission.

2.9.1 Router discovery

If a packet's destination address is nonlocal, then the host must send the packet to one of the routers on the LAN. A host can discover all the routers on the local LAN by means of router advertisement packets that each router periodically broadcasts on the LAN. A router advertisement has a preference level and a time-to-live. Hosts first check that the router corresponds to their own subnet by masking the router's IP address with their subnet mask and comparing with their subnet number. They then install a default route to the router with the highest preference. All nonlocal packets are sent to the default router, if necessary, resolving the router's MAC address with an ARP request.

A router advertisement is placed in a cache and flushed when its time-to-live expires. The time-to-live is typically around half an hour, and routers send advertisements about once every 10 minutes. The idea is that if a router dies, the host deletes old state information automatically. If a newly booted host does not want to wait several minutes for a router advertisement, it can force all routers to send an advertisement using a router solicitation packet.

If a default router goes down, then the host must somehow determine this and switch to an alternative router, if one exists. Otherwise, all packets from the host will be lost without trace (the *black hole* problem). The Internet protocol suite does not specify any single algorithm to cover black hole detection, though several heuristics were proposed in the literature. The general idea is that if a host does not hear anything from a router (such as a reply to an ARP request) for some time, it should assume that the router is down, and it can force routers to identify themselves with a router solicitation message. To prevent network load, hosts are required to send no more than three solicitation messages before they give up and assume that no router is available.

2.9.2 Redirection

With a default route, a host sends all nonlocal packets to only one of possibly many routers that share its subnet on the LAN. It may happen that, for a particular destination, it ought to use another router. To solve this problem, if a host's default router is not the right choice for a given destination, the default router sends a control message (using the *Internet Control Message Protocol* or ICMP) back to the host, informing it of a better choice. This *redirect* message is stored in the host's routing table for future use.

Example 2.11

In Figure B3T73, host HI may have selected R1 as its default router. It may then send a packet for H4 to R1. R1 can reach R4 either through the broadcast LAN and R2, or through R3. Assume, for the moment, that R1's next hop to R4 is through R2. When R1 gets a packet for H4, it can detect that R2 is a better routing choice for HI, because R1's next hop for H4 is R2, which has the same network address as R1 and Ell. It therefore sends an ICMP redirect message to H1, asking it to use R2 in the future, and hands the packet to R2 for transmission. In this way, hosts automatically discover the best path to remote destinations.

2.10 Summary

In this section, we studied many aspects of routing in the Internet in detail. The hard problem in routing is summarizing volatile and voluminous global state to something that a router can use in making local decisions. This problem exists both in the Internet and in the telephone and ATM networks. However, in the latter two networks, switch controllers route calls, instead of packets.

We would like a routing protocol to be robust, minimize its use of memory in routers, choose optimal paths, and require the least overhead. We have several choices in designing such a protocol, including centralized or distributed routing, source-based or hop-by-hop routing, single or multiple-path routing, and static or dynamic routing. Different combinations make different trade-offs in their complexity and use of resources.

The two fundamental ways to route packets in the Internet are to use distance-vector and link-state routing. Distance-vector routing is easy to implement, but suffers from problems such as counting to infinity. We can overcome these problems using techniques such as path-vector, source-tracing, and diffusion-update algorithms. Link-state routing allows each router to get its own copy of the global topology. We have to be careful in disseminating topology to avoid corruption of the individual copies of the topology. This is done with error detection techniques, as well as the lollipop sequence space and aging link-state packets to remove stale information. Once we have the topology, we can compute shortest paths using Dijkstra's algorithm.

Both link-state and distance-vector routing have their pros and cons, and neither seems uniformly superior. They are both common in the Internet.

Choosing the cost of a link is a fundamentally hard problem. The cost influences the shortest routes in the network, and these, in turn, affect the load on a link, and hence its cost (this is similar to the Erlang map in telephone networks). The

network response map and the link metric map allow us to find cost metrics that guarantee convergence of routes and link costs.

In a large network, a router cannot store information about every other router. Instead, we divide the network into a hierarchy of levels, and each router knows only about other routers in its own level of the hierarchy. This reduces routing table sizes, though at the expense of suboptimal routing. Border routers participate in routing in more than one level, mediating exchange of information across levels to minimize the effects of hierarchical routing.

Many of the techniques used for point-to-point wide-area networks are not directly applicable to broadcast LANs, where broadcast and multicast are cheap. The Internet uses a set of special protocols in the local area to efficiently exploit these properties. These include router discovery and path redirection.

Routing is a rich field for study, and we have only touched on some essentials.

Bloc 4

Contrôle de transmission

1.0 Error Control

The end-to-end transfer of data from a transmitting application to a receiving application involves many steps, each subject to error. With adequate error control, we can be confident that the transmitted and received data are identical, although the communication occurred over a series of error-prone routers and links.

Errors can occur both at the bit and at the packet level. At the bit level, the most common error is the inversion of a 0 bit to a 1, or a 1 bit to a 0. We call this bit corruption. At the packet level, we see errors such as packet loss, duplication, or reordering. Error control is the process of detecting and correcting both bit and packet errors.

Bit-level error control usually involves adding redundancy to the transmitted data so that the receiver can detect bit errors. In some schemes, there is sufficient information for the receiver not only to detect errors, but also to correct most of them. At the packet level, we assume that bit-level error control can detect all bit errors. (Detectable but uncorrectable bit errors are treated as a packet loss.) Packet-level error control mechanisms detect and correct packet-level errors such as loss, duplication, and reordering.

We typically implement bit-level error control at the datalink layer of the protocol stack, and packet-level error control is typically found at the transport layer. Thus, bit-level error control is usually hop-by-hop, whereas packet-level error control is usually end-to-end. Generally speaking, we prefer hop-by-hop error control on links where the error rate is high (so-called lossy links) and end-to-end error control when entire path is more or less error free. However, there are many exceptions to this general rule.

1.1 Bit-error detection and correction

The basic idea in error coding is to add redundancy to the transmitted information to allow a receiver to detect or correct errors. The literature concerning error coding and coding theory is vast. In this book, we can only touch on some essentials and present an overview of the results.

There are two common kinds of error coding schemes - block codes and convolutional codes. In a block code, each block of k bits of data is encoded into n bits of information, so that the code contains $n - k$ bits of redundancy. We call the k bits the *data* bits, the encoded n bits the *codeword*, and the code an (n, k) code. For example, if we add 3 bits of redundancy to 5 bits of data, we obtain an 8-bit codeword using an (8, 5) code. In this example, we can identify sets of 8 bits in the coded stream that each correspond to a set of 5 data bits. Unlike a block code, every coded bit in a convolutional code depends on a different set of data bits. Thus, the coded stream does not contain blocks of bits in direct correspondence to a block of data bits. In this section, we will study very briefly some block codes, moving from simple schemes to sophisticated ones.

1.1.1 Parity

A parity code is a $(k + 1, k)$ block code where 1 bit is added to each block of k data bits to make the total number of l's in the $k + 1$-bit codeword even (or odd). The receiver counts the number of 1's in the codeword and checks if it is even (or odd). If the check fails, the codeword is in error.

Example 1.1

Compute the even and odd parity codes for the string "101101."

Solution: The string contains four 1's, which is an even number of l's. To obtain the even-parity codeword, we add a 0 to the end, so that the total number of l's in the codeword is even. This gives us the even-parity codeword as "1011010". For the odd-parity codeword, we must make the number of 1's in the codeword odd, so the codeword is "1011011".

Parity can detect only odd numbers of bit errors. If a codeword has an even number of bit errors, the number of 1's remains even (or odd), so a parity check incorrectly declares the codeword to be valid. For example, if the original codeword with even parity is "1011010", the same codeword with the first two bits in error, that is, "111010", also has even parity. A second problem with parity is that if the parity check on a codeword fails, there is no indication of which

bits are in error. Thus, we can use parity for error detection, but not for error correction.

The main advantage of using parity is that we can compute the parity of a string on-the-fly, with no additional storage and no added delay. However, because of its deficiencies, parity is used only when extremely few errors are expected, and a parity failure suggests a serious problem in the system.

1.1.2 Rectangular codes

In a rectangular code, we arrange data bits into an array and compute parity along each row and column. Thus, a single error shows up as a parity failure in one row and one column, allowing us to both detect and correct the error. This is an example of an *errorcorrecting* code.

Example 1.2

Consider the string of data bits "101100 011100 001001 101000 001101 010100", which we arrange in an array with six rows and six columns, as shown below:

1 0 1 1 0 0 ***1***

0 1 1 1 0 0 ***1***

0 **0** 1 0 0 1 ***0***

1 0 1 0 0 0 ***0***

0 0 1 1 0 1 ***1***

0 1 0 1 0 0 ***0***

0 0 1 0 0 0 1

The values in italics are even parities over the rows and the columns. Each row of 7 bits constitutes a codeword, and all rows but the last contain both data bits and redundant information. If the second bit in the third row is corrupted from a 0 to a 1, then parity checks on both the second column and third row fail, pinpointing the location of the error.

If two errors occur in the same row or column, a rectangular code can only detect them, not correct them. For example, if 2 bits in row 1 are corrupted, then there are parity errors in the corresponding columns, so these errors are detected. However, since the parity check of every row still succeeds, we cannot isolate and correct these errors.

The advantage of a rectangular code is that it is easy to compute, and it can correct a single-bit error. However, before we can compute a rectangular code, we must accumulate at least one row of bits in memory. This introduces coding delay.

1.1.3 Hamming codes

We call the combination of a user's data bits and redundant information a *valid codeword.* An *errored codeword is* a valid codeword with one or more corrupted bits. A key idea in error coding is that we cannot detect an error if it corrupts a valid codeword so that the errored codeword is identical to another valid codeword. Thus, valid codewords must be "different" enough that errored codewords derived from them do not resemble other valid codewords. We quantify this intuition using the concept of *Hamming distance.*

The Hamming distance between two codewords is the minimum number of bit inversions required to transform one codeword into another. For example, the Hamming distance between "101101" and "011101" is 2. If all valid codewords are at least a Hamming distance *h* apart, at least *h* bit corruptions must occur before one valid codeword is transformed to another. Thus, with fewer corruptions, the resulting codeword is distinguishably an errored codeword, which means we can detect up to *h* -1 errors.

Example 1.3

Consider an error code where we represent 1 as 111, and 0 as 000. The only two valid codewords are 000 and 11l, which are a Hamming distance 3 apart. A single-bit error in a valid codeword, say, 000, can result in errored codewords 100, 010, or 001. None of these is a valid codeword, so we know that an error has occurred. Moreover, all three errored codewords are a Hamming distance 1 from 000, and a Hamming distance 2 from 111. Thus, we can interpret them as

corrupted versions of the codeword 000. In other words, a receiver receiving an errored codeword 010 can assume that the actual data bit was a 0, which automatically corrects the error.

If the valid codeword 000 is corrupted with two bit errors, the errored codewords possible are 110, 011, and 101. Since none of these is a valid codeword, we can detect two errors. However, these errored codewords are closer to the valid codeword 111 than to the valid codeword 000. Thus, if we try to correct the two bit errors, we will be mistaken. We should use this code for error correction only when the chance of two bit errors in the same codeword is smaller than our error threshold.

In general, if we want to *detect* up to E errors, then all valid codewords should be at least $E + 1$ distance apart from each other. If we want to *correct* up to E errors, the minimum distance should be at least $2E + 1$.

Besides coming up with the idea of a Hamming distance, Hamming studied the design of *perfect* parity codes. We omit the details about these codes. We just mention that Hamming codes, like rectangular codes, are error correcting. But, they do not require storage. Moreover, they can be pre-computed and stored in a table for quick lookup. Thus, they are suitable for simple hardware and software implementation. For these reasons, a Hamming perfect parity code is widely used in the field.

1.1.4 Interleaved codes

The coding techniques described so far are best suited to random, non-bursty bit errors. Error bursts introduce multiple bit errors within a codeword, which cannot be detected or corrected using parity, rectangular, or Hamming codes. The standard way to solve this problem is to use *interleaving* to convert burst errors to bit errors. In this technique, m consecutive codewords are written in a $n \times m$ matrix, then transmitted column-wise instead of row-wise. Thus, a burst error of up to m bits appears as a single-bit error in each of the m codewords. These single-bit errors can then be corrected with Hamming or other parity codes. Interleaved codes require buffering the input, and so add memory cost and delay.

1.1.5 Cyclic redundancy check

A *cyclic redundancy check* (CRC) *is* one of the most popular techniques for error detection. In this technique, we treat the entire string of data bits as a single number. We divide this number by a predefined constant, called the *generator* of the code, and append the remainder to the data string. The receiver performs the same division and compares the remainder with what was transmitted. If the data string was received without errors, then the remainders match. For example, suppose the string to be transmitted is "110011", which corresponds to decimal 51. If the generator is "110" (decimal 6), the remainder is decimal 3, or binary "0ll". We append "011" to the data string and send it to the receiver. If the receiver received everything correctly, it should come up with the same remainder. Intuitively, it is unlikely that a corruption in the data string will result in a number that has the same remainder as the original data. In this example, for the CRC to fail, the received string should also have a remainder of 3 when divided by 6, which is unlikely. This is the basic idea behind CRC.

We will not give a detailed explanation of CRC coding, but we will merely state some results important to the engineer. Before we do so, we will need some notation.

We represent a block of $(k + 1)$ bits by a polynomial of degree k in the dummy variable x, written as $a_k x^k + \ldots + a_1 x^1 + a_0 x^0$, where a_k is 0 if the bit in that position is 0 and 1 otherwise. For example, the string "10011" is represented by the polynomial $x^4 + x + 1$.

The effectiveness of a CRC code depends on the choice of the generator G, which, when written in polynomial form, is called the *generator polynomial G(x).* It can be shown that a CRC detects the following:

- *All* single-bit errors
- Almost all double-bit errors, if *G(x)* has a factor with at least three terms
- Any odd number of errors, if *G(x)* has a factor x + 1
- *All* bursts of up to *m* errors, if *G(x) is* of degree *m*
- Longer burst errors with probability $1 - 2^{-m}$, if bursts are randomly distributed

Thus, with a good choice of the generator polynomial, CRCs are a powerful mechanism for error detection (and rarely correction). Standard generator polynomials are prescribed by international standards bodies, depending on the expected error pattern.

CRC codes are popular because they can be efficiently implemented in hardware or software. A hardware implementation requires only a shift register and some XOR gates, uses no additional storage, and can compute the CRC on-the-fly. An efficient software algorithm is to pre-compute the remainder for each possible data string of a certain length (say, 16 bits) and store the result in a lookup table. The algorithm looks up the remainder for each 16-bit chunk of the input and adds these, modulus 2, to a running sum. It can be shown that this is equivalent to computing the remainder over the entire string. Thus, we can compute the CRC for an arbitrarily long input string on-the-fly in software, with only one lookup per block of input bits.

1.1.6 BCH and Reed-Solomon codes

Bose-Chaudhuri-Hocquenghem (BCH) codes are CRC-like codes constructed over *blocks* of *m* bits, instead of over single bits. in other words, the alphabet of a BCH coder is not {0,1} as in CRC codes, but a set of 2^m symbols, where each symbol is a distinct *m*-bit binary string. BCH codes are robust to burst errors, since a burst of up to *m* errors results in at most two errors in the BCH alphabet (for errors that span two BCH symbols). BCH codes are also some of the best-known codes for correcting random errors.

Reed-Solomon codes are a special case of BCH codes where the block size (that is, the number of data symbols + the number of redundant symbols) is 2^m, which is also the largest possible block size for a BCH code. It can be shown that a Reed-Solomon code that has $2t$ redundant bits can correct any combination of *t* or fewer errors. Reed-Solomon codes are one of the best block codes for dealing with multiple bursts of errors in a codeword - one variant can correct errors up to 1200 bytes long.

1.1.7 Software coders

Most of the schemes we have studied thus far are meant to be implemented in hardware. Because software speeds are much slower than those of hardware, an efficient error-detection scheme implemented in software must use only the simplest of operations and must minimize the number of times it touches data. Moreover, in many instances, we may be willing to tolerate a higher degree of errors for speed of operation in the common (error-free) case. Thus, schemes such as rectangular coding or convolutional coding, which touch each data byte several times, are unsuitable for software implementation. We have already mentioned that CRCs can, however, be implemented relatively efficiently in software.

A common software error detection scheme, used extensively in the Internet protocol suite (for example, in IP, UDP, and TCP) is the *16-bit one's-complement* checksum. Here, the data string is treated as a series of 16-bit words. The algorithm adds the words together, with end-around carry (that is, if an addition causes an overflow, we add 1 to the sum). The checksum is the one's complement of the sum. The receiver carries out the same computation, over both the data and the checksum. If the final sum is 0, then the data passes the checksum.

The 16-bit one's-complement checksum can catch all 1-bit errors in the data, and, if the data values are uniformly distributed, is expected to incorrectly validate corrupted data only about one time in 65,536. It is efficient, because it touches each 16-bit word only once, and, if the machine word is a multiple of 16 bits, only one or two additions per machine word of data.

1.2 Causes of packet errors

Bit-level and packet-level error control mechanisms are fundamentally different. Bit-level error control uses redundancy to detect, and possibly correct, bit errors. Packet-level error control uses bit-level or packet-level error detection to identify and discard packets in error (including those with uncorrectable bit errors), at the receiver. The receiver sends the sender either a *positive acknowledgment* (ack) for every packet correctly received, or a *negative acknowledgment* (nack) indicating which packets it did not receive. The sender uses these indications to decide which packets were lost, and retransmits them. Thus, in packet-level error control, error correction is usually based on retransmission, rather than redundancy. In this section, we study some causes for packet errors. Section 1.3 describes techniques for packet-level error control.

We can classify packets errors as (a) packet loss, (b) duplication, (c) insertion, and (d) reordering. We study these in turn.

1.2.1 Packet loss

Packet loss due to uncorrectable bit errors is common in wireless links, where loss rates as high as 40% have been measured in the field.

In wired links, particularly fiber-optic links, undetectable bit errors are rare, and the dominant cause of errors is a temporary overload in switch and multiplexor buffers (i.e., congestion). The packet loss behavior of such a network is strongly dependent on its workload. For example, if most traffic sources are *smooth*, that is, the ratio of the standard deviation in the transmission rate to its mean is small, then overloads are uncommon and the packet loss rate is small. As the traffic becomes burstier, for the same network utilization, the loss rate increases, because transient overloads are more common. Networks that carry primarily voice and video data can expect to see fairly smooth data, so that, with reasonably small buffers, packet losses can be kept to a minimum. In data networks such as the Internet, burstiness is a fact of life, so every endpoint should expect to see, and deal with, packet loss.

We now summarize some facts about packet-loss behavior:

- For realistic workloads, packets are more likely to be lost as the network utilization increases
- If a switch or multiplexor drops a packet, it is more likely that consecutive packets arriving to the switch or multiplexor will also be dropped. Thus, there is a strong correlation between consecutive losses at a buffer. That is, packet losses are *bursty*.
- Packet losses decrease as the number of buffers available in the network increases. Under some simplifying assumptions, it can be shown that the probability of packet loss at a multiplexor decreases exponentially with the size of the buffer. Recent measurements suggest, however, that these assumptions may not hold in practice. Thus, the dependence of the packet loss rate on buffer size in real networks is still a matter of debate.
- A low-bit rate (relative to the line speed), widely spaced periodic stream, such as a packetized-audio stream, sees essentially uncorrelated queue lengths and a context-independent packet loss probability at the overloaded network element. Thus, such a stream does not see bursty losses.
- When measuring packet loss on the same path over the period of a day, measurements indicate a positive correlation between packet loss and metrics of end-to-end delay (such as its mean, minimum, and maximum).

1.2.2 Loss of fragmented data

Consider a switch or multiplexor that serves data in units of fixed-size cells. We will assume, as in ATM networks, that some of these cells represent portions of packets, and that if any cell in a packet is lost, the packet is in error. During overload, the incoming data rate is higher than the service rate, and if the overload is sustained for long enough, the data buffer at the switch or multiplexor will overflow, leading to cell loss. Suppose that fifty cells are lost during a particular loss event. If all fifty cells belong to the same packet, then the cell-loss event corresponds to a single packet loss. If the cells belong to fifty different packets, then the cell-loss event corresponds to fifty lost packets. Thus, a single cell-loss event may lead to many lost frames, a form of error multiplication. The degree of error multiplication depends on the arrival pattern of cells during a loss event, which is workload dependent. If overloads are caused by bursts from a single source, there may be no error multiplication. On the other hand, if many sources contribute to the overload, each of them could see a packet loss, Note that as the packet size increases, each cell loss proportionally decreases the error-free (or effective) throughput.

Example 1.4

Assume that the cell loss rate in an ATM network is 10^{-5}, and a packet has a mean size of 100 cells. (a) What is the range of mean packet loss rates? (b) If a link serves 1000 cells/s, has a packet loss rate of 10^{-3} and each loss recovery takes 1 s, what is the effective throughput of the link?

Solution: (a) The packet loss rate can be as low as 10^{-5} (if all cells lost are from a single packet) and as high as 10^{-3} (if each cell lost is from a different packet). (b) Consider the time taken to serve x packets. First note that of these x packets, $x/1000$ will be lost, and it will take time $x/1000$ s to recover from these losses (assume that there are no losses during recovery). Since the link speed is 10 packets/s, the remaining $x - x/1000$ packets will take time $(x - x/1000) * 0.1$ s. Thus, the total time to send x packets is $x/1000 + (x - x/1000) * 0.1$ s, so that the effective throughput is 9.91 packets/s.

Reseachers studied the behavior of a TCP connection in an ATM network with cell losses. They showed that, for a

fixed buffer size, as the packet size increases, the effective throughput decreases, since each cell loss event causes a larger and larger fraction of the successful cell transfers to be wasted. This can be prevented by having switches and multiplexors drop entire packets instead of cells. A clever way to do so is to mark a virtual circuit as being in the drop state whenever the buffer capacity exceeds a threshold. The switch drops cells arriving to a connection in the drop state until it sees a cell with the end-of-packet mark. At this point, the state of the circuit may revert to normal, or may persist in the drop state. In either case, switches mostly drop entire packets, without having to look through their queues. This variant of packet dropping, called *early packet discard*, substantially improves TCP performance over that of ATM networks.

1.2.3 Packet duplication, insertion, and reordering

Retransmission is a common technique to compensate for packet loss. In this technique, the sender sends data, then waits for an *acknowledgment* (ack) of receipt. If it receives no answer for some time, it retransmits the packet. The packet, however, may have been correctly received, but its acknowledgment may have been delayed or lost. Thus, on retransmission, the receiver would receive a duplicate, out-of-sequence packet. This is the main cause for packet duplication and reordering in computer networks.

Packet reordering can also happen if packets belonging to the same stream follow different data paths from source to destination. For example, in the technique of *dispersity routing*, a source sends data to a destination along multiple paths to increase its bandwidth share in the network. Since different packets arrive at different times, the receiver sees reordered packets.

Packet insertion can happen when a packet's header is undetectably corrupted. For example, assume that two processes at a destination are expecting cells on VCIs 1 and 2. Suppose the VCI field in the header of a cell in VCI 2 is undetectably corrupted, so that the cell appears to be on VCI 1. Then, the entity waiting for a cell incorrectly receives this cell on VCI 1 at the destination, leading to a packet insertion error. Another mechanism for packet insertion is the arrival of a packet delayed beyond the close of its connection. For example, a packet on VCI 1 may be delayed in the network beyond the close of VCI 1. If a new connection at the receiver is also allocated VCI 1, it may incorrectly accept the delayed packet, leading to packet insertion.

1.3 Packet-error detection and correction

In the preceding section, we studied some causes for packet errors. In this section, we focus on mechanisms for packet-error detection and correction. We will assume that bit-error detection and correction mechanisms are simultaneously active, so that the information in a packet header used by a packet-level error-control mechanism is very unlikely to be corrupted.

We start with a description of sequence numbers, which are the first line of defense against packet loss, reordering, and insertion.

1.3.1 Sequence numbers

A sequence number in a packet's header indicates its unique position in the sequence of packets transmitted by a source. A sender labels each packet that has not been previously transmitted with a consecutive sequence number in its header. Sequence numbers help the receiver to detect packet loss, reordering, insertion, and duplication.

- Packet reordering and duplication are immediately obvious with sequence numbers.
- Packet loss shows up as a *gap* in the sequence numbers seen by the receiver. For example, assume a transmitter sends packets with sequence numbers 0, 1, 2, ... If the packet with sequence number 4 is lost, the receiver receives packets with sequence numbers 0, 1, 2, 3, 5, 6, etc., with a gap between sequence numbers 3 and 5.
- We can detect packet insertion if the sequence number of the misdelivered packet is very different from the sequence numbers of packets already on that connection. For example, if the last packet received on a connection has a sequence number 5, and the misdelivered packet has a sequence number of 41,232, the receiver can guess that the packet was misdelivered.

We will often use the abstraction of a *sequence number space*, which is the subset of the natural numbers that represents all possible sequence numbers in a packet header. If a sequence number *is n* bits long, the space is of size 2n. When a sequence number reaches the largest number in this space, it *wraps around* to the smallest number in the space, typically 0. For example, if a sequence number is 3 bits long, the sequence number space is 10, 1, 2, 3, 4, 5, 6, 7). When the sequence number reaches 7, the next sequence number wraps around to 0.

In the next two subsections, we will study two important considerations when using sequence numbers: choosing a sequence number length, and choosing an initial sequence number.

1.3.2 Sequence number size

Because sequence numbers take up header space, and every packet must have a header, it is important to minimize the number of bits devoted to a sequence number. How long should a sequence number be? The intuition here is that the sequence number should be long enough so that, taking sequence number wraparound into account, a sender can disambiguate acks even in the worst case.

We define the following terms and symbols:

MPL or *maximum packet lifetime:* The maximum time a packet can exist in the network (also called, in TCP, *maximum segment lifetime,* or *MSL)* (in seconds)

T: The maximum time a sender waiting for an ack persists in retransmitting a packet (in seconds)

A: The maximum time a receiver can hold a packet before sending an ack (in seconds)

R: The maximum transmission rate of the sender (in packets/second)

Consider a packet with sequence number, say, 4, that is repeatedly lost by the network. The sender keeps retransmitting the packet until a time *T* after it first transmitted the packet. In the worst case, the receiver successfully receives the packet the final time it is retransmitted, at time $T + MPL$. The receiver may now dally for a time *A* before sending an ack, which takes at most an additional MPL time to reach the sender (see B4T14). The sender may have generated as many as $(2\ MPL + T + A)\ R$ more sequence numbers in that time, so if the sequence number space wraps around, another packet with sequence number 4 may be transmitted before the sender received the earlier ack. Thus, the arrival of an ack for packet 4 at the sender would be ambiguous. To remove this ambiguity, we must ensure that the sequence number is at least *n* bits long, where $2^n \geq (2\ MPL + T + A)\ R$. This is the desired lower bound on the sequence number size.

Example 1.5

Assume that the *MPL* is 2 min, T = 1 min, A = 500 ms, and the source transmits no faster than *2* Mbps. What is the smallest possible sequence number size if the packet size is at least 40 bytes?

Solution: The largest possible number of packets that could be generated while some packet is still active is given by the expression $(2MPL + T + A)R$. We have:

$MPL = 2$ min = 120 s

T = 1 min = 60 s

A = 500 ms = 0.5 s

$R = 2$ Mbps = 2^6 /8 bytes/s = 2^6/ (40 * 8) packets/s

Plugging this into the right hand side of the expression, we get its value to be (2 * 120 + 60 + 0.5) 2^6 /(40 * 8) = 1,878,125. This corresponds to a minimum sequence number size of 21 bits, since 2^{20} = 1,048,576, and 2^{21} = 2,097,152.

Note that the lower bound on sequence number size requires a bound on *MPL*. Theoretically, we can ensure that a packet does not live in the network longer than a given *MPL* by placing a generation time in its header. Network elements can discard packets with a time stamp more than *MPL* old. This technique requires additional space in the packet header and additional computation at each hop, and so is not used in-practice. Instead, the header typically contains a counter decremented by each intermediate network element. If a network element decrements the counter to zero, it discards the packet. Although this technique does not directly give us a bound on *MPL*, if we can estimate the worst-case delay at each network element, knowing the largest possible value of the counter, we can roughly bound *MPL*. This is sufficient for most engineering purposes. This technique is used in IP, in the Time-to-Live field (TTL).

We can make the sequence number space much smaller if a source does not retransmit packets, because it only needs to be large enough to detect losses and reordering. Two factors, then, determine the minimum size of the sequence space. First, a single burst loss should not lead to a seamless wraparound in the sequence space; otherwise, a receiver would be unable to detect this burst loss. For example, if the sequence number is 3 bits long, the sequence space is of size 8, and a

loss of 8 packets causes a wraparound. If a receiver received packet 2, did not receive packets with sequence numbers 3, 4, 5, 6, 7, 0, 1, 2, and then received packet 3, it would be unable to detect the packet loss. Thus, if we expect the longest possible burst length to be 8 packets, the sequence number should be at least 4 bits long. In general, if the largest possible burst loss is expected to be n packets, then the sequence number must be longer than $\log(n)$ bits. Second, a receiver must not mistake a reordered packet for a valid packet. If the largest possible reordering span is of length n, then a similar argument shows that the sequence number should be longer than $\log(n)$ bits. In practice, we choose the sequence number to be the larger of the two lower bounds. This, however, is usually much smaller than the sequence number required when retransmissions are possible.

In real life, a system designer may not be able to tightly bound the longest possible error burst or reordering span. The only recourse, then, is to design for the worst-case scenario, add a fudge factor, and hope for the best. With any luck, the network design will be obsolete before the worst-case scenario actually happens!

Sequence numbers can be used for error control at both the datalink and the transport layer. Datalink layers usually do not retransmit data, whereas transport layers usually do. Thus, we can use a smaller sequence number at the datalink layer than at the transport layer.

1.3.3 Dealing with packet insertion

Packet-switched networks may delay a packet for up to one *MPL* inside the network. We should somehow identify and reject a delayed packet arriving at a connection; otherwise, we will have a packet insertion error. One can completely avoid the problem by flushing every packet belonging to a connection on connection termination. This is easy if every layer in the network is connection oriented, so that the connection-teardown message propagates to every switch controller along the path. On receiving this message, each controller instructs its local switch to flush all packets belonging to that connection.

For connections layered over a connectionless network (such as TCP connections on top of an IP connectionless network layer), however, there is no explicit connection termination message. Thus, we have to use more sophisticated schemes to solve the problem. We will study some of them next.

Before we begin, we need to know how connections are identified when a connection is established over a connectionless network. In most connectionless networks, packets exchanged between two connected processes carry a pair of numbers, called *port* numbers, identifying the connection endpoints (this is the case when TCP runs over IP). For example, process A on a machine with address 129.x.y.z may be associated with port number 2000, and process B on a machine with address 192.u.v.w may be associated with port number 2004. Then, a packet from the first process to the second carries the source address (129.x.y.z), the destination address (192.u.v.w), the source port number (2000), and the destination port number (2004). These four numbers uniquely identify a connection.

The insertion problem arises when a delayed packet from an old connection has the *same* set of four connection identifiers as a newer connection, whose sequence number is in the range used by the newer connection. Continuing with our example, suppose packets with sequence number 1, 2, and 3 are sent from process A on machine 129.x.y.z to process B on machine 192.u.v.w. Assume that packets 2 and 3 are delayed in the network for one *MPL*, and, meanwhile, process B closes the connection. If (a) the same or other processes on these machines open a new connection, and (b) the machines reassign these processes the same port numbers, and (c) the receiving process in the new connection receives delayed packets 2 and 3 just after packet 1 of the new connection, then it may accept the delayed packets as part of the new connection.

One way to avoid the problem is to label each packet with an incarnation number that is unique for each connection between a pair of endpoints. Thus, packets from different connections can be distinguished by their incarnation numbers. This has the problem that each packet needs an additional field in the header, which reduces the bandwidth efficiency. Moreover, incarnation numbers need to be remembered across system crashes. Otherwise, machine 129.x.y.z may crash and, on rebooting, reuse an incarnation number. This requires memory that can survive crashes. Such memory, called stable storage, is not generally available, and it tends to be expensive. This scheme, therefore, is not popular in practice.

A second solution is to reassign a port number only after one *MPL*. Then, by the time a process can reuse a port number, all old packets would have been discarded. This requires an endpoint to remember the time at which it assigned every port number. If the set of possible port numbers is large, this takes storage, which can be expensive. Moreover, an argument similar to the one in the previous paragraph shows that the system must remember these times across crashes, which again requires expensive stable storage.

Unix-like systems, therefore, use a third scheme, where an end-system assigns port numbers to processes serially, starting from port number 1024. This makes it unlikely, but not impossible, that a newly opened connection will have the same port number as a recently closed one. One way the scheme can fail is when a process opens and closes

connections so rapidly that the port numbers wrap around back to 1024 before the expiration of one MPL. To solve this problem, we ensure that successive connections initiated from the same endpoint use different initial sequence numbers by choosing the initial sequence number from a register that is incremented every clock tick. This guarantees that even if port numbers wrap around, connections that share the same port number choose different initial sequence numbers. Since the new connection's initial sequence number would be much larger than that of the delayed packet, this prevents packet insertion. The modified scheme, however, is still not foolproof! If the computer running process A crashes and reboots, it might reassign port 1024 to another process before the expiration of one *MPL*. Moreover, since the initial sequence number register is reset on a crash, the sequence numbers might still match.

There are several ways to avoid this problem. First, on a reboot, a machine could remember the port numbers and sequence numbers it used earlier. This again requires stable storage, which rules it out as a general solution. Second, we could prevent the clock from resetting on startup. This requires a battery backup for the clock, and we again rule it out for the same reasons. Third, and this is the solution used in practice, a rebooted computer can keep silent for one *MPL*. This flushes all the packets from the earlier connection from the network. So, even if a packet from a new connection has the same port number and the same sequence number as a potentially delayed packet, by the time the new connection is allowed to start, the delayed packet would have been discarded. This finally solves the problem.

Internet hosts usually use an *MPL* of 30 s to 2 min. Since a crashed computer takes around 2 min to reboot anyway, we expect all delayed packets to have been flushed by the time the computer restarts. If a computer reboots faster than one *MPL*, it should remain silent for the rest of the interval (though this is rarely done in practice!).

To sum up, the solution requires us (a) to assign ports serially, (b) to choose an initial sequence number from a clock, and (c) to ensure that a system remains silent for at least one MPL after a crash. When these three conditions are fulfilled, we can guarantee that no packet will be inserted into a conversation (unless, of course, the header is undetectably corrupted). This solution is mandated for all Internet-connected hosts.

1.3.4 Three-way handshake

After choosing an initial sequence number (ISN), as described in Section 1.3.3, a sender must inform its peer about the ISN it plans to use for the connection. The receiver stores this value in its expected sequence number (ESN) for that connection. If the connection is bidirectional (as in TCP), the receiver will, in turn, tell the sender the ISN it plans to use for the reverse connection. The ESN is necessary to distinguish packets from expired connections from packets from active connections, as discussed in Section 1.3.3.

A naive way to exchange ISNs is for the connection initiator to start the connection with a SYNchronization (SYN) packet containing its ISN. The connection acceptor replies with a SYN containing its choice of ISN. The ISNs are then stored in the ESN state of the connection initiator and connection acceptor. Note that the SYN packet itself is not protected against delayed packets because, when a SYN is received, the recipient does not know the expected sequence number. Thus, a delayed SYN can confuse the acceptor, as explained next.

Suppose processes A and B are trying to establish a full-duplex connection, and B is the connection acceptor in the "wait-for-SYN" state. Suppose a delayed SYN from A arrives at B. Since B has no way to tell that this is a delayed SYN, it accepts the ISN as valid. B then replies with its own SYN containing its choice of ISN. If, meanwhile, A has already launched its SYN, it might think that the SYN from B was in response to its latest SYN, and consider the connection open. However, B will reject A's second SYN as a duplicate and will choose an ESN from the earlier SYN. Thus, every subsequent packet from A has an incorrect sequence number and is rejected.

We can solve this problem by adding an extra step to the connection setup (see figure B4T22). In the first step, A sends B a SYN with its ISN, as before. In the second step, B picks an ISN and sends a SYN-ACK packet that contains B's ISN and an ack of A's ISN. In the third step, A acknowledges B's choice of ISN with an ACK packet carrying B's ISN.

When the B receives this packet, both A and B know each other's correct ISN. It can be shown that a three-way handshake is the minimum required to achieve this happy situation.

Note that if a delayed SYN arrives at B, A gets a SYN-ACK that contains the old ISN. It can inform B to discard this connection with a RESET packet. In the same way, if A replies to B with an incorrect ACK packet, B can send A a RESET packet. On receiving a RESET packet, an endpoint replies with a RESET-ACK and drops the connection.

1.3.5 Dealing with system crashes

If a system crashes while it has open connections, then when it restarts, the remote end must not confuse packets from reestablished connections with packets from old connections. A connection where one endpoint has crashed, has closed

the connection, or is otherwise unaware of the connection is called a *half-open* connection.

We use four techniques to deal with half-open connections. First, on booting up, an endpoint waits for all of its previously unacknowledged packets to die away, as described in Section 1.3.3. Second, also on booting up, an endpoint rejects all acks that arrive during a "quiet time". In the worst case, a sender may have sent a packet just before the endpoint crashed, which would take *MPL* to reach the receiver. The ack may return as late as $2*MPL + A$ seconds later. By discarding all incoming packets for $2*MPL + A$, the endpoint can protect itself from acknowledgments arriving to half-open connections. Third, an endpoint times out a connection if there has been no activity for some time, say, $2*MPL$. Thus, if an endpoint reboots, all endpoints that are its connection peers automatically abort the connections. (If an endpoint wants to maintain a connection that would otherwise be timed out, it should periodically send a dummy "keep-alive" packet.) Fourth, and finally, endpoints actively detect and abort half-open connections. If an endpoint receives an unsolicited packet, that is, a packet on an unestablished connection, from a system that erroneously thinks a connection is still up, it should send back a RESET packet to abort the connection. This interlocked set of mechanisms allows us to provide error-free packet transmission despite system crashes and half-open connections.

1.3.6 Loss detection

Thus far, we have assumed that a lost packet is retransmitted by its sender. There are two ways in which a sender can find out that a loss has occurred. First, it can detect a loss with a *timeout,* and second, it can be told about the loss by the receiver using a *negative acknowledgment.* We study these two schemes next.

Timeouts

A sender can use timeouts to detect losses by setting a timer every time it sends a packet. If it does not receive an ack before the timer expires (i.e., times out), it guesses that the packet or its ack was lost and retransmits the packet. The tricky part is in choosing a timeout interval wisely, because a poor choice can degrade performance. If the timer interval is too small, variations in packet service time in the network trigger spurious timeouts, and if the timer interval is too large, a loss is corrected only after a long, annoying pause. We now study some techniques for choosing timer values.

The simplest way to set a timer is to choose a timer interval a priori. This works well if the system is well understood and the variation in packet-service times is small. Fixed timeouts are common in simple, special-purpose systems where these assurances can be made.

A better way to choose a timeout is to base it on past measurements of the roundtrip time (RTT), which is the measured time interval between sending a packet and receiving an ack for it. If the receiver acknowledges every packet, and the sender does not receive an ack one RTT after it sent the packet, it can guess that the packet was lost. Thus, the timeout value should be on the order of one RTT. If the system has no variability, the timeout value can be exactly one RTT However, most practical systems use a value somewhat larger than one RTT (usually twice the measured RTT) to deal with variations in packet-processing time and queuing delay.

As a sender sends a series of packets, it receives a series of acks, and thus accumulates a series of RTT measurements. Instead of using the last measured RTT to set the timer, it makes sense to smooth the series to eliminate random fluctuations (noise). A common technique for doing so is based on the *exponential averaging* filter, which we describe next.

If $rtt(k)$ represents the measured value of the RTT using the k^{th} packet, and a is a tuning parameter in the range [0,1], then the output of the filter is a smooth RTT estimate, $srtt(k)$, which is given by:

$$srtt(k) = a.srtt(k\text{-}1)+ (1 - a)\ rtt(k)$$

In other words, the estimate adds a fraction of the new RTT to itself, retaining a fraction a of past history. The closer a is to 1.0, the larger the weight placed on past history, with a correspondingly smaller dependence on recent measurements. Thus, if the RTTs vary quickly, choosing a small a allows the estimate to track the input quickly. Conversely, if RTTs vary slowly, choosing a large a allows the estimate to ignore most of the noise. Given this estimate, the timeout is chosen to be $bsrtt(k)$, where b is a constant that reflects the variability in packet-processing time and queuing delay in the system. Typical values for b are 1.5 to 2, and for a are 0.6 to 0.9. (TCP recommends $a = 0.9$ and $b = 2$).

An exponential averager is sensitive to the initial value if a is close to 1 and tracks the input closely if a is close to 0. Spikes in the input are smoothed out better if a is close to 1. Thus, the choice of a is critical. Static choices of a tend not to work well if the system dynamics change over time. Ways to adapt a dynamically as a function of the workload were proposed in the literature.

A better estimate for the timeout is based on the mean deviation in the RTT. In this technique, we maintain an additional error term $e(k)$ and its smoothed estimate $m(k)$. These are computed as follows:

$$srtt(k) = asrtt(k\text{-}1)+ (1 - a)\ rtt(k)$$

$$m(k) = |srtt(k) - rtt(k)|$$

$$sm(k)= (1 - a)m(k) + a.sm(k\text{-}1)$$

$$timeout(k) = srtt(k) + b.sm(k)$$

The term *sm* measures the mean deviation from the mean and is an estimate of the standard deviation in *rtt*. If the distribution of the RTTs is approximately Gaussian, we expect a packet to be processed with time *srtt* + *bsm*, where different values of *b* give different confidence intervals. So, for example, if $b = 5$, we expect fewer than 1% of packets to be delayed beyond this timeout value.

Problems with choosing timer values

The previous paragraphs have presented some clever ways to choose timeout based on measuring RTTs. However, even the best timeout estimates have some intrinsic problems. For example, consider timeout values based on an exponential average of the round-trip times. First, it is not clear what the initial value for the timeout estimate *srtt* should be. This is particularly hard to deal with when *a* is large, so that the system converges slowly from the initial value of the estimate to the true RTT. Second, measuring the RTT is difficult in the presence of packet losses. When losses are few, every packet is acked, and a sender can correctly measure RTTs. However, when losses are bursty, as during congestion episodes, the variability in the RTTs is large. Moreover, if an ack acknowledges more than one packet, determining the packet from which the RTT should be measured is difficult. (One way to work around this, called *Karn's algorithm*, is to ignore *all* RTT measurements from retransmitted packets). Finally, even if we correctly determine the timeout, it is still not a good indicator of the system state. If a packet S is timed out, one of several things may have happened: S may be awaiting transmission at some link. It may have been correctly received, but its ack may have been lost. Or, finally, the network may actually have dropped it. The sender cannot choose among these situations solely from the timeout. Thus, timeouts should be used rarely, and if used, must be augmented with other information to disambiguate between possible system states.

Negative acknowledgments

One way to avoid timeouts entirely is for a sender to be informed by the receiver that a packet has been lost by means of a negative acknowledgment or *nack. A* receiver sends a nack when it detects a gap in the sequence of packets it receives. *A* nack contains a range of sequence numbers of packets that have been lost and must be retransmitted. On receiving a nack, the sender retransmits these packets.

The problem with nacks is that they are generated during loss events, which are typically caused by buffer overflow during congestion. In other words, they add to the network load precisely when it is already overloaded. Of course, nacks and packets travel in opposite directions, so packet congestion does not necessarily indicate that the reverse path is also congested. Nevertheless, the network is likely to be overloaded, and it seems somewhat risky to add to the load. *A* second problem with nacks is that if the nacks themselves are lost, then the receiver must retransmit them. This moves the timeout problem from the sender to the receiver, but does not solve the problem of determining what timeout value should be used. For these two reasons, many recent protocols prefer timeouts to nacks.

1.3.7 Retransmissions

Timeouts suggest that a packet was lost-the retransmission strategy decides *which* packets to retransmit on a loss. Moreover, some retransmission, strategies can detect packet loss without a timeout. Two retransmission strategies are in common use: *go-back-n* and *selective retransmission.*

Go-back-n

Before studying go-back-n, we make a small diversion to understand the notion of an error-control window, which is the smallest contiguous subset of the sequence space that contains the sequence numbers of all the packets transmitted but not acknowledged. In other words, a sender does not know whether packets in its error-control window have been successfully received or have been lost. For example, if packets with sequence numbers 5-10 have been transmitted, and only sequence numbers 7 and 8 are known to be correctly received, the error-control window is of size 6 and spans the range 5-10.

It is often useful to limit the size of the error-control window, for reasons we will discuss shortly. For the moment, assume that we fix, in advance, the largest allowed size of the error-control window. When the sender sends a packet, the window expands by 1. Thus, the limit on the size of the error-control window eventually forces the source to stop. Each ack shrinks the window by 1, allowing a blocked sender to send another packet. Note that sending a packet and receiving an ack both shift the window one step forward in sequence space (B4T31). Thus, we call this mechanism *sliding-window* error control.

Returning to go-back-n, in this strategy, a timeout (or nack) causes the entire error-control window to be retransmitted. For example, a sender that has sequence numbers 5-10 in its error-control window would retransmit all six packets on a timeout. We can now see the reason to limit the size of the error-control window with go-back-n. The larger the error-control window, the more the sender has to retransmit on a timeout or nack. Thus, in go-back-n, we prefer to use the smallest feasible error-control window.

Besides its simplicity, go-back-n has several other advantages. First, it is conservative, because a single loss triggers a retransmission of every possible lost packet. Thus, it is suitable when losses are bursty. Second, the receiver's algorithm is also very simple: it accepts a packet if it is in sequence, and rejects it otherwise. Third, a receiver does not require any

storage for out-of-order packets. However, go-back-n wastes bandwidth, because more packets are retransmitted than are strictly necessary. Moreover, if the original loss was due to buffer overload, the burst of packets triggered by go-back-n is likely to contribute to the overload, making it worse. Unless checked by other forces, this can lead to a state where the network carries only retransmissions, and thus makes no net progress! We call this *congestion collapse*.

Because of its waste of bandwidth, a sender using go-back-n can never fully use the link's capacity, unless there are no losses. We can compute the link utilization as a function of the packet loss probability and the error-control window size. Let the packet loss probability be p, and the window size be w. Then, with go-back-n, the maximum link efficiency (the percentage of the link bandwidth used by non-retransmitted packets) is:

$$(1-p)/(1 - p + p.w)$$

Note that, keeping the end-to-end propagation delay constant, *w* increases with link speed (because more packets will be outstanding when more packets can be transmitted in a given amount of time). Thus, the link efficiency decreases with link speed for a given probability of loss. For reasonable window sizes, the efficiency degrades rapidly for error rates larger than about 1 in 10,000.

Example 1.12

Compute the maximum link efficiency with go-back-n when p = 0.01 and *w* = 250 packets. Recompute this when $p = 10^{-5}$.

Solution: For $p = 0.01$, the efficiency is bounded by (1 - 0.01) / (1 - 0.01 + 0.01 * 250) = 0.283. For $p = 10^{-5}$, the efficiency bound improves to 0.997.

Selective retransmission

Selective retransmission is a more intelligent strategy than go-back-n. Here the sender uses additional information to decide which portion of its window is likely to have been lost. This information can come in many ways:

- Each ack from the receiver can carry a bit map of the current error-control window, marking the packets received. Thus, on a timeout, the sender need only retransmit packets that the receiver has not acknowledged. Continuing with our example, if the window is 6 packets, and packets 7 and 8 have been received, the ack for packet 8 might carry the bitmap 001100. On a timeout, if no further information is received, the sender can assume that packets 5, 6, 9, and 10 are lost and can retransmit them. This scheme requires the overhead of a bit map for every ack.

- To avoid the bit-map overhead, the receiver might periodically send a STATE packet with a bit map containing sequence numbers of all the packets that it has received in the recent past. The sender can use this information in much the same way as in the previous paragraph.

- The sender might do *fast retransmission*. Assume that every ack carries the sequence number of the last in-sequence packet seen by the receiver (also called the *cumulative acknowledgment).* If the sender sees the ack sequence number repeated, this indicates either a packet loss or a reordering. For example, if the receiver received packets up to 5 in sequence, but then got packets 7, 8, and 9, then its acks would repeat the cumulative acknowledgment for 5, since this was the last packet received in proper sequence. When the sender receives the ack due to packet 7 with a repeated cumulative acknowledgment, it guesses that 6 was not received, either because it was lost, or because 6 was reordered, perhaps after 7. When the sender receives the ack due to packet 8, the same two possibilities hold, except that 6 must now be reordered after 8. If the cumulative sequence number repeats several times, the sender can guess with high probability that 6 was lost. The sender could use this information to retransmit 6. In general, the rule is that if a cumulative ack repeats several times, then the sender should retransmit the packet with a sequence number one larger than the cumulative ack. In connection-oriented networks, where packet reordering is very rare, even a single repeated cumulative ack is sufficient to trigger a retransmission. Note that this scheme does not require any timers. On the other hand, it does not work well if more than one packet was lost within an error-control window.

The advantage of selective retransmission is that a sender retransmits only the packets which were lost, thus conserving bandwidth. However, both the sender and the receiver need more complex error-control algorithms. Moreover, a receiver needs to buffer packets out of order, awaiting retransmission of a lost packet. As with go-back-n, selective retransmission works best with a limit on the size of the error-control window, since this limits the size of the buffer the receiver needs to reorder out-of-order packets. To see this consider the situation where the error control window is of

size 4, and the source has sent packets 5, 6, 7, and 8. The source cannot send packet 9 before it receives an ack for 5. Now, suppose that 5 is lost, but 6, 7, and 8 are correctly received. The receiver can either drop these packets, or hold on to them, awaiting 5. The source will eventually time out and retransmit 5. When the receiver receives 5, if it has saved 6, 7, and 8, it can pass packets 5-8 to the upper layer. Note that for this to work, the receiver may have to save all but one packet of the error-control window, which requires buffers. Thus, limiting the size of the error-control window reduces the number of buffers needed at the receiver.

SMART

A recent proposal, called SMART (Simple Method to Aid ReTransmissions), combines the benefits of selective retransmissions and go-back-n. In this scheme, each ack from a receiver carries two pieces of information: the last in-sequence packet received, and the sequence number of the packet that caused the ack to be sent. For example, if the receiver received packets 5, 7, 8, 10, it would send acks (5, 5), (5, 7), (5, 8), (5, 10), where the first element of the pair is the cumulative ack, and the second element is the sequence number of the packet initiating the ack. A sender can use this information to construct a bit map of the received packets, exactly as in the first selective retransmission scheme discussed above. However, we avoid the overhead of carrying a bit map in each ack.

When the sender sees a repeated cumulative acknowledgment, it does a fast retransmission, as discussed earlier. Continuing with our example, on receiving (5, 7), the sender immediately sends 6. When the sender gets (5, 8), it knows that 5 has been received, 6 has been retransmitted, 7 has been received, and so has 8. So, it takes no action. However, when it receives (5, 10), it notices that 9 has been neither retransmitted nor received; thus, it retransmits 9.

This scheme is not effective if retransmitted packets themselves are lost. To deal with that, the sender periodically inspects the last cumulative acknowledgment it received. If this value does not change for some time, the sender guesses that its retransmission of the cumulative acknowledgment+1 was lost, and retransmits it. In the worst case, on a timeout, the entire window is retransmitted, as in go-back-n, except that packets known to have been correctly received are not retransmitted. Thus, the scheme combines the efficiency of selective retransmission (but without its overhead) and the conservatism of go-back-n (but without its inefficiency). However, like selective retransmission, it requires a complicated algorithm and also buffers at the receiver.

1.3.8 Forward error correction

Unlike bit-level error control schemes, none of packet-level error control schemes we have discussed so far have used redundancy for error control. In fact, we can use redundancy for packet-level error control as well. This is called forward error *correction (FEC).*

For example, if all packets are the same size, and if every eighth packet is a parity check over the previous seven, the loss of any one of the eight packets can be corrected.

The advantage of forward error correction is that a receiver can recover from a packet loss without a retransmission. For real-time information, such as voice and video, the retransmission delay, which is at least as long as the round-trip time, is larger than can be ergonomically tolerated. Thus, FEC schemes are attractive for real-time data streams, and for links with long propagation delays, such as satellite links.

FEC has several disadvantages. First, error correction can substantially increase the load from a source. Because packet losses increase with load, FEC may actually cause more losses, degrading overall performance. Moreover, FEC is not effective when the packet losses are bursty, as is the case with high-speed networks. Finally, FEC increases the delay in end-to-end transmission, because a receiver must wait for the entire FEC block to be received before it can process the packets in the block. However, if packets from a stream arrive spaced far enough apart that they are not affected by the bursty losses, and the error correction overhead is designed to be small, then FEC schemes might perform well. Although some standards for audio and video transmission require FEC, at least from the research perspective, it appears that the jury is still out on FEC schemes.

1.4 Section summary

Data transmission in a network suffers from bit and packet errors. We distinguish between bit-level and packet-level error control, because they have different causes and different solutions. Both single-bit and multi-bit errors can be detected and corrected using schemes that add redundancy to the transmitted data, such as parity, Hamming codes, CRC, and convolutional codes. Packet errors include not only damage due to bit errors, but also losses, duplication, insertion, and reordering. Packet errors can be corrected with sequence numbers, timeouts, and retransmissions. Each of these techniques requires careful attention to details such as the choice of the initial sequence number, the choice of timer values, and determining which packets to retransmit. When properly done, error control allows us to reliably transport a packet across a series of unreliable links.

2.0 Flow Control

Consider a server transferring a file to a client after fragmenting it into packets. The server faces the nontrivial problem of choosing the rate at which to inject these packets into the network. If it sends all the packets at once, it may send them faster than the network or the client can process them, leading to packet loss. Because the server must retransmit lost packets, it is better off sending packets at a rate that both the network and the client can handle. If, however, the server sends packets slower than the highest sustainable rate, transferring the file takes longer than necessary. Thus, the server should carefully choose its transmission rate to neither overflow nor underflow the network or the client.

Flow control refers to the set of techniques that enable a data source to match its transmission rate to the currently available service rate at a receiver and in the network. Besides this primary goal, a flow control mechanism should meet several other, sometimes mutually contradictory objectives. It should be *simple* to implement, use the *least possible network resources* (in terms of bandwidth and buffers at multiplexing points), and work effectively even when used by many sources (that is, *scale* well). If possible, each member of the ensemble of flow-controlled sources sharing a scarce resource should restrict its usage to its fair share. Finally, the ensemble of sources should be *stable*, which means, loosely speaking, that when the number of sources is fixed, the transmission rate of each source settles down to an equilibrium value. Stability also implies that, if a new source becomes active, existing active sources adjust their transmission rates so that, after a brief transient period, the system settles down to a new equilibrium.

This range of objectives allows for many interesting trade-offs. For example, we can trade simplicity for fairness, designing a scheme that is simple to implement, but does not guarantee a fair share to every source. Other, more complex trade-offs are also possible, and we will see many of them later in this bloc. The variety of choices available to the designer has led to many different flow-control schemes being proposed in the literature (practically every networking conference in the 90's has one or more papers on flow control!). Some schemes described in the literature are only paper proposals, whereas others have been implemented in real networks and are widely used. In general, the more widely used a flow-control algorithm, the better it has been studied, and the more it deals with implementation difficulties. In this bloc, we will study flow-control schemes that either illustrate an important control mechanism or are widely used, or both. We will also evaluate the pros and cons of these schemes, based on how well they satisfy the objectives just described.

We can implement flow control at the application, transport, network, or datalink layer of a protocol stack. The choice of layer depends on the situation at hand. The most common design is to place end-to-end flow control at the transport layer, and hop-by-hop (link-level) flow control in the datalink layer. However, other arrangements are possible and, depending on the situation, are just as correct. In this bloc, we will study the abstract flow control problem, without worrying too much about layering.

We mention in passing that flow control is often confused with congestion control. *Congestion* refers to a sustained overload of *intermediate* network elements. Thus, flow control is one mechanism for congestion control.

This section is organized as follows. We present a model for flow control in Section 2.1, and a taxonomy of flow control schemes in Section 2.2. We can divide flow control techniques into three broad categories: *open loop, closed loop,* and *hybrid.* We only focus on open loop and closed loop, and will not address hybrid schemes. In Section 2.3, we study open-loop flow control, and in Section 2.4 we study closed-loop flow control. In each section, we study a handful of representative schemes that cover an illustrative range of flow-control techniques.

2.1 Flow control Model

We will study flow control in the context of a single source sending a stream of packets on a connection to a single destination or sink, over a path with many switches or routers. We assume that the sink acknowledges every packet. We model each network element, such as a switch, router, or multiplexing point, as a server that serves a certain number of packets from that connection per second. If the scheduling discipline is rate allocating, the service rate refers to the rate allocated to the connection. Otherwise, it is the instantaneous service rate available to the connection at that server. We call the slowest server along the path the bottleneck server. It was shown that, for the purposes of flow control, we can ignore all but the bottleneck server. Thus, the flow control model reduces to the one shown in Figure B4T41.

We can view flow control as *rate-matching with delays.* The bottleneck server removes data from its buffer at a variable rate. The source must match this rate so that the buffer neither overflows nor underflows. The problem is that we know the bottleneck server's current drain rate only after a delay, and the new source rate takes effect only after another delay. The sum of these delays is the *round-trip time (RTT),* which is the time taken for a packet to traverse the path from the

source to the sink and for its acknowledgment to return. (We can exclude the portion of the path from the bottleneck server to the sink if the bottleneck server directly informs the source of its current service rate.) The round-trip time is the fundamental time constant in all feedback flow-control mechanisms, because it is the minimum time required for a source to learn of the effect of its control actions.

Perhaps a more intuitive model of flow control is to imagine a tank of water from which water drains at a variable rate. A source must control its flow so that the tank neither overflows nor empties. If the source's control actions take effect immediately, the problem is straightforward, because the source can simply increase its flow whenever the tank is in danger of emptying, and decrease it when it nears an overflow. However, if the tank is situated, say, a hundred kilometers away, a change in the source's flow rate takes effect only after some time. Moreover, the source knows about the current water level only after a delay. The fundamental time constant in the system is the sum of the times taken for the source's changes to take effect, and for the source to learn of the effect of its change (the round-trip time).

Flow control is a variant of the classical control problem. In classical control, a controller is allowed to change its input to a black box and observe the corresponding output. Its aim choose an input as a function of the observed outputs, so that the system state conforms to some desired objective. The main difference in flow control is that the output of the system (the service rate of a connection) may depend not only on the actions of a particular source, but also on the actions of every *other* source sharing a resource with that connection. This coupling among sources makes flow control fundamentally hard, and not easily amenable to techniques from classical control theory.

2.2 Classification

We can classify flow control schemes into *open-loop, closed-loop,* and *hybrid* schemes. In *open-loop* flow control, a source describes its traffic to the network with a few parameters. During call establishment, the network reserves resources (such as bandwidth and buffers) corresponding to these parameters. During data transmission, if the source shapes its traffic to match its traffic's description, network overload, and thus congestion, is avoided. The difficult problem in open-loop flow control is choosing the right set of parameters to describe a source adequately. Once this is done, the actual flow control (or *regulation,* as it is usually called) is straightforward. We study open-loop flow control in Section 2.3.

In *closed-loop* schemes, a source dynamically adapts its flow to match its current share of network resources. As this share increases and decreases, a source should send faster or slower. There are many interesting and hard problems associated with closed-loop schemes. For example, how should the network inform a source that its service rate has changed? In an *explicit feedback* scheme, it explicitly conveys this information to the source. In an *implicit feedback* scheme, the source infers a change in its service rate by measuring its current performance. Once it receives this information, the source must decide how best to react to the current system state. This depends on its *flow-control strategy*. We study closed-loop flow control schemes in Section 2.4.

Finally, *hybrid* schemes combine aspects of open- and closed-loop flow control. For example, a source may reserve some minimum resources during call setup, but may be given a larger share if the network is idle. Thus, the source must do call setup, as in open-loop flow control, but also adapt to the network state, as in closed-loop flow control. We will not address such schemes in our study.

2.3 Open-loop flow control

In open-loop flow control, during call establishment. a source describes its behavior with a set of parameters called its *traffic descriptor* and negotiates bandwidth and buffer reservations with network elements along the path to its destination. The network operator prescribes the descriptor's parameters, and each source decides parameter values that best describe its traffic. During call setup, each network element examines this description and decides whether it can support the call. If it can. it forwards the setup request to the next element along the path. Otherwise, it negotiates the parameters down to an acceptable value, or blocks the call. In the data transmission phase, the source shapes its traffic to match its descriptor, and each network element schedules traffic from admitted calls to meet the bandwidth, delay, and loss guarantees it makes to them. The hard problems in open-loop flow control are (a) choosing a descriptor at a source, (b) choosing a scheduling discipline at intermediate network elements, and (c) admitting calls so that their performance objectives are met (call admission control). We study the choice of descriptors in Section 2.3.1, scheduling disciplines in bloc 7, but we will not address call admission control in our study.

In open-loop flow control, a source has to capture its entire future behavior with a handful of parameters, because the network's admission-control algorithm uses these parameters to decide whether to admit the source or not. Thus, open-loop flow control works best when a source can describe its traffic well with a small number of parameters, and when it needs to obtain quality-of-service guarantees from the network. If either of these conditions fails to apply, the

source is better off with closed-loop or hybrid flow control.

2.3.1 Traffic Descriptors

A traffic descriptor is a set of parameters that describes the behavior of a data source. Typically, it is a behavior envelope, that is, it describes the worst possible behavior of a source, rather than its exact behavior. A descriptor plays three roles besides describing source traffic. First, it forms the basis of a traffic contract between the source and the network: the source agrees not to violate the descriptor, and in turn, the network promises a particular quality of service. If a source violates its part of the contract, the network cannot guarantee it a performance bound. Second, the descriptor is the input to a regulator, a device through which a source can pass data before it enters the network. To ensure that the source never violates its traffic descriptor, a regulator delays traffic in a buffer when the source rate is higher than expected. Third, the descriptor is also the input to a policer, a device supplied by the network operator that ensures that the source meets its portion of the contract. A policer delays or drops source traffic that violates the descriptor. The regulator and policer are identical in the way they identify descriptor violations: the difference is that a regulator typically delays excess traffic, while a policer typically drops it.

A practical traffic descriptor must have at least these important properties:

- Representativity: The descriptor must adequately represent the long-term behavior of the traffic, so that the network does not reserve too little or too much.
- Verifiability: The network must be able to verify that a source is obeying its promised traffic specification quickly, cheaply, and preferably in hardware.
- Usability: Sources should be able to describe their traffic easily, and network elements should be able to perform admission control with the descriptor easily.

Coming up with good traffic descriptors is difficult because of these conflicting requirements. For example, the series of times at which a source places data onto a connection is a representative, and verifiable traffic descriptor. However, this time series is potentially very long and, for interactive traffic sources, is unknown. Thus, the descriptor is unusable. In contrast, if we choose the source's peak rate as its descriptor, the descriptor is usable and verifiable, but not representative, because resource reservation at the peak rate is wasteful if a source rarely generates data at this rate.

Several traffic descriptors have been proposed in the literature. They are roughly equivalent, though there are subtle differences in their ease of use and descriptive power. We study three common descriptors: peak rate, average rate, and linear bounded arrival process. For each descriptor, we also study the corresponding regulator.

2.3.2 Peak rate

The peak rate is the highest rate at which a source can ever generate data during a call. A trivial bound on the peak rate of a connection is just the speed of the source's access link, because this is the instantaneous peak rate of the source during actual packet transmission. With this definition, a source on a 10-Mbps Ethernet that generates one 100-byte packet per second can be said to have a peak rate of 10 Mbps! Although accurate, this definition is not satisfactory because it does not give a true picture of a source's traffic load. Instead, we measure the peak rate in one of two ways. For networks with fixed-size packets, the peak rate is the inverse of the closest spacing between the starting times of consecutive packets. For variable-sized packets, we must specify the peak rate along with a time window over which we measure this peak rate. Then, the peak rate bounds the total number of packets generated over all windows of the specified size.

Example 2.1

(a) If all packets on a connection are 50 bytes long, and the closest packet spacing is 10 ms, what is the peak rate? (b) If the peak rate of a connection is 8 Mbps over all intervals of 15 ms, what is the largest amount of data that can be generated in 75 ms? (c) In 70 ms?

Solution: (a) The peak rate is 5000 bytes/s. (b) The largest amount allowed in 75 ms is 8 Mbps * 75 ms = 600,000 bits. (c) Since traffic is specified only over an interval of 15 ms, the worst-case amount of data generated in 70 ms is also 600,000 bits (for example, all the data could be generated in the first 5 ms of every consecutive 15-ms interval).

A peak-rate regulator consists of a buffer and a timer. For the moment, assume a fixedsize packet network. When the first packet in a call arrives at the buffer, the regulator forwards the packet and sets a timer for the earliest time it can send the next packet without violating the peak-rate bound, that is, the smallest interarrival time. It delays subsequently

arriving packets in a data buffer until the timer expires. If the timer expires before the next packet arrives, it restarts the timer on packet arrival, and the incoming packet is forwarded without delay. The generalization of this scheme to networks with variable-sized packets is left as an exercise to the reader.

The peak-rate descriptor is easy to compute and police, but it can be a very loose bound, because it is an extremal bound. That is, a single outlier can change this descriptor. For example, consider a data stream where a source generates one fixed-size data packet exactly once a second. It has a peak rate of 1 packet per second. However, even if one packet in the stream "slips" and is sent, say, 10 ms after an earlier packet, the peak rate increases to 100 times its previous value! Peak-rate descriptors are useful only if the traffic sources are very smooth, or if a simple design is more important than efficiency.

2.3.3 Average rate

The key problem with the peak rate is that it is subject to outliers. The motivation behind average-rate descriptors is that averaging the transmission rate over a period of time reduces the effect of outliers. Researchers have proposed two types of average-rate mechanisms. Both mechanisms use two parameters, *t* and *a*, defined as follows:

t = time window over which the rate is measured

a = the number of bits that can be sent in a window of time *t*

In the *jumping-window* descriptor, a source claims that over consecutive windows of length *t* seconds, no more than *a* bits of data will be transmitted. The term "jumping window" refers to the fact that a new time interval starts immediately after the end of the earlier one. The jumping-window descriptor is sensitive to the choice of the starting time of the first window.

In the *moving-window* scheme, the time window moves continuously, so that the source claims that over *all* windows of length *t* seconds, no more than *a* bits of data will be injected into the network. The moving-window scheme (also called the (*r,* T) model) removes the dependency on the starting time of the first window. It also enforces a tighter bound on spikes in the input traffic.

An average-rate regulator is identical to a variable-packet-size peak-rate regulator, because both restrict the maximum amount of information that can be transmitted in a given interval of time. For a jumping-window descriptor, at time 0, a counter is initialized to 0 and is incremented by the packet size of each departing packet. Every *t* seconds, the counter is reset to 0. When a packet arrives, the regulator computes whether sending the packet would result in too much data being sent in the current window. This test reduces to testing whether the sum of the current counter value and the current packet size is larger or smaller than *a*. Depending on the result, the regulator either forwards the packet immediately or buffers it until the next time window.

In one technique for building a moving-window descriptor, besides the counter described earlier, the regulator stores the departure time and packet size of every departing packet. The test for delaying or forwarding a packet is the same as before. In addition, *t* seconds after a packet departs, the counter is decremented by its size. Thus, the counter reflects the number of bits sent in the past *t* seconds. If the regulator decides to delay a packet, it can determine the earliest time it can transmit the packet by examining the list of packet departure times. This technique is not used in practice because it requires the regulator to store a lot of information, which is hard to do at high speed. A second technique for building a moving-window regulator, which is the one used in practice, is the leaky-bucket regulator described in the next subsection (2.3.4).

Example 2.2

An average-rate descriptor is specified with *a* = 100 Kbytes, *t* = 1 s. Packet arrival times and sizes are (0.2 s, 20 Kbytes), (0.25 s, 40 Kbytes), (0.5 s, 20 Kbytes), (0.6 s, 20 Kbytes), (0.8 s, 10 Kbytes), (1.0 s, 30 Kbytes), (1.7 s, 30 Kbytes), (1.9 s, 30 Kbytes). What are the departure times with the jumping-window and moving-window regulators?

Solution: With a jumping window, the packet arriving at time 0.8 s is delayed to the second window. With a moving window, the packet arriving at time 0.8 s is delayed to time 1.0. The packet arriving at time 1.0 is delayed until a time x such that no more than 100 Kbytes have been sent in the interval [*x - t, x*], i.e., 1.25 s. At this time, the effects of the first two packets are erased and the packet arriving at time 1.0 can be sent. For both regulators, the last two packets depart as soon as they arrive because the jumping-window counter is not exceeded, and by time 1,7, the moving-window counter is 40 Kbytes.

2.3.4 Linear bounded arrival processes

Linear bounded arrival processes, or LBAPs, are a popular class of source descriptors. An LBAP-constrained source bounds the number of bits it transmits in any interval of length t by a linear function of t. We characterize this linear function by two parameters, σ and ρ so that:

Number of bits transmitted in any interval of length $t \leq \rho t + \sigma$

ρ corresponds roughly to the long-term average rate *allocated* by the network to the source (which may be substantially larger than the source's true average rate), and σ the longest burst a source may send, given the choice of ρ while still obeying the LBAP descriptor. In other words, assuming for the moment that ρ is the source's average rate, an LBAP characterizes a source that has an intrinsic long-term average rate ρ, but can sometimes deviate from this rate, as specified by σ. Since an LBAP is a generalization of the average-rate descriptor, it is also insensitive to outliers.

A *leaky-bucket regulator* regulates an LBAP descriptor. Intuitively, the regulator collects *tokens* in a bucket, which fills up at a steady drip rate. Each token is permission for the source to send a certain number of bits into the network. When a packet arrives at the regulator, the regulator sends the packet if the bucket has enough tokens. Otherwise, the packet waits either until the bucket has enough tokens or until the packet is discarded. If the bucket is already full of tokens, incoming tokens overflow and are not available to future packets. Thus, at any time, the largest burst a source can send into the network is roughly proportional to the size of the leaky bucket.

More formally, a leaky bucket accumulates fixed-size tokens in a token bucket and transmits a packet only if the sum of the token sizes in the bucket adds up to the packet's size (Figure B4T52). On a packet departure, the regulator removes tokens corresponding to the packet size from the token bucket. The regulator periodically adds tokens to the bucket (at a rate ρ). However, the bucket overflows if the number of tokens crosses some threshold, called its *depth*, σ. A leaky bucket limits the size of a transmitted burst to a little more than the bucket's depth (since tokens may arrive while the bucket's worth of packets are being transmitted), and over the long term, the rate at which packets depart the regulator is limited by the rate at which tokens are added to the bucket. The regulator delays a packet if it does not have sufficient tokens for transmission. Typically, we initialize the bucket to be full.

Example 2.3

Tokens of size 100 bytes are added to a leaky-bucket regulator of capacity 500 bytes twice a second. (a) What is the average rate, peak rate, and largest burst size of the regulated traffic stream? (b) Can this regulator handle a packet of size 700 bytes? (c) If a packet of size 400 bytes arrives when the bucket contains tokens worth 200 bytes, and there are no other packets awaiting service, what is the least and most delay it could have before transmission?

Solution: (a) The average rate is 200 bytes/s = 1.6 Kbps. The largest burst size is 500 bytes. The peak rate is unbounded, because a burst of up to 500 bytes can be transmitted arbitrarily fast. (b) No, because the packet will never have enough tokens to be transmitted. (c) If the packet arrives just before the arrival of a token, it needs to wait for only a little over 0.5 s; if it arrives just after the token, it has to wait for 1 s.

A leaky bucket can be used both as a peak-rate and a moving-window average rate regulator, because they are both special cases of an LBAP. If the token replenishment interval corresponds to the peak rate, and the token bucket size is set to one token, then the leaky bucket is a peak-rate regulator. Similarly, setting the token-bucket limit to one token and replenishing the bucket at the average rate makes it a moving-window average rate regulator. (In both cases, with variable-sized packets, we have to be careful that the token is at least as large as the largest packet.)

In a common variant of the leaky-bucket regulator, the token bucket is augmented with a peak-rate regulator, so that packet bursts arrive into the network no faster than this peak rate. This allows us to control the average rate, the peak rate, and the largest burst from a source.

Note that a leaky bucket regulator has both a token bucket and a data buffer (if it did not have a data buffer, it would be a policer). Packets that arrive to the regulator that cannot be sent immediately are delayed in the data buffer. Intuitively, the larger the token bucket, the smaller the data buffer need be, because an arriving packet uses a token and departs the regulator, instead of being delayed in the data buffer. In fact, it was shown that the performance of a leaky bucket with a data buffer and a token bucket (in terms of the packet loss rate from the regulator) depends only on the sum of the token-bucket size and the data-buffer size. In other words, confirming our intuition, a larger token bucket size exactly offsets a smaller data buffer.

Choosing LBAP parameters

Given an intrinsically limited traffic source, such as a stored compressed video, we would like to come up with an LBAP descriptor for the source that is minimal in the sense that no other descriptor has both a smaller σ and a smaller

p. If a source must pay for the resources it consumes, a minimal descriptor is likely to have the smallest price. Unfortunately, the minimal LBAP descriptor for a source is not unique. Given the size of the data buffer at the regulator and the maximum loss allowed at the regulator, each choice of the token arrival rate has a corresponding minimum burst size so that the loss parameter is met. To see this, consider a source that has some intrinsic peak rate P and average rate A measured over a long interval. If the token arrival rate ρ is less than or equal to A then the regulator buffer grows without bound, and σ must be infinite if we want to avoid packet loss. If ρ > P, then there are always tokens on hand when a packet arrives, and σ can be as small as one maximal-sized packet (Figure B4T55). As we increase ρ in the range [A, P], the minimum σ needed to meet the loss bound decreases. Any p and its corresponding σ is an equivalent minimal descriptor of the source. The set of all (σ, ρ) pairs that form the minimal LBAP descriptors for a source are described by the (σ, ρ) curve for the source.

For arbitrary sources, there is no easy way to pick the appropriate (σ, ρ) pair that describes it "best." However, for many common sources, the (σ, ρ) curve has a distinct "knee," which makes the choice of parameters straightforward. A knee in the (σ, ρ) curve indicates that for descriptors that are slightly away from the knee, either the σ or the ρ parameter rapidly increases. Thus, the optimal choice of the parameters is the value at the knee (Figure B4T55).

LBAP descriptors are popular in practice and also in academic papers, because they are usable and verifiable. They correctly model the fact that even a "smooth" source may have periods in which it is bursty. However, they do not accurately represent sources that have occasional very large bursts. For such sources, if the regulator delay is to be small, and the loss probability low, the σ parameter has to be chosen to be fairly large so that the burst is drained away into the network. Unfortunately, this makes the network more expensive, because it has to size internal buffers to be large enough to handle large bursts. A better solution is to renegotiate the LBAP descriptor just before the burst (if the occurrence of a burst can be predicted, or is already known, as with stored video), so that we increase the drain rate to handle the burst, then decrease it back to the long-term average rate. If a burst lasts long enough, we can start renegotiation *after* detecting the start of the burst. However, this does not perform as well, because the regulator's data buffer fills while the renegotiation procedure, which can take more than one round-trip time, is going on..

2.4 Closed-loop flow control

In open-loop flow control, a source specifies a traffic descriptor during call establishment, and, during data transfer, ensures that its traffic meets this description. Even if the network load changes while the call is in progress, an admitted source need not change its descriptor or its traffic, because each network element reserves sufficient resources to meet the source's performance requirements.

In closed-loop flow control, we assume that network elements do not reserve sufficient resources for the call, either because they do not support resource reservation, or because they overbook resources to get additional statistical multiplexing gain. In Bloc 7, we will study how a network element can use a scheduling discipline to dynamically allocate transmission rates to an ensemble of feedback flow-controlled sources. Given such an allocation, the aim of closed-loop flow control is to adjust a source's transmission rate dynamically, in response to feedback signals, so that the ensemble of sources does not overload the network. If closed-loop flow control is ineffective, sources either suffer excessive packet loss or underutilize network resources.

Taxonomy of closed-loop flow-control schemes

In general, a traffic source must control its transmission rate not only in response to the receiver's state, but also in response to the network state. The *first generation* of flow-control protocols did not explicitly consider network state; they simply matched the source rate to the service rate at the destination. The three important protocols in this generation, which we will study in Sections 2.4.1-2.4.3, are *on-off*, *stop-and-wait,* and *static-window* flow control.

The second generation of protocols changes the source rate in response to both the sink state and the network state. We can categorize these protocols in three complementary ways:

Implicit versus explicit state measurement: With explicit measurement, a network element uses an explicit control message to communicate the current sustainable data rate to every source. In an implicit measurement scheme, a source uses performance measurements to dynamically infer its share of network bandwidth. Explicit schemes can control a source's rate more precisely than implicit schemes, because a source has better information. On the other hand, they require both a communication and a computation overhead. Ideally, we would like a scheme that has as little overhead as an implicit scheme, but performs nearly as well as an explicit scheme.

Dynamic window versus dynamic rate: We define the error-control window to be the number of packets sent from a source, but yet to be acknowledged. Because the source must stop after it has a window's worth of packets in flight (see Section 1.3.7 in this bloc for more details), by limiting the error-control window size, we automatically limit the source's transmission rate. Thus, we can use the error-control window for flow control. To distinguish between these

distinct uses of a window, we will call the window used for flow control the *flow-control window* or the *transmission window*. In an *adaptive-window* scheme, we indirectly control a source's transmission rate by modifying its transmission window. In an *adaptive-rate* scheme, we directly control the source rate. Every time a source sends a packet, it sets a timer with a timeout value equal to the inverse of the current transmission rate, and transmits the next packet when the timer expires.

A dynamic-window scheme has the disadvantage that the window is used for both error control and rate control. This coupling is often problematic. For example, if a receiver has only a few buffers to hold out-of-order packets, and error control is based on selective retransmission, the error-control window must be small. Unfortunately, this limits the maximum transmission rate from the source. We comment on some other disadvantages of window-based flow control at the end of Section 2.4.5, after we have more context for these comments.

Window-based control has two main advantages over rate-based flow control. First, it is easier to implement, because it does not require a fine-grained timer, which can be expensive in some systems. Second, a window automatically limits the damage a source can inflict on a network. After transmitting a window's worth of packets, a source stops. With rate-based flow control, a source may continue to send packets into the network if it fails to receive rate-throttling information. Thus, we must carefully engineer rate-based flow control to be robust to packet loss and corruption.

Hop-by-hop versus end-to-end control: We can automatically make a first-generation flow-control scheme responsive to network state (in addition to receiver state) by implementing it between every adjacent pair of network elements. For example, stop-and-wait becomes a second-generation flow control scheme if we use it not just between a source and a sink, but also between every pair of adjacent routers. This change typically improves performance: the control delay is smaller, and each element only needs to react to a change in the next element, which is typically more effective than responding to changes in all elements along the path. Besides, by limiting the buffer buildup at each element, a hop-by-hop scheme more evenly distributes buffer usage. Hop-by-hop schemes, however, make the network elements more complicated. Moreover, unless hop-by-hop control is done per-connection, it can be unfair.

Because these design elements are complementary, we can come up with eight possible combinations. Note that there are no implicit hop-by-hop closed-loop flow control schemes. An implicit scheme tries to minimize the work done within the network by guessing the available service rate in the network. A hop-by-hop scheme requires a considerable amount of work to be done at each switch, so the additional overhead for explicit control is small. Thus, hop-by-hop schemes tend to use explicit control.

In Sections 2.4.1-2.4.8 we will study some flow control schemes. Of these schemes, we will study the TCP scheme in more detail than the others schemes because of it is widely implemented on a variety of platforms.

2.4.1 On-off

In on-off flow control, the receiver sends the transmitter an *On* signal when it can receive data, and an *Off* signal when it can accept no more data. The transmitter sends as fast as it can when it is in the On state and is idle when it is in the Off state.

Evaluation

On-off control is effective when the delay between the receiver and the sender is small. It works poorly when the propagation delay between the sender and receiver is large, because the receiver needs to buffer all the data that arrive before the Off signal takes effect. If the Off packet is delayed or lost, the receiver continues to receive data at the source's peak rate, leading to potential buffer overflow and loss. Moreover, intermediate network elements are subjected to abrupt data bursts from the sources, making packet loss in the network more likely.

On-off control is primarily used over short serial lines or LANs, where propagation delays are small and packet losses are rare. It is the basis for the XON/XOFF protocol used to control serial input-output devices such as printers and mice.

2.4.2 Stop-and-wait

In the stop-and-wait protocol, one of the earliest attempts at flow control, a source sends a single packet and waits for an acknowledgment before sending the next packet. If it receives no acknowledgment for some time, it times out and retransmits the packet.

Stop-and-wait simultaneously provides error control and flow control. It provides error control because if a packet is lost, the source repeatedly retransmits it until the receiver acknowledges it. It provides flow control because the sender waits for an acknowledgment before sending a packet. Thus, stop-and-wait forces the sender to slow down to a rate slower than can be supported at the receiver.

Evaluation

Stop-and-wait is useful in networks where the propagation delay is small, but is inefficient otherwise (see Figure B4T65). In the figure, each vertical line corresponds to a network element. Time increases along the vertical axis. We represent a packet transmission as a parallelogram: the slope of the parallelogram represents the link delay, and the width is inversely proportional to the transmission rate of the link (thus, the width is also proportional to the time a packet spends on a link). It is clear from the figure that a source must wait for one round-trip time to elapse after it sends a packet before it can send the next one. Thus, the best possible throughput is one packet per round-trip time. This decreases rapidly as the propagation delay, relative to the packet transmission time, increases.

Example 2.4

What is the peak throughput achievable by a source employing stop-and-wait flow control when the maximum packet size is 1000 bytes, and the network spans (a) 10km, (b) 5000 km?

Solution: Assuming a direct fiber-optic line between endpoints, the speed-of-light propagation delay is $1/(0.7 * 3 * 10^5$ s/km, since the speed of light in fiber is approximately 0.7c, where c (the speed of light in vacuum) = $3 * 10^5$ km/s. This works out to 4.76 μs/km. For the purposes of this example, we will ignore the effects of queuing and switching delays. (a) For a 10 km line, the round-trip delay is thus 2 * 47.6 μs = 95.2 μs The maximum possible throughput is thus 1 packet/RTT = 1000 * 8 bits/95.2 μs = 84.03 Mbps. (b) Since the link is 500 times longer, the maximum speed goes down by a factor of 500 to 84.03/500 Mbps = 0.168 Mbps.

2.4.3 Static window

Stop-and-wait flow control has a peak throughput of one packet per round-trip time (RTT). This is inefficient if the propagation delay, and thus the RTT, is large. In static-window flow control, we allow a source to send up to *w* packets before it stops and waits for an acknowledgment. In other words, we have:

Transmission window = *w*

When a source that has *w* packets outstanding receives an ack, we allow it to transmit another packet, because the left edge of the sliding window slides one step forward. It takes at least one round-trip time for the source to receive an ack for a packet. In this time, it could have sent at most *w* packets. Thus, its maximum achievable throughput is *w*/*RTT* packets/s, a factor of *w* better than stop-and-wait. Figure B4T66 illustrates a source allowed to have three packets outstanding at any time, i.e., has a window of size three. Thus, its throughput is almost triple that of a corresponding stop-and-wait source. How large should *w* be? Let:

Bottleneck service rate along the path = μ packets/s

Round-trip time = *R* s

The source's sending rate is at most *w*/*R* packets/second. If this is to keep the bottleneck fully utilized, we must have:

$w/R \geq \mu \Rightarrow w \geq R\mu$

If this is an equality, that is, $w = R\mu$, the source transmits packets exactly as fast as the bottleneck can handle them, and the flow control is optimal. If $w < R\mu$, then the bottleneck is idle for part of the round-trip time, and if $w > R\mu$,

Number of packets buffered at the bottleneck = $w - R\mu$

Because of the importance of the value $R\mu$, we call it the *bandwidth-delay product* or the *optimal window size.*

Example 2.5

Compute the optimal window size when packet size is 53 bytes, the RTT is 60 ms, and bottleneck bandwidth is (a) 1.5 Mbps (the standard T1 trunk speed), (b) 155 Mbps (the standard OC-3 trunk speed).

Solution: (a) The bottleneck rate in packets/see is 1.5 Mbps/53 * 8 bits/packet = 3537.7 pkts/s. Thus, the optimal window is 3537.7 * 0.06 = 212.3 packets. (b) Similarly, at OC3 rates, the bottleneck rate is 365,566 pkts/s, and the optimal window is 21,933 packets.

Evaluation

The problem with static-window flow control is that the optimal window size depends on the bottleneck service rate and the round-trip time. However, not only do the bottleneck rate and the round-trip time differ from connection to connection, but they also vary with time even for the same connection. Choosing a single static window size suitable for all connections is therefore impossible. Unfortunately, a poor choice of w can lead to severe performance problems. If w is too small, a source may have data to send, and the bottleneck may have spare capacity, but the window size prevents the source from using the available capacity. If w is too large, $w - R\mu$ packets are buffered at the bottleneck; if the bottleneck does not have sufficient buffering, packets may be lost. We avoid these problems in the second generation of flow-control protocols by *dynamically* varying a connection's window size to be always close to the current optimal.

2.4.4 DECbit flow control

The DECbit is an explicit, end-to-end dynamic window flow control scheme. The key idea behind the DECbit scheme is that every packet header carries a bit that can be set by an intermediate network element that is experiencing congestion (i.e., a sustained queue buildup). The receiver copies the bit from a data packet to its acknowledgment, and sends the acknowledgment back to the source (Figure B4T69). The source modifies its transmission-window size based on the series of bits it receives in the acknowledgment headers as follows: The source increases its windows until it starts building queues at the bottleneck server (because its window size is larger than the optimal window size), causing that server to set bits on the source's packets. When this happens, the source reduces its window size, and bits are no longer set. In equilibrium, therefore, the source-window size oscillates around the optimal window size. If the propagation delay R or the bottleneck service rate μ changes, the source-window size adapts to this change and oscillates about a new optimal point. Note that the scheme does not require any particular scheduling discipline at multiplexing points. In our discussion, we assume, as the authors did, that this is first-come-first-served.

Evaluation

The DECbit scheme has several useful properties. It requires only one additional bit in the packet header and does not require per-connection queuing at servers. Endpoints can implement the scheme in software, without additional hardware support. Moreover, many simulations, and experience in the field, have shown that the control is stable.

However, the DECbit scheme has two serious flaws. First, it assumes that the endpoints are cooperative. If a source chooses to ignore the bits it receives in its acks, it can drive the server to the panic state, so that all other sources sharing the server are affected. Thus, a single malicious or misbehaving source can affect the performance of all other sources. We will see in bloc 4, that to control misbehaving or malicious sources, a network must provide either per-connection queuing, or per-connection policing using a traffic descriptor.

The second problem with DECbit is that it has a very conservative window increase policy. If the initial window size is 5 and the optimal window size is 200, the source will take 390 RTTs to reach this value, because the window increases by only 1 every two RTTs. If a source does not have much data to send, it finishes its transmission before reaching its optimal window size! Thus, DECbit performs poorly in networks where the bandwidth-delay product is large, which is expected to be the case for future wide-area networks.

2.4.5 TCP flow control

TCP slow-start is an implicit end-to-end dynamic window flow and congestion control scheme. The flow-control scheme in TCP, designed by Jacobson and Karels, is similar to the DECbit scheme, but differs in one important detail. Instead of receiving explicit congestion information from network elements, a source dynamically adjusts its flow control window in response to implicit signals of network overload. Specifically, a source increases its window size until it detects a packet loss. At this point, the source reduces the window size, and the cycle repeats (Figure B4T76).

A source starts with a window of size 1 and increases it exponentially until the window size reaches a threshold, and linearly after that. We maintain the current window size in a floating-point number, the integral part of which is the window size (this is explained further in Example 2.6). In the exponential (or *slow-start*) phase, a source increases its window by 1 every time it receives an ack. If every packet is acked, this doubles the window size every RTT (in

practice, some receivers send only one ack for every two packets, halving the increase rate). In the linear (or *congestion-avoidance*) phase, the source increases its window by 1/⌊current_window⌋ every time it receives an ack. This increases the window by 1 every RTT. A threshold variable called the *slow-start threshold* or *ssthresh* controls the transition from the exponential to the linear phase. The source initializes this variable to half the initial window size, and, on detecting a loss, resets it to half the current window size.

There are two widely used variants of TCP. The *Tahoe* version detects losses using timeouts. On a timeout, it decreases its flow control window to 1, sets *ssthresh* to half the current window size, and enters slow start. The window thus rises exponentially to half its previous value, then continues with a linear increase. The *Reno* version detects losses using both timeouts and the receipt of three acks with the same cumulative sequence number (the fast *retransmit scheme* explained in Section 1.3.7). On a timeout, a TCP Reno source behaves in the same way as a TCP Tahoe source. However, on a fast retransmit, it decreases both *ssthresh* and its flow-control window to half its previous value. At this point, since the flow-control window is already as large as *ssthresh*, the source goes directly into the linear increase phase, skipping the exponential increase phase. Moreover, after a fast retransmit, the source is allowed to send one packet for each duplicate cumulative ack received, even if this causes it to exceed the flow-control window size (this is called fast recovery). The intuition is that each cumulative ack, even if duplicate, signals the availability of network resources. By inflating the actual window beyond the nominal window, a source can exploit this capacity. This window inflation ceases when a source receives the first non-duplicate cumulative ack, indicating that the retransmission succeeded.

Example 2.6

Let us trace the evolution of the window size of a source that is transmitting data over a connection with RTT 1. Assume that its initial value for *ssthresh* is 5, and the largest allowed flow control window is 10. We will also assume that the bandwidth-delay product on the connection is 4 (that is, the bottleneck service rate is 4 packets /second), and the bottleneck has a buffer size of 4 packets.

In the first RTT, the source sends packet 1 at time 0 and receives an ack at time 1. At the end of the first RTT, the source increases its window by 1 for this ack, doubling its window to 2. Thus, at time 1, it sends packets 2 and 3. As each ack arrives, the window increases by 1, so that at the end of the second RTT, the window is 4.

In the third RTT, the source sends packet 4, 5, 6, 7. At the end of the third RTT, when the ack for 4 arrives, the window increases to 5 and reaches the slow-start threshold. Thus, the acks for 5, 6, 7, which arrive during the fourth RTT, each contributes 1/5 to the window size, and when the ack for 7 is received, the window reaches 5.6 (only the integer portion is used for flow control).

In the fourth RTT, five packets can be outstanding, and the source transmits 8, 9, 10, 11, 12. When the source receives the ack for 9, during the fifth RTT, the window finally increases to 6, and acks for 10, 11, and 12 each contribute 1/6 to the window. Thus, at the end of the fifth RTT, the window reaches 6.5. In the absence of losses, the window increases slowly until it reaches nine packets during the eighth RTT. Of these, four are "in the pipeline," because the bandwidth delay product is 4. Four more can be buffered in the bottleneck buffer. Thus, one packet is lost at the bottleneck (the packet with sequence number 42). This causes the receiver to repeat the cumulative acknowledgment on packets sent during the ninth RTT, triggering a fast retransmission.

At the start of the ninth RTT, the source has nine packets outstanding and has a window size of 9.333. During the ninth RTT, the source receives acks for packets 34-41, which increases its window, at the end of the ninth RTT, to 10.2. Thus, the source, during the 9th RTT, sends ten packets, 43-52. The last ack received during the ninth RTT is the ack for 41, which was the last packet sent before a packet loss.

In the tenth RTT, the source receives acks for packets sent in the ninth RTT. The acks for packets 43-52 all carry the same cumulative ack number, that is, 41 (these acks increase the window from 10.2 to 10.5). On the third such duplicate, the source invokes fast retransmission (see Section 1.3.7) and retransmits 42. (Meanwhile, when it got the acks for 43 and 44, it transmitted 53 and 54.) In both TCP-Tahoe and TCP-Reno, fast retransmission causes the source to set its *ssthresh* value to ⌊10.5/2⌋ = 5. TCP-Reno sets its window size to 10.5/2 = 5.25 and enters the linear increase phase. Thus, in the tenth RTT, a TCP-Reno source has five packets outstanding: 42 and 53-56. In TCP-Tahoe, the window drops to 1. It cannot send any more packets after 42. Thus, in the tenth RTT, a TCP-Tahoe source has three packets outstanding: 53, 54, and 42. In two round-trip times, the window comes back to 4. and the source reenters the linear increase phase.

The table below illustrates the behavior of TCP-Reno.

RTT #	Window range during RTT	Packets sent in this RTT	*ssthresh*	Peak buffer size

1	1-2	1	5	0
2	2-4	2, 3	5	0
3	4-5	4-7	5	0
4	5-5.6	8-12	5	1
5	5.6-6.5	13-18	5	2
6	6.5-7.428	19-25	5	3
7	7.428-8.375	26-33	5	4
8	8.375-9.333	34-42	5	4/drop
9	9.333-10.2	43-52	5	4/drop
10	10.2-10.5/5.25	42, 53-56	5	1

This example is somewhat artificial, because it uses packet sequence numbers rather than TCP's segment ranges, and because the RTT is always 1, without any variation during the connection. Many real-life traces of TCP in action, with copious explanations, can be found in the Stevens books given in reference in B1T4).

The TCP-Tahoe and TCP-Reno flow-control algorithms have motivations similar to the DECbit algorithm we studied in previous section. Both use dynamic-window flow control. In either case, a source uses a conservative window-increase policy (exponential and then additive in TCP, additive in DECbit), and a multiplicative decrease policy. We know that an additive-increase, multiplicative-decrease policy is stable, unlike variations such as additive increase, additive decrease, or multiplicative increase, additive decrease, so this coincidence is not surprising. The main difference between the algorithms is that the TCP-Tahoe and -Reno algorithms do not require explicit information about the current congestion status from the network. They view the network as a black box, which they probe by increasing the transmission-window size and looking for packet loss. In contrast, with DECbit, the network explicitly communicates information about queue buildups to the sources. A second difference is that the two TCP algorithms do not filter information at the source, for example, by treating multiple packet losses over a time interval as an indication of congestion, instead of a single packet loss. This is because they operate the network close to overload, and unless they cut back the window size immediately after detecting a loss, the network is in danger of sustained overload.

Evaluation

TCP flow-control algorithms are effective over a wide range of bandwidths, and because of their popularity, there is considerable experience in understanding their performance. Moreover, they are about the best that one can do when the network does FIFO queuing (the common case), and an endpoint cannot make any assumptions about the structure of the network or its operation.

However, the algorithms have several weaknesses. First, they assume that any loss is an indication of an overload, and thus immediately reduce the window size. This is because most packet networks currently are nearly loss free, except for congestive losses. Wireless networks, however, have a different loss behavior, and multipath interference or shadowing effects can lead to frequent link-level errors. In such situations, assuming random loss, it can be shown the effective throughput of a TCP connection decreases in proportion to $p(R\mu)^2$, where p is the probability of a random loss, and $R\mu$ is the bandwidth-delay product expressed in packets. For a cross-country link where the first hop is over a wireless link, typical window sizes are around 15 packets (assuming a packet size of 500 bytes, a round-trip delay of 60 ms, and an available bandwidth of 1 Mbps). Thus, on this connection, the throughput degrades as $1/225p$, where p is the probability of a random loss on the wireless link. As the available bandwidth increases, μ increases linearly, but the degradation in throughput goes up as μ^2 , so that the effective throughput *decreases!*

Many solutions have been proposed for this problem. One approach is to use link-level error control to make the link appear error free. The problem is that the link-level and TCP-level error control schemes must coordinate their actions, to avoid, for example, retransmission at both the link level and the TCP level. However, TCP is unaware of the link's actions, leading to many subtle problems. A second approach is to modify TCP's flow-control strategy to allow it to distinguish between congestive losses and link losses. The datalink layer informs TCP about link-level losses, which TCP retransmits *without* shutting down the flow-control window. This, too, is problematic because it is a layering violation. Moreover, it only works if the wireless hop is the first or last hop in the connection. The question of how best

to modify TCP to deal well with non-congestive losses is still an area of active research.

A second problem with TCP is that a TCP source always assumes that a packet loss is due to its own large window size, and thus cuts back on its window. When used in networks with FCFS queuing, this leads to situations where the presence of a single malicious or misbehaved source causes all the TCP connections to back off immediately, giving a free hand to the misbehaving source. Using a round-robin-like algorithm at the queuing points can solve this problem, though only at the expense of per-connection queuing at multiplexing points.

Third, the algorithm detects overload using loss. Thus, each endpoint tries to increase its window until some buffer overflows. Even in the steady state, bottleneck buffers are kept full, so that the arrival of a burst of traffic is very likely to cause packet loss. Each lost packet represents wasted work, so using loss to measure the state of the network is inherently wasteful.

Fourth, for short transfers (which constitute the bulk of connections on the Internet), the algorithm is sensitive to the initial values of parameters such as *ssthresh* (the slow start threshold). If this is too large, the source ramps up its window exponentially to an unsustainable value, causing multiple packet losses. Since the fast retransmit algorithm recovers well only from a single loss, overall performance is substantially degraded. Recent research has addressed this issue by coming up with heuristics to estimate these parameters from the first few packets sent on a connection.

Finally, a TCP endpoint underutilizes the bottleneck link if the bottleneck has so few buffers that a source loses packets when it is still in its exponential increase phase. This corresponds to roughly $R\mu/3$ packets' worth of buffering per connection at the bottleneck. Thus, if TCP is to work well, a network designer must budget for a minimum amount of buffering at each multiplexing point, increasing the cost of the network.

2.4.6 TCP-Vegas

Because of the widespread use of TCP on the Internet, considerable effort has gone into improving it. One attempt is TCP-Vegas. TCP-Vegas attempts to improves TCP's retransmission mechanism, by using a better fast-retransmission heuristic, and TCP's congestion-avoidance strategy. Recall that in TCP-Reno and TCP-Tahoe, the congestion-avoidance mechanism is to increase the transmission-window size over time, decreasing it if there is a loss. The main idea in Vegas congestion avoidance is that a source can compute the *expected* throughput of a connection as:

Expected throughput = Transmission_window_size / Propagation_delay

The denominator is estimated by computing the smallest RTT seen so far, which is likely to be close to the true propagation delay. The numerator is easily available to the algorithm. Thus, over any period, TCP-Vegas can compute the expected throughput along the path. The source can also measure the *actual* throughput by measuring how many acks it receives in a round-trip time. If the actual throughput over the same period is less than the expected throughput, then the source adjusts its window size, and thus its transmission rate.

In practice, the source sends a distinguished packet and records the transmission time of this packet. When the source receives the ack for the packet, it computes the expected and actual throughput (call them *e* and *a)* as described in the previous paragraph. If $e < a$, this means that the Propagation_delay value is too large, and it adjusts this value. If $e \geq a$, then *(e - a) RTT* packets transmitted in the previous RTT are still in the bottleneck buffer. The source does not change its window if this number lies in a range a to b where these are user-selected low and high watermarks. Otherwise, the source adjusts the transmission window so that in the next RTT the expected number of packets in the buffer reaches this value.

Evaluation

TCP-Vegas has been simulated but not extensively tested in the field. Limited simulations showed that it performs better than TCP-Tahoe or -Reno. However, a serious concern is whether TCP-Vegas connections sharing a link with TCP-Reno connections gain better performance at the expense of TCP-Reno. Given the popularity of TCP, incremental improvements in its performance are worth the effort.

2.4.7 NETBLT

NETBLT was the first widely known *rate-based flow-control* protocol. Dynamic-window flowcontrol schemes control the transmission rate by adjusting the size of the transmission window. Thus, the window is used both for error control and flow control. In contrast, in rate-based flow control, the transmission rate does not depend on a flow-control

window. Consequently, losses and retransmissions, which modify the error-control window, do not directly affect the rate at which data is transmitted into the network. This decoupling of error and flow control considerably simplifies both components.

A NETBLT source divides application data into several *buffers*, where each buffer consists of some number of packets. The source and destination negotiate a transfer rate for each buffer. The rate is expressed as a burst size and a burst rate, so that a source needs to control the rate only at the granularity of a burst. Data packets are placed in a transmission queue drained at the negotiated rate. If packets are lost, retransmitted packets are also placed in the same transmission queue so that the source transmission rate is limited to the negotiated rate independent of the loss behavior. This is unlike TCP, where a packet loss causes a decrease in the source transmission rate.

In the original NETBLT scheme, the source and the destination negotiate a transfer rate independent of the capacity available in the network. If the negotiated rate is too high, however, this leads to persistent packet loss. To solve this problem, in the revised version of the protocol, when a source transmits data at a rate r, it explicitly informs the receiver about this rate, and the receiver measures the rate r' at which it receives data. If $r' < r$, then the largest rate the network can support must be smaller than r. Thus, the source multiplicatively reduces its rate to $\beta.r$ where $\beta < 1$. On the other hand, if $r' = r$, the source additively increases its rate to $r + \alpha$.

Evaluation

NETBLT's main contribution to flow control is in its separation of flow and error control. The original design ignored bottlenecks in the network, and the revised algorithm adapts only slowly to changing capacity in the bottleneck, because a source takes control decisions only once for each buffer's worth of packets. Thus, the algorithm tends to either overflow or underflow the bottleneck buffer. Nevertheless, ratebased flow control, an idea first embodied in NETBLT, is a rich and interesting area for research. When carefully implemented, rate-based flow control has the potential to provide a complete solution to the flow-control problem.

2.4.8 Packet-pair

Packet-pair flow control improves on NETBLT's performance by better estimating the bottleneck capacity in the network. Moreover, it *predicts* the future service rate in the network and *corrects* for incorrect past predictions. These allow it to maintain a certain number of packets in the bottleneck queue precisely (its *setpoint)* despite variations in the bottleneck service rate. Unlike other flow-control schemes, packet-pair explicitly assumes that all the bottlenecks in the networks serve packets in round-robin order. Thus, when two packets belonging to the same connection enter a server back-to-back, an interval that is inversely proportional to the connection's service rate at the bottleneck separates them when they leave (Figure B4T80). Note that the separation is largest at the bottleneck server. Thus, if the receiver measures the packet separation (or if the source measures the acknowledgment separation), the receiver (or source) can directly determine the bottleneck service rate. A packet-pair source sends *all* packets as pairs (except if the source only has a single packet to send). Thus, a source can update its estimate of the bottleneck service rate with every pair of acks it receives. If the bottleneck service rate changes with time, the source automatically detects and adapts to the change.

Evaluation

Exhaustive simulations show that packet-pair flow control is stable in a variety of scenarios. For example, Figure B4T82 plots the sequence number versus time for four packet-pair sources that share a common bottleneck server. We see that all four sources make progress at equal rates. Moreover, there are no pauses due to timeouts and retransmissions. Indeed, this is the best performance that one can expect. It has been shown that for many scenarios, packet-pair's performance is nearly as good as an (unimplementable) optimal flow-control scheme that assumes that network elements have infinite buffers. Packet-pair scales well with network size and does not require any explicit information from network elements. Thus, it is a good choice for flow control in networks of round-robin servers.

Unfortunately, packet-pair requires that all intermediate routers support round-robin scheduling mechanisms like those will be described in bloc 4. Most current networks do not provide round-robin-like servers. This limits severely the utility of packet-pair.

2.4.9 Comparison of closed-loop schemes

In this section, we have studied several closed-loop flow-control schemes. Starting with the simplest schemes, such as on-off and stop-and-wait, we examined incrementally more complex schemes. The increasing complexity reflects the deeper understanding of flow control that the past decade of research has produced. With the many choices for flow control, a natural question is to ask which scheme is the "best." Unfortunately, there is no simple answer to this question. Each scheme makes different assumptions about its environment of operation; comparing schemes that make different assumptions is like comparing apples with oranges. The "best" scheme depends on the environment of

operation and the constraints on the system. Given the binding constraints in effect, a designer must choose a flow-control scheme that best matches the situation.

For example, some authors argue that a static window scheme is sufficient if the scheduling discipline is round--robin-like and we can give every router sufficient buffering. Given the current price of memory and technology trends in the cost of memory and the capacity of wide-area trunks, they claim that giving every connection a maximal-sized window (the speed of the fastest trunk in the network multiplied by the largest possible delay) is affordable, and avoids the complexity of dynamic window schemes. However, their proposal requires admission control for every call and round-robin queuing, and switches must have enough memory to ensure an acceptably low blocking rate even at peak call-arrival rates. These complications drive up the cost of the switch compared to, say that of an FCFS datagram switch assumed by TCP-Vegas. If switch cost is a constraint, a reasonable trade-off might be to make the buffer management mechanism more complex, such as by using the credit-based scheme, or the endpoint control more sophisticated, as with TCP-Vegas, because this substantially reduces the memory cost. The point is that we do not really have the "best" solution; we only have a range of solutions that make different trade-offs among complexity, cost, and performance.

We now make some general observations about the trade-offs made by the various schemes:

- Flow control is easier if servers implement fair-share (round-robin like) scheduling. Otherwise, either all the endpoints must be cooperative, or their allocated rate must be policed. Unfortunately, all three options have their own problems: (a) Fair-share scheduling requires perconnection state at the switches, making them more complex. (b) We cannot assume cooperation in a public network, though it is reasonable in a private network. (c) Policing requires more hardware at the network edge, adding to the cost.

- Explicit schemes perform better and are more robust than implicit schemes. However, they not only put a greater traffic load on the network, but also require more complex switches. Unless carefully designed, this increased load may make them less likely than implicit schemes to scale to large networks.

- Hop-by-hop schemes are more responsive to rapid changes in bottleneck utilization, but add complexity to intermediate network elements.

- Separating error control and flow control is a good idea, because changes in the errorcontrol window should not necessarily affect flow control. However, if the only indication of congestion is a lost packet, this is hard to do.

- Rate-based schemes are intrinsically unstable, because sources continue to send even if the rate update packets are lost (unless, as in the EERC scheme, sources reduce their rate automatically lacking feedback). Credit-based schemes are intrinsically stable. However, ratebased schemes typically require less complexity and fewer buffers at the switches.

In a given situation, these trade-offs must be kept in mind when choosing a flow control scheme.

Bloc 5

Architecture de protocoles haute performance

1.0 Protocol Layering

The previous four blocs have presented an overview of the telephone network, Internet, and ATM networks, as well as details about the so-called "lower layers" (physical to transport). This bloc describes *protocol layering*. We will study the notion of a protocol, protocol layering, and why layering is critical for building, operating, and maintaining networks. We will study the seven-layered protocol architecture for data communication developed by the International Organization for Standardization (ISO). We will also discuss some approaches followed by researchers to enhance protocol performance. Finally, we present an enhanced performance protocol architecture.

1.1 Protocols and protocol layering

Consider a customer who walks into a post office to send a letter to a friend abroad. A postal worker accepts the letter and forwards it to a postal worker in the foreign country, who eventually delivers it to the foreign customer. Notice that postal delivery involves an exchange of a letter between at least two sets of parties: first, the customer and her friend, who view the postal network as a black box that accepts letters at one end and delivers it at the other; and second, the two postal workers, who also handle the letter, though only on behalf of their customers (Figure B5T4). The two (or more) parties at the same level communicating with each other are called *peer entities.* In our example, the two customers are peers, as are the two postal workers. A *protocol is* a set of rules and formats that govern the communication between communicating peers. It allows the peers to agree on a set of valid messages and the meaning of each message. In general, protocols mediate *any* function that requires cooperation between peers.

Example 1.1

Consider two peers who want to exchange a file over a network that corrupts packets, but never loses or reorders them. To do so, they can use a file exchange protocol such as this: The sender sends the file as a series of packets, followed by a number computed by performing some function on each byte of the file. This function, called the *checksum,* can be as simple as the sum of all the bytes in the file. The receiver computes the checksum on the received file. If the checksums match, it sends an *OK* response; otherwise, it sends a *not-OK* response. If the sender gets an *OK* response, it stops; otherwise, it periodically retransmits all the packets in the file until it receives an *OK* response. This set of message exchanges informally describes the file exchange protocol. If the checksum is chosen to detect almost all packet corruptions, and at least one round of file exchange succeeds, this protocol, with high probability, will achieve its goal of reliable file transfer.

This protocol is simple but far from optimal. Even if one bit in one packet is corrupted, the checksum is likely to fail, and the sender must retransmit the entire file. It would be more efficient to retransmit only the corrupted packet. Moreover, we have left out many important details. For example, if the link to the sender and the receiver goes down, the sender will keep sending packets ad *infinitum. A* more insidious problem is that the *OK* and *not-OK* messages themselves are not protected from corruption. Thus, if the network corrupts both a packet and the *not-OK* response, the sender might incorrectly think that the network successfully transferred the file. If a protocol is to be robust, its designer must try to anticipate every eventuality!

An important role of a protocol is to describe the *semantics* of a message, that is, the meaning associated with the bits carried in a message. For example, the file exchange protocol in Example 1.1 must allow the sender and receiver to distinguish between file packets and checksum packets. Thus, the file and checksum packets must be in some agreed-upon format, as must be the values chosen to represent *OK* and *not-OK.* A protocol also specifies the *actions* taken on receipt of a message. For example, on receiving the checksum packet, the receiver must compute the checksum and match it with the one it receives. These agreements about message formats and actions to be taken when receiving a message are a detailed *specification* of the protocol.

Thus far, we have described a protocol as something that enables the solution of a shared problem. One may alternatively think of a protocol as providing a *service,* such as reliable file transfer. In other words, peer entities use a protocol to provide a service to a higher-level entity. The *service interface* to the peer entities hides the details of the protocols used in actually providing the service.

Each shared function (or service) in a communication network is achieved by an associated protocol. Thus, a network

that provides many services must define many protocols. It turns out that some of these protocols are independent of each other, while others are dependent. A common form of dependency is where Protocol A uses Protocol B as a step in its execution. For example, one step in the execution of the file exchange protocol we discussed in Example 1.1 involves transferring a packet to the receiver. This transfer itself may be governed by, say, a data transfer protocol. We call this form of dependency *layering:* the file transfer protocol is layered above the data transfer protocol. The lower layer provides a service used by the higher layer in its execution. This is similar to the way a subroutine in a programming language provides a service to its calling program.

The interface between the lower and the upper layer is called a *service access point* or SAP. The packets exchanged between peer entities in the same layer are called *protocol data units (PDUs).* The packets handed to a layer by an upper layer are called *service data units (SDUs). A* protocol data unit is typically a service data unit with an additional header or trailer that carries information needed by that layer's protocol.

Example 1.2

Consider, again, the customer who hands a letter to a postal worker for delivery. The *service* provided by the postal service is mail delivery service. The *service access point* is the post office. Between customers, the *protocol data unit* is a letter. The *service data unit* for the post office is a letter, and the *protocol data unit is* a mailbag that contains letters to a common destination. Each service data unit or letter has a protocol header or envelope with a destination address, an optional source address, and a tamper-resistant stamp to indicate prepayment of service charges. Suppose the peer customers are using the mail service to negotiate a business deal. Then, the negotiation protocol is *layered* above the mail delivery protocol.

Continuing with the file exchange protocol in Example 1.1, the email protocol can utilize the service provided by the file exchange protocol, so we could layer the email protocol above the file transfer protocol. The three layers, email transfer, file transfer, and data transfer, thus constitute a *protocol stack.* Each layer in the stack uses the layer below it and provides services to the layer above it. A key observation is that once we define the service provided by a layer, we need know nothing further about the details of how the layer actually implements this service. The implementation of a layer in a protocol stack can therefore change without affecting the layers above (or below). This property is fundamental for designing networks that survive technological change. As we will see later in Section 5.3, the same information-hiding can also dramatically reduce system performance.

5.2 The importance of layering

Protocol layering provides three important advantages. First, it allows a complex problem to be broken into smaller, more manageable pieces. Second, because the specification of a layer says nothing about its implementation, the implementation details of a layer are hidden *(abstracted)* from other layers. This allows implementations to change without disturbing the rest of the stack. Third, many upper layers can share the services provided by a lower layer. We look at these three properties in more detail next.

A protocol stack allows a network to provide sophisticated services by composing simpler ones. For example, consider a World Wide Web user who clicks on some highlighted text, thus setting into play activities such as determining the location of the server pointed to by that text, establishing a connection between the computer running the browser and the server, and reliable file transfer. Layering allows us to decompose this complicated task into simpler ones that we can solve independently and in parallel. Since each layer solves only a part of the problem, it is easier to write and debug than a monolithic approach to solving the whole problem.

The second advantage of layering is separation of implementation and specification. As we saw earlier, a layer is specified by the services it provides. If the service interface remains unchanged, we can therefore replace the implementation of a layer without affecting the other layers in the stack. For example, the longdistance component of the telephone network has migrated from copper wire to fiber optic cables at the physical layer without affecting customers' expectations of what telephone service means. This is because the layered telephone stack hides the physical medium used for carrying voice samples from the layers above. As technology improves, we can replace lower layers at will without risking the investment made at higher layers in the protocol stack, and vice versa.

Finally, layering allows us to reuse functionality. A lower layer implements common functionality once, which can then be shared by many upper layers. A good example of this is evident in the evolution of networking in the Microsoft Windows operating system. Early versions of the system had little networking support. Thus, every application program written provided its own protocol stack bundled into the application. Over time, application developers converged on the *Winsock* standard, which implements the Internet protocol stack in a shared library module, offering a standard interface to all application programs. Newer application programs layered over Winsock therefore do not need to bundle in networking.

5.3 Problems with layering

Layering is a form of *information hiding*. A lower layer presents only a service interface to an upper layer, hiding the details of how it provides the service. If the correct operation of a layer were to depend on knowing implementation details of another layer, changes in one layer could result in a change to every other layer, an expensive and complex proposition. Thus, network engineers are wary of *layering violations*, that is, a situation where a layer uses knowledge of the implementation details of another layer in its own operation.

Though information-hiding has its benefits, it can sometimes lead to poor performance. To avoid this penalty, in situations where an upper layer can optimize its actions by knowing what a lower layer is doing, we can reveal information that would normally be hidden behind a layer boundary. For example, consider a flow control protocol that is responsible for throttling a source when it thinks that the network is overloaded. A widely used heuristic to discover an overload is to assume that overloads are correlated with packet loss. The protocol, therefore, could throttle the source sending rate when it detects a packet loss. (This is the heuristic used in the Internet's Transmission Control Protocol, which we studied in bloc 4). Suppose, first, that the flow control protocol is layered above a protocol that is actually responsible for data transfer. Then, the flow control protocol does not know how packets are transferred across the network. Next, suppose that the end-system is connected to the network over a lossy wireless link so that packet losses happen mostly because of link errors, rather than network overload. Unfortunately, even in this situation, the flow control protocol thinks that the network is congested, and throttles a source even when there is no need to do so. The information that a packet is lost on the link, which is available to the lower (data transfer) layer, is hidden from the higher (flow control) layer, which results in inefficient flow control. If the lower layer were to inform the upper layer about packet loss on a link, the flow control layer could distinguish between the link and congestive losses and could do a better job. This, however, violates layering, because the flow control layer now knows about the details of data transfer over its local link. So, if the data transfer layer changes, perhaps because the end-system is using a different link technology, the flow control layer, which ought to be independent of the link technology, must also change. Here, we see a tension between informationhiding on one hand and performance on the other. The art of protocol stack design is to leak enough information between layers to allow efficient performance, but not so much that it is hard to change the implementation of a layer. Choosing this balance wisely is the hallmark of good design.

5.4 ISO OSI reference model

All connected systems in a network must agree not only on what each layer does, but also on the protocols that are required to provide these services. We say that a set of protocols is *open* if protocol details are publicly available, and changes to the set are managed by an organization whose membership and transactions are open to the public. A system that implements open protocol standards is called an *open system*. The idea is that any vendor who has the specifications for an open standard can build a standards-compliant open system. This promotes competition, while still ensuring interoperability.

The Open System Interconnect (OSI) reference model developed by the International Organization for Standards (ISO) is the international standard that describes a particular choice of layering, and a particular choice of protocols that carry out the services at each layer. Although OSI protocols are far from common in current networks, the OSI reference model is a good way to think of the functions provided by computer networks, and the model maps well to the protocol layering in common use. Thus, it is worthwhile to understand ISO's seven-layer OSI stack.

ISO distinguishes among three things that look identical at first glance: the *reference model*, the *service architecture*, and the *protocol architecture*.

- The reference model formally defines what is meant by a layer, a service, a service access point, name, etc. These concepts provide the building blocks for defining a seven-layer model for communication.

- The service architecture describes the services provided by each layer. As we saw earlier, each layer provides a service access point to the layer above and provides services using the services of a layer below. The service architecture tells us what services each layer provides and the details of its service access point interface. It does not, however, specify the protocols used to implement these services.

- Finally, the protocol architecture is the set of protocols that implement the service architecture. It is possible for a network to be OSI-service compliant without being interoperable with other OSI-service-compliant networks, since the two networks may implement the same service using different protocols.

Following popular usage, we will not distinguish among the reference model, the service architecture, and the protocol architecture. We will first look at the layering prescribed by the reference model, then look at the services provided by each layer.

5.5 The seven layers

Figure B5T17 shows the seven layers of the ISO OSI reference model. Note that the end-systems contain all seven layers, but intermediate systems, such as switches and routers, implement only the lowest three layers. This is because the third (network) layer provides the abstraction of end-to-end data transfer to the layers above. Thus, at the transport layer and above, the peer layers at an endpoint can talk directly to their peers at the remote end-system without having to worry about intermediate systems. This is like the two postal customers in Example 1.1, who can mail each other letters without worrying about how the letters make their way through the postal system. At the level of the peer customers, protocol actions happen only at the endpoints. Intermediate systems, which provide only network-layer functionality (the equivalent of sorting and distributing mail), need not implement higher layers of the protocol stack (such as those between postal customers).

We will use two running examples to illustrate the operation of the seven-layer protocol stack. The first example continues with Example 1.1, which involves two customers using the postal network to exchange letters. In the second example, we discuss the Internet protocol stack. Taken together, these examples develop some intuitions about the function of each layer, and the reasoning behind ISO's service architecture.

5.5.1 Physical layer

The physical layer is the lowest layer in the OSI stack. It is responsible for moving information between two systems connected by a single physical link. Physical-layer protocols describe such things as the coding scheme used to represent a bit on a communication link, connector shape and size, and bit-level synchronization. The physical layer provides the abstraction of bit transport, independent of the link technology. It is not aware of the notion of a packet or a multiplexing frame. Some details about the physical layer coding schemes were provided in bloc 2, section 1.

Example 1.3

(a) In the postal network, the physical layer provides the technology for transporting letters. These include mail vans, bicycles, postal workers carrying mailbags, and airplanes. The datalink layer hands the physical layer a letter that it expects will eventually appear (with a certain probability) at the other end of a "link."

(b) In the Internet, the physical layer provides the technology for transporting bits. These include coaxial cable transmitters and receivers (in local area networks), satellite transponders and base stations, and optical fiber links with associated optical transmitters and receivers. The datalink layer hands the physical layer a bit that it expects will eventually appear (with a certain probability) at the other end of the link. With a broadcast medium, such as a coaxial cable or a radio link, the same bit appears at multiple endpoints.

5.5.2 Datalink layer

Consider a local area network (LAN) where a coaxial cable links several computers together. The physical layer ensures that a bit placed on the cable by any end-system can be received by all other end-systems. However, for a *packet to* be transferred over the LAN, an end-system must distinguish between an "idle" bit pattern, which occupies the cable when no one is transmitting, and "data" bits, which are parts of a packet. It can do so, for example, if every packet is preceded and followed by a special bit pattern, such as "01010101," which each end-system monitors. If these bit patterns or markers do not occur in the data stream, they frame the beginning and end of a packet. (If these patterns do occur in the data stream, then the transmitting datalink layer replaces them with special markers that are converted to the original bit pattern at the receiver). We call the insertion of these types of markers in the bit stream framing. One of the important duties of a datalink layer is to frame packets.

For point-to-point communication over a broadcast LAN (such as the coaxial-cable link just described), all but the destination of a packet should refrain from receiving it.

We can achieve this by associating each end-system with a unique *datalink-layer address* and requiring end-systems to only receive packets addressed to them. Moreover, because multiple end-systems share a common medium, we also need some way to arbitrate access to the medium. The datalink layer's *medium access control (MAC)* sublayer provides these two functions.

Some datalink layers provide not just framing and point-to-point communication, but also the ability to retransmit packets corrupted on the link (error control), and to pace the rate at which packets are placed on a link (flow control). These functions are considered part of the *logical link control* sub-layer of the datalink layer, which is layered above the medium access control sub-layer.

Because the datalink layer is very dependent on the nature of the physical medium, each physical medium is usually associated with its own datalink layer protocol. In most systems, a single hostadaptor card provides both the datalink layer and the physical layer. For example, a commercially available Ethernet card provides not only the transmitter and receiver for placing and receiving bits on a coaxial cable, but also the ability to frame packets and the medium access control to arbitrate access to the medium. The network layer simply hands a packet with a particular destination address to the card, expecting that it will eventually appear at the other end of the link. The host-adaptor card may also flag errors in received packets.

Example 1.4

(a) Consider the actions at a post office when it receives a letter. At the end of every collection period, a postal worker sorts mail according to whether it is local or remote. If it is local, it is distributed locally. Otherwise, all remote mail is placed in a mail bag and sent to the central post office. The mail bag "frames" the letters, so that the central post office can distinguish between incoming letters from each smaller post office. Moreover, the address on the mail bag allows it to be delivered to the correct central post office (if there is more than one in the area). Thus, a mail bag is the equivalent of a frame. The medium access control is the set of traffic rules and regulations a mail truck must obey in getting to its destination.

(b) The Internet supports a variety of datalink-layer protocols, of which the most common at the time of writing is Ethernet. Ethernet defines unique bit sequences to demarcate data bits in frames. Its MAC sub-layer supports point-to-point communication with 6-byte globally unique datalink addresses to identify end-systems. Before placing a frame on the link, the MAC layer at the transmitter checks to see if the medium is already in use (we call this carrier sensing). Despite this check, two end-systems may still launch frames nearly simultaneously, and these collisions are resolved using the carrier-sense multiple access/collision detect (CSMA/CD) protocol. Besides Ethernet, other common data links in the Internet include Fiber Distributed Data Interface (FDDI), Synchronous Optical Network (SONET), and High-level Data Link Control (HDLC).

5.5.3 Network layer

The main function of the network layer is to logically concatenate a set of links to form the abstraction of an end-to-end link. It allows an end-system to communicate with any other end-system by computing a route between them and using the services of the datalink layer to carry packets on this route. The network layer therefore allows a higher layer to communicate directly with its peer, though this peer may be reached only by traversing many intermediate systems. The network layer also hides specificities of the datalink layer from higher layers. For example, given a link that limits the largest allowed packet size, the network layer can fragment data handed to it by the transport layer when it enters such a link *(segmentation)* and reassemble it at the destination so that the next-higher (transport) layer is not aware of this limitation. Finally, to uniquely identify an end-system, the network layer provides unique network-wide addresses.

Unlike datalink-layer addresses, network-layer addresses correspond to network topology, so that they can be *aggregated* in routing tables. We studied this point in detail in bloc 3.

The network layer is found both at end-systems and at intermediate systems. At an end-system, its task is limited primarily to hiding details of the datalink layer-for example, with segmentation and reassembly. It may also provide some error detection. At intermediate systems, and at end-systems connected to multiple routers, besides these functions, it participates in a routing protocol to discover the next hop for every possible destination in the network. In a datagram network, the network layer is responsible for forwarding packets, scheduling their transmission order, and, if necessary, dropping excess packets. Thus, it plays a critical role in providing end-to-end delay and lossrate guarantees.

We organize the network layer differently in datagram and connection-oriented networks. In datagram networks, the network layer provides both routing and data forwarding. In contrast, in a connectionoriented network, we distinguish between the *data plane* and the *control plane* . The data plane is the set of protocols involved in transferring data. The control plane contains protocols responsible for network and connection control. Thus, routing, call-establishment, and call-teardown protocols are in the control plane of a connection-oriented network, whereas data-forwarding protocols are in the data plane. In such networks, layers above the network layer may also be partitioned between the data and control planes. A good rule of thumb to distinguish between data and control plane actions is that if a function involves touching every packet. then it is in the data plane; otherwise, it is in the control plane.

Example 1.5

(a) The postal network's service interface to the outside world is at the network layer. This layer sets up internal routing tables so that postal workers can identify a source-to-destination path for every letter. Postal workers forward letters along this path, using the services of the datalink layer, to provide the abstraction of a single end-to-end logical link. Thus, like a datagram network, the postal network provides both routing and forwarding. Unlike data networks, however, postal network topology changes very slowly over time. Therefore, routing is essentially static. In contrast, a

computer network uses dynamic routing to deal with rapid changes in network topology, for example, due to a power outage.

Interestingly, the postal network provides multiple qualities of service. Priority-mail letter services allow a customer to pay more to ensure speedy delivery. Similarly, bulk mailers pay less if they accept a "best-effort" delivery where they have to presort letters and put up with possibly large delays. These quality-of-service guarantees require *every* intermediate system in the postal network to be aware of customer requirements and act in according to these requirements.

(b) The Internet's *Internet Protocol (IP)* is found in all Internet-compatible routers and end-systems. It presents an end-system with the abstraction of an end-to-end logical link by providing routing, packet-forwarding, and segmentation and reassembly. End-systems that want to communicate with a remote destination need only format a packet with an IP header and the destination's IP address, and IP will carry it there, wherever in the world it may be!

The beauty of IP is that we can layer it over practically any datalink layer technology, because it makes very few assumptions about the datalink layer. This makes it easy to extend the Internet over new datalink technologies as they arise. The price to pay for this simplicity is that IP provides only a single "best-effort" quality of service. Although the IP header has a field that describes the type-of-service that a packet desires, in practice, this field is rarely heeded, since IP does not require the datalink layer to distinguish between different service qualities. For example, if a high-priority IP packet crosses a shared medium, such as Ethernet, the packet may still suffer long delays as it waits for lower-priority packets from other attached systems to use the medium. Since IP does not require Ethernet to provide different qualities of service, IP in turn cannot provide these to upper layers.

5.5.4 Transport layer

The transport layer is to the network layer as the datalink layer is to the physical layer. The network layer usually provides a "raw" end-to-end packet transport service. The transport layer uses this service to create the abstraction of an error-controlled, and flow-controlled, end-to-end link. By error-controlled, we mean that the transport layer guarantees that, with very high probability, a message will reach the destination despite packet loss, corruption, and duplication. Transport-layer protocols deal with loss by retransmitting lost packets; with corruption by detecting, discarding, and retransmitting corrupted packets; and with duplication by detecting and discarding duplicate packets.

Transport layers also provide flow *control*, that is, matching the transmission rate of a source to the rate currently sustainable on the path to the destination. For example, consider a personal computer (PC) communicating with a supercomputer. Usually, the supercomputer can pump out data far faster than the PC can absorb it. Flow control forces the supercomputer to pace its transmission to match the PC. It can do so either explicitly, where the PC informs the supercomputer of its maximum sustainable transfer rate, or implicitly, where the supercomputer uses some heuristics to estimate the PC's service rate.

The third main function of a transport layer is to multiplex multiple applications to the same end-to-end connection. The network layer typically routes data to a particular end-system, without distinguishing among the applications supported at the end-system. The transport layer adds an application-specific identifier (usually called a *port number)* to a packet, so that the receiving end-system can hand an incoming packet to the correct application.

Not all transport layers implement the same set of services. Some lightweight transport layers provide a cursory (but easily computed) error-detection algorithm and multiplexing, but no flow control or retransmission. Others provide extensive error checking, sophisticated flow control, and retransmissions. An application must choose the transport mechanisms best suited to its needs.

Example 1.6

(a) The postal system does not have a transport layer. This is implemented, if at all, by postal service customers. Suppose customers Alice and Bob want to achieve *error-free service,* with Alice sending a steady stream of letters to Bob, say, at the rate of a *letter a* day. The postal system does not corrupt letters, but it can reorder or drop them. To protect against *these, A*lice may write the date of posting on the letter (that is, in the letter's header). Bob can use these dates as *sequence numbers* to *detect and* correct letter reordering. Moreover, if Bob does not receive a letter posted on a particular day for a "long" time, he can assume that the letter is lost. He can then ask Alice to retransmit the lost *letter. In* this way, Alice and Bob can use sequence numbers, timeouts, and retransmissions to recover from letter loss and reordering. If Alice and Bob *were clerks* in a large firm, who carried out mail exchange on behalf of their bosses, then from the perspective of their bosses, the link *between A*lice and Bob is reliable and in-order.

(b) The two popular transport-layer protocols in the Internet are the *User Datagram Protocol (UDP)* and the *Transmission Control Protocol (TCP). UDP* provides multiplexing, but not error recovery or flow control. It is typically used by applications that can *either live* with packet loss (such as audio or video applications), or those that provide

their own retransmissions (such as the Network File System (NFS) protocol).

TCP provides error detection and correction, as well as multiplexing and flow control. It provides higher layers with the abstraction of an error-free, reliable, in-order stream of bits. Most Internet applications, such as the World Wide Web and file transfer, use TCP. TCP achieves error control using sequence numbers, timeouts, and retransmissions as outlined in part (a) of this example. It also does flow control by dynamically estimating the capacity available on the path from the source to the destination.

5.5.5 Session layer (it's layer 5!)

The transport layer usually provides a degree of abstraction sufficient for most application programs. In some networks, however, an additional session layer adds the abstraction of full-duplex service (if the transport layer does not already provide this), expedited data delivery, and session synchronization.

- Some transport layers provide only unidirectional or *simplex* transmission. In such networks, a session layer can manage two transport connections to provide the abstraction of bidirectional *or full-duplex service.*
- Expedited data delivery allows some messages to skip ahead of others in the session-layer message queue (using a separate, low-delay transport connection).
- Synchronization allows endpoints to place marks at specific points during data transfer and to roll back to a prespecified mark. This is useful for applications that need *atomic* data transfer, that is, either all or none of the data in a set should be transferred. Rollbacks allow a session layer to cleanly abort transfer of a data set.

Example 1.7

(a) Consider a firm that has a clerk who handles only incoming mail ("receiving") and a clerk who handles only outgoing mail ("shipping"). Each clerk, therefore, implements one end of the transport layer. Suppose they report to a chief clerk who not only accepts letters for transmission from the firm's employees, but also delivers incoming mail to them. The chief clerk, therefore, plays the role of a session layer, managing two simplex connections to present the abstraction of a duplex connection. The chief clerk may also expedite some letters (using courier service instead of regular service). Finally, if the clerk is given a set of letters such that either all or none must be delivered (such as invitations to an important meeting), she can arrange with her peer to discard incomplete sets to provide this abstraction.

(b) The Internet does not have a standard session layer protocol. TCP already provides duplex and expedited data delivery, and session rollbacks are part of the application, if necessary.

5.5.6 Presentation layer

Unlike the other layers, which are concerned mostly with *meta-data* in headers, the presentation layer deals with *data.* The presentation layer hides data representation differences between applications. For example, one machine may store a word with high-order bytes first *(big-endian),* and another with the high-order bytes last *(little-endian).* The presentation layer converts the representations to a network standard so that this difference is hidden from the applications. The presentation layer may also encrypt data, both to authenticate it to the receiving application and to prevent unauthorized parties from accessing it. Like the session layer, the presentation layer is often ad hoc, if present at all.

Example 5.8

(a) In the postal example, assume that the letters are being exchanged between firms in two countries that do not share a common language. The *contents* of the letter therefore must be translated before the recipient can understand it. This translation can be done either at the sending or receiving end. The person doing the translation plays the role of the presentation layer.

(b) The Internet does not support a standard presentation layer, and Internet applications usually incorporate appropriate presentation-layer functionality. The Internet, however, does support a standard byte-ordering for representing integers (but not floating-point numbers). In a Unix system, the macros *htons* and *htonl* convert a short or long number from host to network order. At the receiving host, the macros *ntohs* and *ntohl* convert them back to the byte order appropriate for the receiving host. Unfortunately, the wide variety in floating-point formats does not allow a similar simple conversion.

5.5.7 Application layer

The application layer is just another name for the set of applications running on an endsystem. The application layer

uses the complex services provided by the lower layers of the protocol stack, and does not provide services to any other layer. Although the ISO has described a standard set of applications, any program that uses the presentation layer's services can be considered a part of the application layer.

Example 1.9

(a) In the postal example, the application layer is the entity in the firm that creates and mails letters. Suppose this were a department that does mass mailings for an automobile company. Let us trace the actions taken when the department is asked to send recall letters because of a defect in a product. The mail department, as part of application layer processing, collects the names and addresses of affected customers, then writes letters to each of them. The job of the presentation layer is to translate letters that are being mailed abroad. Letters, perhaps after translation, are handed to a session-layer mail clerk, who may send some by courier, some by priority mail, and others as bulk-rate mail. Each of these services corresponds to a particular transport clerk, who sends letters over the postal network and waits for an acknowledgment, retransmitting a letter if no acknowledgment is received in a reasonable time. The postal system, which supports the network layer interface, finds a route for each letter and transfers it to its destination. The datalink layer abstracts the details of whether the mail was transferred by truck, airplane, ship, or bicycle. Finally, the physical layer provides the actual transfer of mail.

We see that the protocol stack carefully coordinates many actions to provide a complex service interface to the end user (here, the person ordering the mass mailing). Each layer of the stack adds some services to the layer below it, so that, at the application layer, an application has a rich set of communication choices.

(b) The Internet supports many applications, some of which are so popular that they are sometimes mistaken for the Internet itself! The most widely used application at the time of the writing is the World Wide Web (WWW), which allows an application called a *browser* to retrieve specially formatted files from another application called a *server*. The formatting commands in the file not only allow a browser to display text, graphics, sound, and movies, but also allow the document to contain links to other formatted documents. A user can *navigate* from one document to another by following these links. The basic step in the World Wide Web, which is just file transfer, uses TCP for reliable communication. Some specialized services, such as real-time audio and video retrieval, use UDR All services beyond those provided by TCP are made a part of the browser and server applications themselves.

The Internet's protocol layers, which we have studied in earlier examples, allow a browser to reliably access files from any end-system in the Internet. The files are transferred packet by packet over a route that dynamically compensates for failures, at a rate that automatically compensates for the activity of other users in the network. By building these services layer by layer, we can implement complex applications, such as a WWW browser, with relatively little added effort.

Examples 1.3-1.9 show how layering breaks up the complex task of reliable world wide communication into several sub-problems, and how each protocol layer tackles a different sub-problem. The end user need not worry about how the packets got to the other end, possible traffic congestion, how files are fragmented into packets, or how to deal with physical links. Each layer of the stack provides a clean abstraction to the next higher layer so that the stack as a whole provides a complete service to the application. This is a key benefit of protocol layering.

However, protocol layering was suspected by many researchers as being the cause of bad performance, and a large number of papers addresses this aspect. We present in section 2 an overview of these research activities.

2.0 High Performance Protocol Architecture

The development of high speed networking applications requires improvements to classical data communication protocols (such as TCP) in order to efficiently provide the required services to the applications. In this section, we will first give a rapid presentation of protocol optimization techniques and of some high speed transport protocols developed for specific application needs. We will then present a new protocol architecture based on the "Application Level Framing" (ALF) concept. ALF states that applications send data as a sequence of autonomous "frames", which will be at the same time the unit of transmission, the unit of control and the unit of processing. This allows for efficient design of application specific protocols.

2.1 Introduction

The development of high speed networking applications such as audio and video conferencing, collaborative work, supercomputer visualization, transactional applications and WWW, requires efficient communication protocols. As networks proceed to higher speeds, the performance bottleneck is shifting from the bandwidth of the transmission media to the processing time necessary to execute higher layer protocols. In fact, the existing standard transport protocols (e.g. TCP) were defined in the seventies. At that time, communication lines had low bandwidth and poor quality, and complex protocol functions were necessary to compensate for the transmission errors. The emergence of high speed networks has changed the situation, and these protocols would not be designed in the same way today. One the other hand, the transport protocols such as TCP or TP4 were designed to provide a so called "standard transport service" mainly by performing end-to-end error control for point-to-point connections: this includes detection and correction of packet losses, duplications, mis-ordering and alterations. However, the application environment is changing. New applications with specific communication requirements are being considered. Depending on the application, these requirements may be one or more of the following: (1) high bit rates, (2) low jitter data transfer, (3) simultaneous exchange of multiple data streams with different "type of service" such as audio, video and textual data, (4) reliable multicast data transmission service, (5) low latency transfer for RPC based applications, variable error control, etc. The above requirements imply the necessity to revise the data communication services and protocols in order to fulfill the specific application needs. In fact, applications may "classically" choose either the connection-less or the connection oriented transport services. In both cases, the application needs are expressed in terms of quality of service (QoS) parameters such as e.g. the transit delay or the maximum throughput. However, the applications should be involved in the choice of the control mechanisms and not only in selecting the parameters of a "standard" transport service. Placing most of the burden of network adaptation in the user equipment is in line with the "end to end argument", a key point of the Internet architecture. It contributes in keeping the network simple, and is very often the only way to scale up a complex system. This should allow to build high performance communication modules. Performance here is defined as the efficient use of resources while still meeting the applications' requirements. We will detail later the issues that influence the performance of a communication system. This section presents a survey of several techniques in the area of high performance protocols. The rest of the section is organized as follows. In subsection 2.2, we present the state of the art in high performance protocols: we start by studying the impact of the environment on performance and then we present different performance enhancement techniques ranging from protocol parameter tuning and adaptation algorithms support to the design of special purpose protocols. Section 2.3 discusses architectural design issues and shows why the layered model needs to be re-considered. We also present in this section, the ALF and concept as a foundation of a high performance protocol architecture.

2.2 High performance protocols

Early work on high performance protocols concentrated on the optimization of each layer separately. Concerning the transport protocols, several approaches have been studied such as the work on *enhanced environments* for protocol development, the *tuning* of standard general purpose protocols, including the design of *adaptive* transmission control algorithms, or the development of *specialized* protocols. In fact, the protocol execution environment has a strong impact on the performance of the communication system. The environment overhead is mainly due to interrupt handling, timer management, buffer allocation and process scheduling. Work on enhanced execution environments concentrated on blocking on multiple events, low management overhead timers, good I/O buffer management and better resource scheduling. However, these issues are generally independent of the communication protocol and therefore will not be detailed in this section as we focus only on protocol related issues.

2.2.1 Tuning of standard protocols

The general purpose transport protocols, such as TCP or TP4[25], support complex control mechanisms for connection management, error control and flow control on heterogeneous networks. The advantage is that these protocols may be used by a large number of applications requiring the standard reliable point to point transport service. The price to pay is a limitation of the protocol performance. A careful analysis of these protocols showed three types of issues to be resolved in order to enhance the performance:

- a bad choice of the acknowledgments strategy,
- a bad choice of the time-out values,
- limitations due to the lack of congestion control algorithms.

The *tuning* of these protocols consists of chosing the good values for the parameters and designing adaptive transmission control algorithms.

The acknowledgments

TCP uses positive cumulative acknowledgments indicating the number of the next expected octet. These transport level acknowledgments (ACKs) have a bad impact on performance and their number should be minimized. Clark proposed in RFC813[26] to delay the emission of an ACK in certain cases with the hope of grouping several ACKs. Another modification of TCP was proposed in order to enhance the performance over links with high *bandwidth-delay* product: the use of selective ACKs. In the case of packet loss, the sender should only resend the lost packet and not all the unacknowledged packets. Furthermore, a Negative Acknowledgment scheme (NACK) was proposed to be used in several protocols (e.g. NETBLT, and XTP). A NACK has the semantics of an explicit retransmission request from the destination. This has the advantages of reducing the number of control packets (the NACKs) if there are no packet losses, and transferring the error detection problem to the receiver, thus allowing for more scalability in the case of a multicast transmission.

The timers

Transport protocols use several timers for connection management and error detection (TP4 uses 7 different timers). In addition to the timer management cost, a major problem is to find the "good" time-out values for the timers. This is particularly important for the retransmission timer. The time-out value should satisfy two incompatible goals: a rapid detection of packet losses and a reduction of the number of false alarms due to a delayed packet and not a "real" loss. The optimal value depends on the round trip time (RTT) between the source and the destination which, in turn, depends on several factors such as the network load, the routes taken by the packets and probably the packet size. This imposes that the time-out value be adapted to the RTT variation. The effective retransmission time-out in TCP is a function of the "smoothed" RTT. The smoothing factor filters the transient changes and its value determines the responsiveness of the algorithm to network changes. However, a difficulty may arise in the case of retransmission because of the lack of information whether the received ACK corresponds to the first or the second transmission of the packet. Therefore, the RTT sample corresponding to a retransmitted packet should not be taken into account when computing the smoothed RTT. In addition, the RTT variance should be taken into account to compute the retransmission time-out. Even with these tunings, it is still possible to have unnecessary retransmissions due e.g. to a packet blocked on the local network interface, a lost ACK or a bad RTT estimate. The main problem comes from the fact that timers are *local* means to estimate external events. Some researchers proposed to avoid the use of transport level timers for transmission control.

2.2.2 Adaptive Congestion control

Transport level congestion control ensures that network resources are shared efficiently and network congestion is avoided. In fact, window based flow control algorithms tend to use large window values in order to "fill the pipe" in the case of networks with high bandwidth delay product. However, if several transport connections with large windows share a "low bandwidth" link, packet queuing delays may be observed in the router accessing the link, resulting in unuseful retransmissions. This positive feedback may cause a congestion if specific control algorithms are not applied. Van Jacobson proposed the "slow start" algorithm that has been implemented in TCP since 1989. We described this algorithm in bloc 4. The implementation of this algorithm within the TCP version resulted in enhanced performance for the protocol.

---

[25] TP4 is mentioned for historical reasons.

[26] RFCs can be obtained from the IETF web page: http://www.ietf.org

2.2.3 Special purpose protocols

Some of the early work in the domain of enhanced transport *service* proposed the design of light weight special purpose protocols for specific application needs (e.g. NETBLT for bulk data transfer and VMTP for transactional applications). NETBLT or NETwork Block Transfer was described in bloc 4. VMTP or Versatile Message Transfer Protocol is another special purpose protocol that was designed for Intra-System communication (e.g. RPC for file pages access, etc). The nice feature about this protocol was the support of multicast transmission at the transport level. In order to speed up the request/response time, no transport level connection is established. Stable addresses are used instead to provide a sort of semi-permanent connection. Application level replies may be used to acknowledge the reception of the requests. This protocol was one of the first protocols in conformance with the Application Level Framing concept. However, the protocol could only be used in a local network environment, and even in this case the performance was not very good (4.5 Mbps over Ethernet). XTP or Xpress Transfer Protocol was initially designed to be implemented on specialized hardware with execution efficiency as inherent part of the design process. Therefore, many syntactical and algorithmic choices of the protocol are oriented to facilitate high speed operation over low error rate networks. The hardware circuit implementation project was discarded in 1992, but the protocol survives within the "XTP-Forum". XTP 4.0 revision was released in March 1995. In the previous releases, (3.6 and before) XTP had been a "transfer" protocol integrating layers 3 and 4 processing. We concentrate the description of the XTP protocol on this transfer layer architecture. Integrating layers 3 and 4 reduces the packet processing overhead and enables several performance optimizations. Among these optimizations, (i) the possibility to calculate the checksum on the fly by using packet headers and trailers, (ii) a lightweight checksum algorithm that can be computed at high speed in software, (iii) fixed length fields for fast access to the header, (iv) fast connection establishment and release based on a "1-way handshake" mechanism and (v) fast de-multiplexing of the incoming packets by the exchange of local access keys. In addition, the XTP protocol provided several functional enhancement (e.g. the support of ``pseudo" reliable multicast, rate control with the participation of intermediate routers, priority support and variable error control controlled by the sender and adapted to the application needs). However, there was no pre-defined XTP service. XTP provided a programmable toolkit (a set of mechanisms and not predefined strategies). The selection of the required functionalities is done by the application. The XTP protocol designers proposed new ideas (e.g. the layers integration) incompatible with the OSI "reference" model. The ideas represented a step toward integration. However, these ideas were discarded in the 4.0 version and XTP became a "normal" protocol.

The "special purpose protocols" approach has a major limitation: the diversity of the applications increases the complexity of the transport by the support of several protocols. Each application would then choose a protocol corresponding to its specific needs.

2.3 A new protocol architecture

The approaches presented in the previous section are not adequate to provide high performance communication systems for multimedia applications for two main reasons. The first one is that these protocols do not include the upper layers data processing overhead in the transmission control loop. This overhead seems to be "the" effective bottleneck for these application (e.g. video coding or decoding for a video conferencing application, or data presentation for a data base replication). The second reason is that multimedia applications have specific communication needs that do not correspond to the standard general purpose or the proposed special purpose protocols. The performance of workstations has increased with the advent of modern RISC architectures but not at the same pace as the network bandwidth during past years. Furthermore, access to primary memory is relatively costly compared to cache and registers and the discrepancy between the processor and memory performance is expected to get worse. These observations lead some researchers to reconsider the protocol architectural model and not only mechanisms within a specific layer or even the definition of a new protocol. The goal of the revision is to simplify the design of communication systems that take into account (i) the status of the network and the environment and (2) the applications needs. In this section, we discuss the issues concerning the design of this new protocol architecture.

2.3.1 Layering considered harmful

The OSI reference model adopted as a standard in 1980, was designed in the seventies. The layer organization reflected the hardware architecture of those days depicted in Figure (B5T32). The artificial separation of concerns in entities called layers is partly due to the hardware architecture of the 70's. The service/protocol concept derives historically from the model presenting interfaces between different service providers. There would be a link, network, transport and session provider, perhaps all of which would have different potential vendors.

The physical and data link layer corresponded to mechanisms implemented by the modem and the driver. The network and transport layers were generally implemented in front-ends, and the session layer corresponded to the control of the dialogue via the channel between the mainframe (where the application hosted) and the front-end. Data presentation was integrated in the application (e.g. X.409). With the advent of workstations (see Figure B5T33), the architecture of the implementations have changed, specially for the upper layers (transport and above). The transport layer is usually

implemented in the kernel and accessible to the application through the socket interface. The application is run at the user level and integrates specific parts of the presentation layer. The session layer presents, however, a problem (see Figure B5T35). There is no need for such a layer in this architecture. In other words, the data transfer synchronization needs are better known by the application. It is therefore suitable to let the application control its synchronization needs rather than to delegate this control to the session layer. There is another problem posed by this layer in case it is implemented between the presentation and the transport: this would impose an asynchronous data exchange between these layers (which is known to degrade performance). This implies two data copies to/from the memory before data is transmitted on the network. These operations are relatively costly on RISC workstations. The session control should therefore be integrated within the application.

Even in a single vendor software implementation, one could accuse the layered model (TCP/IP or OSI) of causing inefficiency. In fact, the operations of multiplexing and segmentation both hide vital information that lower layers need to optimize their performance. In fact, the application should be able to exchange control information with the transport. This means that the transport should be aware of the application (just like VTMP replies served as ACKs for the requests). This allows to avoid bad decisions that may be taken by the protocol (e.g. send an ACK when there is a reply outstanding, or close the window or retransmit the packet after a packet loss during a video transfer). The two key issues for the performance enhancement are reducing the asynchrony and involving the application in the transmission control.

Some researchers made the observation that if the performance of the communication system is to be enhanced the slowest part in the chain should be used in the best way. Many researchers agree that the data manipulation at the higher level is the bottleneck (presentation decoding, encryption, etc). However, transport level resequencing delays the processing of the PDUs received out of sequence until the lost or delayed PDU is received. An efficient utilization of the slowest part of the chain requires therefore that the application be able to process out of sequence packets.

2.3.2 Application Level Framing

Application Level Framing (ALF) was proposed as a key architectural principle for the design of a new generation of protocols. ALF is in fact the natural result of advanced networking experiments, which showed the need for the following rules:

1. Efficient transmission can only be achieved if the unit of error and flow control is exactly equal to the unit of transmission. This rule is not satisfied for transport protocols over ATM networks. In fact, the control unit (the transport PDU) is not equal to the transmission unit (the ATM cell). Obvious penalties for violating this rule are unnecessary large retransmissions in case of errors and inefficient memory usage.

2. A key idea is that the unit of transmission should also be the unit of processing. Otherwise, large queues will build up in front of the receiving processes and eventually slow down the application.

3. Experience with multimedia services, showed that adaptive applications (see figure B5T53) are much easier to develop if the unit of processing is also the unit of control. Violating this rule means a separation between the application and the transmission control which may result in bad choices taken by the transport.

According to the ALF principle, applications send their data in autonomous "frames" (or Application Data Units (ADUs)) meaningful to the application. It is also desirable that the presentation and transport layers preserve the frame boundaries as they process the data. In fact, this is in line with the widespread view that multiplexing of application data streams should only be done once in the protocol suite. The sending and receiving application should define what data goes in an ADU so that the ADUs can be processed out of order. The ADU will be considered as the unit of "data manipulation", which will simplify the processing. For example, an image server will send messages corresponding to well identified parts of the picture. When a receiver receives such a frame, it can immediately decompress the data and ``paint" the corresponding pixels on the screen, thus minimizing response times. The ADU size should not exceed the minimum MTU (Maximum Transmission Unit) of all the subnetworks traversed by the ADU in order to avoid segmentation.

ALF was proposed in 1990 and extensively studied by researchers. Almost all "research" based networking application (early audio and video conferencing tools, white boards, virtual environments are based on this concept).

Several experiments were made to evaluate the advantages and disadvantages of ALF. These experiments show that efficient implementation of distributed applications may benefit from new architectural considerations such as ALF. Performance improvements were shown in some experiments. However, this was under high packet error rate condition. In fact, if there were no packet losses in the network, ALF complexity would impose a performance penalty. On the other hand, ALF implies that communication systems are "tailored" for each application. This requires an automated approach for protocol code generation, which will allow to "easily" generate a large panoply of communication systems tailored to application needs. Therefore, a protocol compiler that generates efficient code is a

step towards the support of new ALF based applications. A protocol compiler takes as input an abstract specification of a protocol and generates an implementation of that protocol. Protocol compilers usually produce inefficient code both in terms of speed and code size.

2.4 Summary

After a survey of several approaches for enhanced protocols performance, this section describes a high performance protocol architecture for efficient implementation of multimedia applications. This protocol architecture is based on ALF, which is a design principle allowing efficient processing of the application data units. Experiments with Application Level Framing (ALF) showed performance gain in the case of a heterogeneous Internet. However, the adaptive applications approach (with ALF) alone is not sufficient and a new network service should be supported by the network. What should the network provide as a support for that? There is a need to re-think what should be *in* the high speed networks of the future.

Bloc 7

Ordonnancement

1.0 Mécanismes d'ordonnancement dans les routeurs

1.1 Introduction

Computer networks allow users to share (or *multiplex)* resources such as printers, file systems, long-distance trunks, and sites on the World Wide Web. Sharing, however, automatically introduces the problem *of contention* for the shared resource.

Example 1.1

Consider the set of queries made to a server hosting a World Wide Web search engine. Each query represents a service request from a user contending for access to the shared resource, that is, the search engine. Assume that the server can serve only one request at a time, so that requests that arrive when a server is busy must wait. A busy server holds an incoming search request in a *service queue* and eventually selects it for service. We define the *queuing delay* of a request as the time between its arrival and eventual service. The server allocates different mean queuing delays to different requests by its choice of service order. For example, if the server always serves requests from a particular user A as soon as they arrive, then A's requests receive a lower mean queuing delay than requests from other users.

Given a set of resource requests in the service queue, a server uses a *scheduling discipline* to decide which request to serve next. Scheduling disciplines are important because they are the key to fairly sharing network resources and to providing performance-critical applications, such as telephony and interactive multi-participant games, with performance guarantees. In this bloc we will study several scheduling techniques, comparing their relative merits in providing fairness and performance guarantees.

A scheduling discipline actually has two orthogonal components. First, it decides the order in which requests are serviced. Second, it manages the service queue of requests awaiting service. To understand this further, let us return to the situation in Example 1.1. If, because of a statistical fluctuation, requests arrive at the search engine faster than it can serve them, some requests must wait in the service queue, taking up storage space. If storage is limited, and requests keep coming in faster than they can be served, then the server must eventually drop some requests. Which ones should the server drop? In the same way as a scheduling discipline allocates different *delays* to different users by its choice of service order, it allocates different *loss rates* to different users by its choice of which requests to drop. Continuing with Example 1.1, if the server preferentially drops service requests from user B whenever the service queue overflows, B has a higher loss rate than A. Thus, a scheduling discipline allocates different service qualities to different users by its choice of service order and by its choice of which requests to drop.

Though a network must schedule access to *every* multiplexed resource, in this bloc we will restrict our attention to the two most commonly scheduled resources: the bandwidth on a link, and buffers at a multiplexing point. Techniques for scheduling these resources are applicable, with little change, to other multiplexed resources.

Scheduling, like error control and flow control, can be done at one of several layers of a protocol stack. In the literature, scheduling disciplines are usually studied for the output queue of a switch and are placed in the network layer. However, as we saw in the Web server application, we also need scheduling at the application layer. In general, any layer dealing with a multiplexed resource must deal with scheduling.

We note in passing that scheduling is important only when statistical fluctuations in the input traffic result in queuing at a multiplexing point. In circuit-switched networks, source traffic is smooth and without significant fluctuations. Therefore, scheduling is not an important problem in such networks, and we will study only packet-switched networks (such as the Internet and ATM networks) in the rest of the bloc.

1.1.1 Why do we need a scheduling discipline?

Before we study scheduling disciplines in detail, we make a small detour to understand why we need a nontrivial scheduling discipline in the first place. Ultimately, the motivation comes down to our guesses about the performance requirements of future networked applications which we summarize here.

Most experts agree that future networks will carry at least two types of applications. Some applications (which are already common on the Internet) are relatively insensitive to the performance they receive from the network. They are

happy to accept whatever performance the network gives them. For example, a file transfer application would prefer to have an high bandwidth and very low end-to-end delay. On the other hand, it works correctly, though with degraded performance, as the available bandwidth decreases and the available end-to-end delay increases. In other words. the performance requirements of such applications are *elastic:* they can *adapt* to the resources available. Such applications are called *best-effort* applications, because the network promises them only to attempt to deliver their packets, without guaranteeing them any particular performance bound. Note that *best-effort* service, which is the service provided for best-effort applications, does not require the network to reserve resources for a connection.

Besides best-effort applications, we expect future networks to carry traffic from applications that do require a bound on performance. For example, an application that carries voice as a 64-Kbps stream becomes nearly unusable if the network provides less than 64 Kbps on the end-to-end path[27]. Moreover, if the application is two-way and interactive, human ergonomic constraints require the round-trip delay to be smaller than around 150 ms. If the network wants to support a perceptually "good" two-way voice application, it must guarantee, besides a bandwidth of 64 kbps, a round-trip delay of around 150 ms. Thus, this application, and other applications of its kind, demand a *guarantee* of service quality from the network. We call these applications *guaranteed-service* applications. Guaranteed-service applications require the network to reserve resources on their behalf[28].

The reason these application characteristics affect scheduling is that the performance received by a connection depends principally on the scheduling discipline present at each multiplexed server along the connection's path from a source to a destination. These servers are typically the ones scheduling packets at each output link at a switch or router. Recall that a switch queues resource requests, represented by packets ready for transmission, in a per-link output queue. At each output queue, a server uses a scheduling discipline to choose which ready packet to transmit next, and to control access to output queue buffers. The server can allocate different *mean delays* to different connections by its choice of service order. It can allocate different *bandwidths* to connections by serving at least a certain number of packets from a particular connection in a given time interval. Finally, it can allocate different *loss rates* to connections by giving them more or fewer buffers. Thus, to build a network that gives guaranteed-service applications performance guarantees, we require scheduling disciplines that can support per-connection delay, bandwidth, and loss bounds.

Although best-effort applications do not require performance bounds, the partitioning of available bandwidth and buffers to best-effort connections, which is the role of a scheduling discipline, determines how *fair* the network is. Fair resource allocation is an intuitively desirable property of a computer network (see also Section 1.2.2). To build a network that allocates resources fairly among best-effort connections, we must implement scheduling disciplines that support fair resource allocation among best-effort connections at each switch.

In this bloc, we will study the requirements of a scheduling discipline (Section 1.2) and the degrees of freedom available in designing a scheduling discipline (Section 1.3). We then examine some link scheduling disciplines that are suitable for best-effort connections (Section 1.4), and finally others that are suitable for guaranteed-service connections (Section 1.5).

1.2 Requirements

A scheduling discipline must satisfy four sometimes contradictory requirements:

- Ease of implementation (for both guaranteed-service and best-effort connections)
- Fairness and protection (for best-effort connections)
- Performance bounds (for guaranteed-service connections)
- Ease and efficiency of admission control (for guaranteed-service connections)

Each scheduling discipline makes a different trade-off among these requirements. Depending on the situation, some of these requirements may be more important than others. The "best" choice, therefore, depends on the applicable binding constraints. In this section, we study the requirements in more detail.

---

[27] However, some audio applications can change the audio codec used in order to adapt to the available bandwidth.

[28] We admit the existence of such applications in order to motivate the work on quality of service support.

1.2.1 Ease of implementation

In a high-speed network, a server may need to pick the next packet for transmission every time a packet departs, which can be once every few microseconds. Thus, it has very little time to make a decision. A scheduling discipline for a high-speed network should require only a few simple operations; preferably, it should be implementable inexpensively in terms of hardware. In particular, the number of operations to implement a discipline should be as independent of the number of scheduled connections as possible. If a switch is serving N simultaneous connections, a scheduler that takes $O(N)$ time does not scale, and we prefer a scheduler that takes $O(1)$ time. Local-area switches typically serve 50-100 simultaneous connections, whereas wide-area switches can serve up to 100,000 simultaneous connections. Thus, scaling is particularly important for wide-area switches.

If we want to implement a scheduler in hardware, the scheduling discipline must be amenable to easy hardware implementation. Surprisingly enough, with modern VLSI technology, it is nearly as easy to implement the logic of a complicated scheduling algorithm as that of a simple one. The binding constraint, instead, is mainly in the memory required to maintain *scheduling state* (such as pointers to packet queues, and memory about the service already received by a connection) and the time required to access this state. The smaller the scheduling state associated with scheduling discipline, the easier it is to implement in VLSI. For example, implementing a scheduler that keeps all packets in a single shared buffer is easy, because the only state the scheduler requires is pointers to the head and tail of the shared queue, and these pointers can be rapidly accessed. In contrast, a scheduler that keeps a per-connection service queue needs a pointer to the head and tail of every queue. The state needed scales linearly with the number of connections, raising questions about its feasibility in a large wide-area switch.

1.2.2 Fairness and protection

A scheduling discipline allocates a share of the link capacity and output queue buffers to each connection it serves. We call an allocation at a switch fair if the allocation satisfies the max-min allocation criterion[29]. Fairness is an intuitively desirable property of a scheduling discipline serving best-effort connections. For guaranteed-service connections, which should pay the network operator a fee in proportion to their resource usage, fairness is not a concern.

Fair resource allocation to a set of connections is a *global* objective, whereas a scheduling discipline takes only *local* resource allocation decisions. To translate from a local decision to a global one, each connection should limit its resource usage to the *smallest* locally fair allocation along its path. It can be shown that this results in a globally fair allocation. Note that a sudden decrease in the resource usage by one connection can potentially increase the fair allocations of other connections who share part of their path with that connection. However, sources can increase their resource usage to the new globally fair allocation only after a propagation delay. Thus, in a network where usage patterns change rapidly with time, even if every switch locally allocates resources fairly, users may not receive globally fair allocations, because by the time a source adapts to the current allocation, its allocation may have changed!

Protection means that misbehavior by one connection (by sending packets at a rate faster than its fair share) should not affect the performance received by other connections. For example, FCFS does not provide protection, because the mean delay of a source may increase if the sum of the arrival rates over all sources increases. Thus, a misbehaving source, by sending too fast, increases the mean delay of all other connections. In contrast, with round-robin scheduling, a misbehaving source overflows its own queue, and the other sources are unaffected. If a scheduler provides protection, then it also guarantees a minimum bandwidth to every connection, whatever the behavior of other connections.

The relationship between fairness and protection is that a fair scheduler automatically provides protection, because it limits a misbehaving connection to its fair share. However, the converse need not be true. For example, if connections are *policed* at the entrance to the network (that is, the network forces them to conform to a predeclared traffic pattern) they are protected from each other, but their resource shares may not be fair. We study scheduling disciplines that provide fair service in Section 1.4.

1.2.3 Performance bounds

The third major requirement of a scheduling discipline is that it should allow a network operator to guarantee *arbitrary* per-connection performance bounds, restricted only by the Conservation Law (for example, we cannot give *all* connections a delay lower than they would receive with FCFS). An operator can guarantee performance bounds for a connection only by reserving some network resources, either on-the-fly, during the call-establishment phase of the connection, or in advance. Since the amount of resources reserved for a connection depends on its traffic intensity,

[29] The max-min fair share allocation is defined as follows: resources are allocated in order of increasing demands, no source gets a resource share larger than its demand, sources with unsatisfied demands get an equal share of the resource.

guaranteed-performance connections must agree to limit their usage. We can view the relationship between the network operator and the user as a legal contract: the user agrees that its traffic will remain within certain bounds, and, in turn, the operator guarantees that the network will meet the connection's performance requirements. In this bloc, we assume that a user somehow communicates the performance requirements for a connection to the operator. Note that to meet its contract, an operator must control a connection's performance not only when served by a single scheduler, but also when the connection passes through many schedulers in tandem. In a heterogeneous network, where different parts of the network may employ different scheduling disciplines, guaranteeing end-to-end performance bounds is a hard problem, and an area of active research.

To specify and guarantee performance requirements, we have to be more precise about how to measure a connection's performance. We now take an extended diversion to study this problem.

Deterministic and statistical bounds

We start by noting that performance bounds can be expressed either *deterministically* or *statistically*. A deterministic bound holds for every packet sent on a connection. A statistical bound is a probabilistic bound on performance. For example, a deterministic delay bound of 10 s means that every packet sent on a connection has an end-to-end delay smaller than 10 s. On the other hand, a statistical bound of 10 s with a parameter of 0.99 means that the probability that a packet has a delay greater than 10 seconds is smaller than 0.01. In general, deterministic bounds require a larger fraction of the network resources to be reserved than statistical bounds.

Another way to express statistical bounds is as a *one-in-N* bound. In the latter form, the guarantee is that no more than one packet in N *consecutive packets* will violate the bound. For example, the statistical bound just discussed could be expressed as follows: no more than one packet in 100 will have a delay larger than 10 seconds. Statistical bounds of the second sort are *easier to* verify at the endpoint but harder to implement, because intermediate network elements need to keep track of the state of every connection. Continuing with the example, if a network element drops a packet from a connection, it must remember not to drop packets from that connection for at least a hundred more packets (the situation is even more complex when multiple switches must cooperate to meet a one-in-N bound).

Common performance parameters

Four common performance parameters are widely used in the literature: bandwidth, delay, delay-jitter, and loss. We study these in turn.

A *bandwidth* bound requires that a connection receive at least a minimum bandwidth (measured over a prespecified interval) from the network. Guaranteed-service connections usually require at least a bandwidth bound. In practice, most current integrated-service networks provide only a bandwidth bound.

A *delay* bound is a deterministic or statistical bound on some parameter of the delay distribution, such as the worst-case delay, the mean delay, or the 99-percentile delay (Figure B7T17).

- The *worst-case* delay is the largest delay suffered by a packet from a connection. To compute the worst-case delay, we assume that every other connection at every scheduler along the path behaves in the worst possible manner. Computing it is easy if we can clearly identify the worst case.

- Strictly speaking, we must measure the *average* delay over all possible traffic arrival patterns of every other connection in the system, because these may influence the delay of a packet in the connection under study. Thus, the *true* average delay is impossible to measure, or even to define precisely. In practice, when we refer to a connection's average delay, we are talking about the mean delay *measured* over the packets sent on that connection. If the connection lasts long enough. and the other traffic sources are independent, then this approximates the true average delay.

- Ninety-nine percent of the packets on a connection are guaranteed to suffer a delay smaller than the *99-percentile* delay. As with the mean delay, the true 99percentile delay is impossible to measure, and we usually settle for the measured 99-percentile delay.

  Though the worst-case delay is usually substantially larger than the measured mean delay, because it is impossible for a network to compute or guarantee the true average-case delay, guaranteed-service networks are expected to specify and guarantee only the deterministic or statistical worst-case delay.

  Some disciplines couple bandwidth and delay bounds such that to obtain a lower worstcase delay, a connection needs to reserve a larger bandwidth at a scheduler. This is inefficient for connections that would like to obtain a smaller worst-case delay while still reserving only a small bandwidth. We therefore prefer disciplines that decouple performance bounds over disciplines that couple them.

- A *delay-jitter* bound requires that the network bound the difference between the largest and smallest delays received by packets on that connection (this is equivalent to bounding the width of the end-to-end delay histogram-see Figure B7T17). Note that the delay-jitter bound is trivially as large as the delay bound less the propagation delay, since a packet's queuing delay could be as small as zero and as large as the delay bound less the propagation delay.

Finally, a loss *bound* requires that the fraction of packets lost on a connection be smaller than some bound. As with the average delay, the true loss rate is impossible to measure, and we usually settle for the loss rate measured over a certain number of packets, over a certain time interval, or over the lifetime of the connection. In this bloc, we will only consider the simple case of a zero loss bound.

1.2.4 Ease and efficiency of admission control

A scheduling discipline should permit easy admission control. A switch controller should be able to decide, given the current set of connections and the descriptor for a new connection, whether it is possible to meet the new connection's performance bounds without jeopardizing the performance of existing connections. Moreover, the scheduling discipline should not lead to network underutilization (subjectively measured by the network operator). For example, with FCFS scheduling, we can give all connections a worst-case delay bound by restricting the number of connections and the largest burst size that each connection may send. However, this typically results in an underutilized network.

1.3 Fundamental choices

There are four principal degrees of freedom in designing a scheduling discipline:

- The number of priority levels (Section 1.3.1)
- Whether each level is work-conserving or non-work-conserving (Section 1.3.2)
- The degree of aggregation of connections within a level (Section 1.3.3)
- Service order within a level (Section 1.3.4)

In this section, we will examine these four choices.

1.3.1 Priority

In a priority scheduling scheme, each connection is associated with a priority level. If there are *n* priority levels, and a higher-numbered priority level corresponds to a connection with higher priority, the scheduler serves a packet from priority level *k* only if there are no packets awaiting service in levels $k + 1, k + 2,... n$ (we also call this *multilevel priority with exhaustive service).* Priority allows a scheduler to give packets at a higher priority level a lower mean queuing delay at the expense of packets at lower priority levels.

A scheduler can have an arbitrary number of priority levels. In practice, the number of levels depends on the number of delay classes that the network operator wants to support. In an integrated services network, at least three priority levels are desirable: a high priority level for urgent messages, usually for network control; a medium priority level for guaranteed service traffic; and a low priority level for best-effort traffic.

A priority scheme allows a misbehaving user at a higher priority level to increase the delay and decrease the available bandwidth of connections at all lower priority levels. An extreme case of this is *starvation,* where the scheduler never serves a packet at a lower priority level, because there is always something to send from a higher priority level. Thus, in a priority scheduler, it is critical that appropriate admission control and policing restrict the service rates from all but the lowest priority level.

Implementation

Priorities are simple to implement in both software and hardware because, to make a scheduling decision, a scheduler needs to determine only the highest priority nonempty service queue. It also requires only a small amount of scheduling state for buffer management: two pointers at each priority level that point to the head and tail of the service queue at that level.

Analysis

If a combination of admission control and policing restricts the rate at which data enters a priority scheduler, the performance of connections at a lower level can be computed by modeling service at that level by a *vacationing server.* The idea is that the server is on vacation when it is serving higher priority levels. The worst-case arrival pattern at the higher priority levels bounds vacation durations, and thus the worst-case delay for the packet at the head of the per-level service queue (subsequent packets could have larger worst-case delays). Moreover, the mean rate at which level *k* is served is just the link rate minus the rate at which vacations are taken at all higher priority levels.

1.3.2 Work-conserving versus non-work-conserving disciplines

A work-conserving scheduler is idle only when there is no packet awaiting service. In contrast, a non-work-conserving scheduler may be idle even if it has packets to serve. At first glance, non-work-conserving schedulers make little sense: why would network operators want to leave a line idle, wasting bandwidth, if they could use the time to carry traffic? The reason is that a non-work-conserving scheduler, by idling away time, makes the traffic arriving at downstream switches more predictable, thus reducing both the buffer size necessary at output queues, and the delay jitter experienced by a connection. Let us see how this works with an example.

Example 1.2

Consider two switches S1 and S2 in tandem, with connections A and B sharing the link S1-S2, but going to different output queues at switch S2 (Figure B7T22). Now, focus on the pattern of packet arrivals from A at S2. Since A and B share link S1 - S2, this arrival pattern may depend not only on the way A is served at S1, but also on B's traffic arrival pattern at S1. For example, if S1 serves packets in FCFS order, and many packets from A pile up behind a burst from B at the head of the queue, when B's packets finally depart, packets from A will arrive at S2's output queue in a burst, even if they entered S1 evenly spaced. Observe that if S1 is free to transmit an entire burst from A at the link rate, then, to prevent packet loss, S2's output queue must be large enough to absorb the burst. Moreover, if this burst is buffered at S2 awaiting service, and more packets arrive on connection A before S2 can serve the burst, then the burst grows even larger as it exits S2. It is possible to come up with pathological scenarios where each switch adds a little to an initial burst so that as the traffic from A leaves the network, it does so as a series of fast bursts. This potentially bursty behavior leads to a large delay jitter (because the first packet of a burst has a smaller queuing delay than the last) and the need for larger buffers at switches (because they must accommodate large bursts). With a non-work-conserving discipline, a connection needs the same number of buffers, no matter how deep in the network it is.

We can reduce both the delay jitter and the number of buffers needed to prevent packet loss if we allow a switch to send a packet only when the packet is *eligible.* If a packet is not eligible for service, the switch leaves its output queue idle until the packet becomes eligible. By choosing eligibility times carefully, we can ensure that the output from a switch is predictable, so that bursts do not build up within a network. For example, if the $(k + 1)^{st}$ packet on connection A becomes eligible for service only *i* seconds after the service of the k^{th} packet, we ensure that the downstream switch receives packets on A no faster than one every *i* seconds. If the downstream switch serves the connection at a rate faster than one every *i* seconds, it needs to buffer at most one packet from A, limiting the size of the buffer at the switch. Moreover, since the queuing delay at each switch is bounded, the end-to-end delay jitter for packets from A is also bounded. Thus, a scheduler that delays packets until they are eligible, while potentially wasting bandwidth, makes the traffic in the network more predictable and easier to manage.

Besides reducing the need for switch buffers and the delay-jitter bound, non-work-conserving disciplines have two other advantages. First, with non-work-conserving schedulers, the sum of the per-hop bounds can tightly bound the end-to-end delay and delay-jitter bounds. This makes the end-to-end performance bounds relatively simple to compute. Second, since the regulator reshapes traffic at each hop, it is easy to bound the performance of heterogeneous networks, where each switch may implement a different non-work-conserving scheduling discipline. This becomes particularly important when it is necessary to compute performance bounds over paths that span multiple administrative domains.

Choosing eligibility times

There are two well-understood ways for a scheduler to choose eligibility times. With *rate-jitter* regulation, the scheduler guarantees that source traffic leaving the scheduler conforms to a given *rate descriptor.* For example, assume that all packets are the same size, and the scheduler wants to guarantee a particular peak rate. Then, the eligibility time for a packet is the sum of the eligibility time of the previous packet on that connection and the inverse of the peak rate (that is, the time taken to serve a fixed-size packet at the peak rate). More precisely, if $E(k)$ represents the eligibility time for the kth packet, and $A(k)$ its arrival time at the scheduler, then

$E(1) = A(1)$

$E(k + 1) = \max(E(k) + Xmin, A(k + 1))$,

where *Xmin is* the inverse of the peak rate, measured in seconds/packet. This is similar to the peakrate regulator described in bloc 4.

Another way to choose eligibility times is with a *delay-jitter regulator*. A scheduler implementing this regulator guarantees that the sum of the *queuing* delay in the previous switch and the *regulation* delay in the current switch is constant. This removes the effect of queuing-delay variability in the previous switch. In other words, the output stream from the regulator is a time-shifted and *fully reconstructed* version of the traffic that entered the previous switch. By composing a series of such regulators, the network assures a source that at every switch the input arrival pattern is fully reconstructed, so that burstiness does not build up within the network. More precisely, if E(*i, k) is* the eligibility time for the *k*th packet at the *i*th switch, then

$E(0,k) = A(0,k)$

$E(i + 1, k) = E(i, k) + D + L$

where *D* is the delay bound at the previous switch, and L is the largest possible delay on the transmission link between switch *i* and switch *i* + 1. The *k*th packet is eligible for service at the 0th switch the moment it arrives. However, at subsequent switches, it becomes eligible for service only after a fixed time interval of length *D* + *L*, which is the longest possible delay in the previous switch and in the previous link. So, if a packet is served before its delay bound at the previous switch, or receives a delay smaller than L on the link, the delay-jitter regulator at the downstream switch adds sufficient delay to convert this to the longest possible delay. When the packet leaves the regulator, it is as if it received exactly D seconds of queuing delay at the previous switch and L seconds of propagation delay at the previous link. Thus, the output of the regulator is a (*D* + *L*)-second time-shifted version of the traffic that left the previous regulator along the path. If a source is policed at the input to obey a certain traffic descriptor, the delay-jitter regulator automatically assures us that the source's traffic obeys this descriptor at the output of every regulator in the network. Note that a delay-jitter regulator, by itself, cannot provide protection between sources, because misbehaving traffic at the entrance to the network is simply replicated at every hop.

A delay-jitter regulator is harder to implement than a rate-jitter regulator. Not only does it require the network operator to know a bound on the propagation delay on each link it also requires the network to maintain clock synchrony at adjacent switches at all times. Since, in the real world, clocks drift out of synchrony unless corrected, delay-jitter regulation implicitly assumes the presence of a mechanism for maintaining clock synchrony. Because of its complexity, it seems unlikely that delay-jitter regulation will make the transition from a research paper to the real world. Nevertheless, it is worth studying, because it introduces the notion of perfect traffic reconstruction. We can judge the effectiveness of other regulators by the extent to which they can reconstruct the input traffic stream.

1.3.3 Degree of aggregation

The third degree of freedom in the design of scheduling disciplines is the degree to which individual connections are aggregated in deciding their service order. At one extreme, the scheduler uses a single state variable to describe all connections, which must all, therefore, share the same quality of service. At the other extreme, the scheduler maintains per-connection state, and can give different connections different bandwidth and delay bounds. With an intermediate degree of aggregation, the scheduler treats all packets from connections in the same *class* the same way (that is, maintains per-class state). In this scheme, the scheduler provides different qualities of service to different classes, while connections within a class share the same service quality. Classes of service, therefore, allow us to bridge the spectrum between per-connection service guarantees and a single shared service quality.

Aggregation trades off a smaller amount of scheduler state for a coarser-grained differentiation in quality of service. Moreover, as the preceding example shows, for some disciplines, the greater the degree of aggregation, the fewer connections can be admitted. Why, then, should we aggregate connections? The main advantage is in reducing the state in the scheduler, which is important for two reasons. First, as discussed in Section 1.2.1, scheduler state is the critical resource in implementing a scheduler. Thus, the smaller the amount of scheduler state required for a particular scheduling discipline, the easier it is to implement. Second, when establishing a connection, a source needs to know if schedulers along the route can support its performance requirements. Instead of blindly trying all possible paths, switches could advertise their current state to the rest of the network, allowing a source to select a path that is likely to have sufficient resources. However, the more state there is in the scheduler, the more there is to advertise, which costs bandwidth. Thus, reducing the amount of scheduler state is a good idea.

A second advantage of aggregation is that it evenly distributes the jitter induced by bursts due to members of the class. Consider a set of ten connections sharing a non-work-conserving scheduler that regulates each of them to the same service rate. Assume that nine of them send packets evenly spaced, while a tenth one sends them in bursts of five packets at a time. If connections are not aggregated, the tenth connection would have a large delay jitter, because the fifth packet from every burst would have a much larger delay than the first. The other connections would be unaffected by this burstiness. In contrast, if the connections are aggregated, the burst from the tenth connection would be served

consecutively, so that all other connections would see an increased jitter. However. the jitter for the tenth connection would be smaller because all other members in its aggregation class share its jitter. Thus. aggregation allows connections to obtain a smaller jitter when they send a burst, at the expense of being willing to absorb jitter when other connections sharing the class send a burst.

The main problem with aggregation is that connections within the same class are not protected from each other. Because the scheduler cannot distinguish between connections in the same class, the misbehavior of one connection in the class affects the whole. For example, if the class is served by a non-work-conserving scheduler with a peak-rate rate-jitter regulator, even if one connection of the class sends data too fast, all other connections will experience packet delays and may suffer from packet loss. Thus, the degree of aggregation is inversely proportional to the degree of protection. When designing a network, we have to balance these pros and cons to choose the appropriate degree of aggregation.

A second problem with aggregation is that if a scheduler gives aggregated congestion feedback signals to an ensemble of feedback flow-controlled connections, a well-behaved connection may perceive congestion signals caused by the bad behavior of other connections as misbehavior on its own part. Therefore, if any one of the aggregated connections sends at a rate fast enough to build up queues at bottleneck, all the other connections receiving a shared feedback signal will be affected. We can show that in a network of schedulers that provide aggregate feedback to an ensemble of feedback flow-controlled connections, even if individual connections are well-behaved, the ensemble could still be unstable (that is, the vector of transmission rates from each source never converges to an equilibrium value). Moreover, if any one of the connections chooses to ignore congestion signals, it can hog the entire capacity available to the aggregate. For example, if we have an ensemble of TCP-style feedback flow-control connections sharing a single FCFS queue, where the aggregated feedback signal is packet loss, the misbehavior of one connection will cause all other connections to lower their window size, and thus their transmission rate. In contrast, under the same assumptions, non-aggregated service and non-aggregated feedback results in guaranteed fair allocations. Thus, whenever it is feasible, we prefer per-connection queuing to aggregated queuing. However, if the traffic sharing a class is well behaved, and if all the connections sharing a class have approximately the same behavior, then aggregating them into a class does not significantly affect the degree of protection. (If they are well behaved, they do not need to be protected from each other in the first place!)

1.3.4 Service order within a priority level and aggregation class

The fourth and final degree of freedom in designing scheduling disciplines is the order in which the scheduler serves packets from connections (or aggregates of connections) at the same priority level. There are two fundamental choices: serving packets in the order they arrive, or serving them out of order, according to a per-packet *service tag.* With the second option, the properties of the scheduler depend heavily upon how the scheduler computes the tag. Nevertheless, all scheduling disciplines that serve packets in non-FCFS order need to sort packets implicitly or explicitly, which results in an implementation overhead.

The main problem with FCFS is that it does not allow packets that want a lower delay to skip to the head of the queue. In contrast, if we serve in order of service tags, we can give packets that want a low delay a lower tag value than others in the queue, which allows them to jump to the head of the queue. A second problem with FCFS service is that the allocation of bandwidth to individual connections is not max-min fair (see Section 1.2). Connections receive service roughly in proportion to the speed at which they send data into the network. During overload, when bandwidth is scarce, this rewards bandwidth "hogs," at the expense of connections obeying a cooperative closed-loop flow control scheme, such as the TCP dynamic window scheme. Thus, FCFS service is an incentive for an endpoint to behave greedily, because greediness is rewarded!

With service in order of service tags, it is possible to achieve allocations that are "close" to max-min fair. We discuss this in more detail in Section 1.4.

Implementation

FCFS service is simple to implement, because packets can be placed in a queue as they arrive and removed from the queue when the trunk becomes free. To implement a discipline that serves packets according to explicit service tags, we need several mechanisms. First, on packet arrival, we need to compute its tag. This may require looking up the state associated with its connection, or with the aggregate that contains its connection. We must then insert the tagged packet in a data structure that allows us to determine the packet with the smallest service tag (since this is the next packet to serve) and the largest service tag (since this is the next packet to drop, if the scheduler runs out of buffers).

1.3.5 Summary

In this section, we have examined the four degrees of freedom in designing scheduling disciplines: (1) priority, (2)

work-conserving or non-work-conserving service, (3) degree of aggregation. and (4) service order within a priority level. By appropriately combining elements along each axis, we can come up with new disciplines that provide a trade-off between implementation complexity and desirable features. The important message, as always, is that each feature comes at some cost. Depending on the situation, a designer must match the need for a feature with its implementation cost. For example, for a small LAN switch, where traffic loads are likely to be much smaller than the available capacity, service queues are usually small, and users are cooperative. In such a system, a singlepriority FCFS scheduler, or, at most, a two-priority scheduler, with the higher level devoted to guaranteedservice traffic, is sufficient. For a heavily loaded wide-area public switch with possibly noncooperative users, a scheduler must provide protection with the ability to sustain high trunk utilization. This may require a more sophisticated scheduling discipline, which may involve the use of multiple priority levels, non-workconserving service at some priority levels, aggregation within some priority levels, and non-FCFS service order.

1.4 Scheduling best-effort connections

1.4.1 Generalized processor sharing

Section 1.2 makes the case that the scheduling objectives for best-effort and guaranteed-service connections are different. For best-effort connections, we would like the scheduling discipline to provide a *max-min* fair allocation, as described in Section 1.2. We can achieve a max-min fair allocation with an ideal (and unimplementable) work-conserving scheduling discipline called *Generalized Processor Sharing* (GPS). Intuitively, GPS serves packets as if they are in separate logical queues, visiting each nonempty queue in turn and serving an infinitesimally small amount of data from each queue, so that, in any finite time interval, it can visit every logical queue at least once. Connections can be associated with service weights, and they receive service in proportion to this weight whenever they have data in the queue. If they do not have data, the scheduler skips to the next nonempty queue. We claim that this service order results in a max-min fair share bandwidth allocation.

To see this, consider a switch where N connections with equal weights send data to the scheduler infinitely fast. The server should allocate each of them a $1/N^{th}$ share of the bandwidth, which is their max-min fair share. Because the scheduler serves an infinitesimal from each connection in turn, it achieves this goal. Now, if one source, say, source A, sends data more slowly than this share, its queue at the scheduler is occasionally empty. When this is so, the scheduler skips A's queue, and, because of its round-robin service, the time thus saved is equally distributed to the other connections. Now, if another connection, say, B, has an incoming rate that is *larger* than $1/N$ but *smaller* than the new service rate it receives because A's queue is occasionally empty, B's queue will also be occasionally empty. Thus, the remaining connections (other than A and B) will receive a little more service, which may, in turn, cause still other connections' queues to be occasionally empty. Continuing in this fashion, we see that every connection that has a demand smaller than its fair share gets allocated its demand, whereas every connection that has a greater demand gets an equal share. Thus, by definition, GPS service achieves the max-min fair share.

If we allow a connection to specify a weight, then in each round of service, a GPS server serves data from each nonempty connection queue in proportion to the connection's weight. An extension of the previous argument shows that the GPS server also achieves the max-min *weighted* fair share. Note that, because GPS is fair, from the arguments in Section 1.2.2, it also offers protection.

While GPS is ideal in that it exactly achieves a max-min fair allocation, it is also unimplementable. We now study some scheduling disciplines that are implementable, and try to approximate GPS

1.4.2 Weighted round-robin

The simplest emulation of GPS is *round-robin,* which serves a packet from each nonempty connection queue, instead of an infinitesimal amount. Round-robin approximates GPS reasonably well when all connections have equal weights and all packets have the same size. If connections have different weights, then *weighted round-robin* (WRR) serves a connection in proportion to its weight.

Example 1.3

Suppose connections A, B, and C have the same packet size, and weights 0.5, 0.75, and 1.0. How many packets from each connection should a round-robin server serve in each round?

Solution: We normalize the weights so that they are all integers, giving us weights 2, 3, and 4. Then in each round of service, the server serves two packets from A, three from B, and four from C.

If packets from different connections have different sizes, a weighted round-robin server divides each connection's weight by its mean packet size to obtain a normalized set of weights.

Example 1.4

Suppose connections A, B, and C have mean packet sizes of 50,500, and 1500 bytes, and weights 0.5, 0.75, and 1.0. How many packets from each connection should a round-robin server serve in each round?

Solution: We divide the weights by the mean packet size to obtain normalized weights 0.01, 0.0015, and 0.000666. Normalizing again to obtain integer weights, we get weights 60, 9, and 4. Thus, the scheduler serves 60 packets from A, 9 from B, and 4 from C in each round of service. This results in 3000 bytes from A, 4500 bytes from B, and 6000 from C served in each round, which is exactly according to their weights of 0.5, 0.75, and 1.0.

To emulate GPS correctly when packets can be of different sizes, a weighted round-robin server must know a source's mean packet size in advance. However, in practice, a source's packet size may be unpredictable. For example, if a source is sending compressed video images, packet size may depend on the degree of motion in a scene. If a source cannot predict its mean packet size, a weighted round-robin server cannot allocate bandwidth fairly.

A second problem with weighted round-robin service is that it is fair only over time scales longer than a round time. At a shorter time scale, some connections may get more service than others. If a connection has a small weight, or the number of connections is large, this may lead to long periods of unfairness. This is illustrated by Example 1.5.

Example 1.5

Consider a wide-area T3 trunk that serves 500 ATM virtual circuits with weight 1 and 500 virtual circuits with weight 10. If no connection is ever idle, what is the length of one round?

Solution: A Kilobyte cell takes 9.422 g on a 45 Mbps T3 trunk. A round takes 500 + 500 * 10 = 5500 service times = 51.82 ms. Thus, over a time smaller than 51.82 ms, some connections get more service than others (that is, the service can be unfair).

More sophisticated scheduling disciplines, which we study next, eliminate these two problems. These sophisticated schemes are desirable mainly in the context of a variable-packet size or slower-speed network. In high-speed ATM networks, with fixed-size packets and short round times, GPS emulation using weighted round-robin is usually good enough.

1.4.3 Deficit round-robin

Deficit round-robin modifies weighted round-robin scheduling to allow it to handle variable packet sizes without knowing the mean packet size of each connection in advance. A DRR scheduler associates each connection with a *deficit counter* initialized to 0. The scheduler visits each connection in turn and tries to serve one *quantum* worth of bits from each visited connection. The packet at the head of the queue is served if it is no larger than the quantum size. If it is larger, the quantum is added to the connection's deficit counter. If the scheduler visits a connection such that the sum of the connection's deficit counter and the quantum is larger than or equal to the size of the packet at the head of the queue, then the packet at the head of the queue is served, and the deficit counter is reduced by the packet size. For example, let the quantum size be 1000 bytes, and let connections A, B, and C have packets of size 1500, 800, and 1200 bytes queued at a DRR scheduler (Figure B7T34). In the first round, A's counter increases to 1000, B's first packet is served, and its deficit counter becomes 200 (= 1000 - 800). C's counter is 1000, and no packet from C is served. In the second round, A's packet is served, and its counter is set to 1000 + 1000 - 1500 = 500. Similarly, C's counter is set to 800. Since there is no packet at B, its counter is reset to 0 (otherwise, B builds up credits indefinitely, eventually leading to unfairness). In the weighted version of DRR, the scheduler serves the quantum size times $\phi(i)$ bits from connection i.

To assure that the DRR scheduler always serves at least one packet from each connection, the quantum size is recommended to be at least *Pmax*, the largest possible packet size in the network. A DRR scheduler does only a constant amount of work per step in its execution. Thus, implementing in hardware is easy.

Let the frame time, F, be the largest possible time taken by the scheduler to serve each of the backlogged connections (taking into account their relative weights). We can also show that the relative fairness bound for DRR is $3F/r$. Thus, the fairness is linked to the frame size.

The main attraction of DRR is its ease of implementation. However, like weighted round robin, it is unfair at time scales smaller than a frame time. For example, consider a 45 Mbps link shared by 500 connections, where the packet size can be as large 8 Kbytes. Then, the frame time can be as large as 725 ms. DRR, therefore, is unfair on time scales smaller than 725 ms. Things are much better if the packet size is 48 bytes, as with ATM networks, but with fixed-size packets DRR reduces to weighted round-robin. Thus, DRR is suitable when fairness requirements are loose or when the packet sizes are small.

1.4.4 Weighted fair queuing and packet-by-packet generalized processor sharing

Weighted fair queuing (WFQ) and packet-by-packet generalized processor sharing (PGPS) are approximations of GPS scheduling that do not make GPS's infinitesimal packet size assumption, and, with variable-size packets, do not need to know a connection's mean packet size in advance. Since PGPS and WFQ, though independently invented, are identical, we will only study WFQ

The intuition behind WFQ is to compute the time a packet would complete service had we been serving packets with a GPS server, then serve packets in order of these finishing times. In other words, WFQ simulates GPS "on the side" and uses the results of this simulation to determine service order. We call a packet's finishing time under GPS a *finish number,* rather than a finish time, to emphasize that it is only a service tag that indicates the relative order in which the packet is to be served, and has nothing to do with the actual time at which the packet is served.

Finish number computation

The computation of the finish-number depends on a variable called the round number. We will first study how to compute the round number when all weights are equal. For the moment, model a GPS server as doing bit-by-bit service, rather than infinitesimal-by-infinitesimal service. With this simplification, we define the round number to be the number of rounds of service a bit-by-bit round-robin scheduler has completed at a given time. A non-integer round number represent a partial round of service, so that, for example, a round number of 3.5 indicates that we are halfway through the fourth round of service. Note that each round of service takes a variable amount of time: the more connections served in a round, the longer the round takes. We call a connection active if the largest finish number of a packet either in its queue or last served from its queue is larger than the current round number. Thus, the length of a round, that is, the time taken to serve one bit from each active connection, is proportional to the number of active connections.

If we know the round number, we calculate the finish number as follows. The finish number of a packet arriving at an *inactive* connection is the sum of the current round number and the packet size in bits, because this is the round number when a bit-by-bit round-robin server would have completed service of that packet. For example, if a packet of size 10 bits arrives when the round number is 3, then the packet completes service ten rounds later, in the thirteenth round, when the round number is 13. If a packet arrives on an *active* connection, the arriving packet's finish number is the sum of the largest finish number of a packet in its queue (or last served from its queue) and the arriving packet's size (in bits). For example, if a packet of size 10 bits arrives to a queue that already contains a packet with a finish number of 20, we know that the packet in the queue completes service when the round number is 20. Thus, the incoming packet completes service when the round number reaches 20 + 10 = 30. Combining these statements, if $P(i, k, t)$ is the size of the kth packet that arrives on connection i at time t, when $R(t)$ is the round number, and F (i, k - 1, t) is the finish number for the $(k - 1)$th packet on that connection, then

$$F(i, k, t) = \max\{F(i, k - 1, t),\ R(t)\} + P(i, k, t)$$

We define the largest finish number of a packet in a connection's queue, or served from that connection's queue, to be the *connection's* finish number.

We mentioned earlier that the round number increases at a rate inversely proportional to the number of active connections. Instead of viewing the round number as the number of service rounds completed by a bit-by-bit round-robin server, we can redefine the round number to be a real-valued variable that increases at a rate inversely proportional to the number of currently active connections. From this perspective, the round number no longer has a physical meaning: it is just a convenient abstraction useful in computing finish numbers. With this modification, WFQ emulates GPS, instead of bit-by-bit round-robin scheduling.

Implementation

When a packet arrives to a fair queuing scheduler, the scheduler updates its notion of the current round number, if necessary performing an iterated deletion. The scheduler then computes the finish number of the packet and places it in a priority queue, so that it is served in order of its finish number. Note that within a connection, finish numbers increase monotonically with time. The key to implementing fair queuing is in devising a fast and efficient priority queue, as discussed in Section 1.3.4.

Evaluation

Weighted fair queuing (or PGPS) has three desirable properties. First, because it approximates GPS, it protects connections from each other. This firewalling property *is* important for public data networks. Second, under certain assumptions, a connection can obtain a worst-case end-to-end queuing delay that is independent of the number of hops it traverses and of the behavior of other connections (we will study this in Section 1.5). This allows networks of fair queuing schedulers to provide real-time performance guarantees, which is important for guaranteed-service connections. Finally, WFQ gives users an incentive to implement intelligent flow control mechanisms at the end-systems. With

WFQ, a source is not required to send at a rate smaller than its currently allocated rate. However, a source that consistently sends more than its fair share is likely to lose packets from its own buffers, so it has an incentive to match its flow to the currently available service rate.

On the other hand, a WFQ scheduler requires per-connection (or per-aggregate) scheduler state, which leads to implementation complexity and can be expensive for schedulers that serve large numbers of connections. It requires an expensive iterated deletion algorithm to update its round number. Moreover, it requires explicit sorting of the output queue on the service tag, which requires time and complex hardware or software.

Despite these problems, increasing numbers of manufacturers are implementing weighted fair queuing in their router and switch products. In the late 90s, both the leading line of routers from Cisco, Inc., and ATM switches from FORE Systems, Inc., provide some form of WFQ scheduling. In the next subsection, we will examine some variants of WFQ that alleviate some problems faced with WFQ.

However, by 2003, although WFQ technology is available in routers, it is not widely "used" and no scheduling is done in the intermediate routers.

1.4.5 Summary

The ideal discipline for best-effort connections is generalized processor sharing (GPS) (Section 1.4.1), but it is unimplementable because it serves an infinitesimal amount from each nonempty connection queue. An obvious emulation of GPS uses weighted round-robin (Section 1.4.2), but this does not work well if packets can be of different sizes, or if we want a fair allocation of bandwidth at small time scales. The deficit round-robin discipline is similar to weighted round-robin, but works well even with variable-size packets (Section 1.4.3). However, it, too, does not allocate bandwidth fairly at short time scales. We overcome this with the weighted fair queuing (WFQ) algorithm, where packets are served in order of service tags, which are computed by simulating GPS in parallel (Section 1.4.4). Although WFQ compares favorably with GPS in fairness, it is complex because of its simulation of GPS.

1.5 Scheduling guaranteed-service connections

1.5.1 Weighted fair queuing

It turns out that the weighted fair queuing (WFQ) discipline can also be used to give connections performance guarantees. First, note that WFQ gives a bandwidth bound to each connection, since connection i is guaranteed $\phi(i, k) / \Sigma(\phi(j, k)$ share of the link capacity at its kth hop. Parekh and Gallager proved an important bound on the worst-case end-to-end delay suffered by a connection traversing a series of WFQ schedulers; we now state this bound.

Let a leaky-bucket-constrained source i with parameters $(\sigma(i), \rho(i))$ pass through K schedulers, where the kth scheduler, $1 \leq k \leq K$, has a link rate $r(k)$. Let $g(i, k)$ be the service rate assigned to the connection at the kth scheduler, where:

$$g(i, k) = \phi(i, k).r(k) / \Sigma \phi(j, k)$$

and we take the sum over all connections sharing the kth scheduler. Let $g(i)$ be the smallest of the $g(i, k)$s over all k. We assume that $g(i) \geq \rho(i)$; otherwise, the queue at one of the schedulers will build up without bound. If the largest packet allowed on that connection is of size $Pmax(i)$, and the largest packet allowed in the network is of size $Pmax$, then, independent of the number of schedulers the connection traverses, and independent of the behavior of the other connections (even if they are not leaky-bucket bounded), the worst-case end-to-end queuing and transmission delay $D^*(i)$ is bounded by:

To understand this result, note that when $Pmax$ is close to 0, that is, when all packets are infinitesimally small, the delay

$$D^*(i) \leq \sigma(i) / g(i) + \sum_{k=1}^{K-1} P\max(i) / g(i,k) + \sum_{k=1}^{K} P\max / r(k)$$

is bounded by $\sigma(i)/g(i)$. Intuitively, this means that though the connection actually traverses a series of schedulers, it behaves as if it were served by a single scheduler with rate $g(i)$, so that when the source sends a burst of length $\sigma(i)$, it experiences a worst-case delay $\sigma(i)/g(i)$. A correction term, $Pmax(i)/g(i, k)$ at each scheduler, models the situation where a packet from source i arrives just after it would have received service under GPS. It can be shown that this packet is delayed by at most $Pmax(i)/g(i, k)$. The third term, which is independent of $g(i)$, reflects the fact that if a packet from i arrives at a busy scheduler, the packet may have to wait up to $Pmax/r(k)$ time before it is served.

Parekh and Gallager's theorem shows that, with a suitable choice of parameters, a network of WFQ servers can provide worst-case end-to-end delay guarantees. A source needing a particular worstcase end-to-end delay bound need only pick

an appropriate value for *g*.

Example 1.6

Consider a connection with leaky bucket parameters (16,384 bytes, 150 Kbps) that traverses 10 hops on a network where all the links have a bandwidth of 45 Mbps. If the largest allowed packet in the network is 8192 bytes long, what *g* value will guarantee an end-to-end delay of 100 ms? Assume a propagation delay of 30 ms.

Solution: The queuing delay must be bounded by 100 - 30 = 70 ms. Plugging this into the above equation, we get $70 * 10^{-3} = \{(16384 * 8) + (9 * 8192 * 8)\} / g + (10 * 8192* 8)/(45 * 10^6)$, so that g = 12.87 Mbps. Notice that this is more than **85** times larger than the source's average rate of 150 Kbps. This is because the $(K - 1)Pmax/g$ term contributes to nearly 46 ms of the 70 ms delay bound, and the $K*Pmax/r$ term contributes another 14 ms. The σ/g term contributes only 10.8 ms to the end-to-end delay! The lesson is that with large packets, packet delays can be quite substantial.

WFQ does not provide a nontrivial delay-jitter bound (i.e., a delay-jitter bound smaller than the delay bound itself). Moreover, as shown in Example 1.6, to obtain a lower delay bound, a connection must make a **much** large reservation, even if it cannot use the entire reserved bandwidth. Finally, WFQ has the problems with implementation complexity that we discussed in Section 1.3.4.

1.5.2 Delay-earliest-due-date and jitter-earliest-due-date

In classic earliest-due-date (EDD) scheduling, we assign each packet a deadline, and the scheduler serves packets in order of their deadlines. If the scheduler is over-committed, then some packets miss their deadlines. With EDD, packets assigned deadlines closer to their arrival times receive a lower delay than packets assigned deadlines farther away from their arrival times.

Delay-EDD is an extension of EDD that specifies the process by which the scheduler assigns deadlines to packets. During call setup, each source negotiates a service contract with the scheduler. The contract states that if a source obeys a peak rate descriptor, then every packet on that connection receives a worst-case delay smaller than some bound. During call admission, the scheduler ensures not only that the sum of the peak rates of the admitted calls is smaller than the link capacity, but also that even in the worst case, when every connection sends traffic at its peak rate, it meets its delay bound (the *schedulability* test).

The key to delay-EDD lies in the way the scheduler assigns deadlines to packets. The scheduler sets a packet's deadline to the time at which it should be sent had it been received according to the connection's contract, that is, slower than its peak rate. By reserving bandwidth at a connection's peak rate, a delay-EDD scheduler can ensure that it has served the previous packet from that connection before the next packet arrives, so that every packet from that connection obeying the peak rate constraint receives a hard delay bound. Note that the delay bound for a connection is independent of its bandwidth reservation, in that a connection reserving a small bandwidth can still obtain a small delay bound. Therefore, unlike GPS-emulation schedulers, delay-EDD separates the bandwidth and delay bounds, but at the cost of reserving bandwidth at the peak rate, giving up temporal statistical multiplexing gain.

Since a delay-EDD scheduler serves packets in order of their deadlines, it needs to place them in a priority queue as in WFQ. The scheduler also needs to store per-connection finish numbers as in WFQ. Thus, its implementation is as complex as a WFQ implementation, except that it does not need round-number computation. Delay-EDD's main advantage over WFQ-like schedulers is that it provides end-to-end delay bounds independent of the bandwidth guaranteed to a connection. On the other hand, it requires every connection to reserve bandwidth at its peak rate, whereas a WFQ-like server need only reserve bandwidth at the connection's average rate. Moreover, it cannot provide a nontrivial end-to-end delay-jitter bound.

In a jitter-EDD (J-EDD) scheduler, a delay-jitter regulator (described in Section 1.3.2) precedes the EDD scheduler. With a delay-jitter regulator, all packets receive the same delay at every hop (except at the last hop), so the difference between the largest and the smallest delays, which is the delay jitter along the connection, is reduced to the delay jitter on the last hop. Thus, a network of J-EDD schedulers can give connections end-to-end bandwidth, delay, and delay-jitter bounds. The J-EDD scheduler incorporates a delay-EDD scheduler, so to obtain a worst-case delay bound, a connection must reserve bandwidth at its peak rate. The call admission control algorithm is identical to that of delay-EDD.

1.5.3 Rate-controlled scheduling

Rate-controlled scheduling disciplines are a class of scheduling disciplines that can give connections bandwidth, delay, and delay-jitter bounds. A rate-controlled scheduler has two components: a *regulator* and a *scheduler* (Figure B7T45). Incoming packets are placed in the regulator, which uses one of many algorithms to determine the packet's *eligibility*

time. When a packet becomes eligible, it is placed in the scheduler, which arbitrates among eligible packets. By delaying packets in the regulator, we can shape the flow of incoming packets to obey any constraint. Two examples of constraints are (1) packets should arrive to the scheduler at a rate less than a peak rate (a rate-jitter regulator) and (2) packets should arrive to the scheduler a constant delay after leaving the scheduler at the previous switch (a delay-jitter regulator). The scheduler can serve packets first-come-first-served, place them in a multilevel priority queue (thus giving some of them lower delays), or serve them using WFQ (thus giving them a weighted fair share of the link bandwidth). The service properties of a rate-controlled scheduler therefore depend on the choices of the regulator and scheduler.

It can be shown that a rate-controlled scheduler can emulate a wide range of work-conserving and non-work-conserving scheduling disciplines. For example, if the regulator ensures that for each connection, over a specified time period, no more than a specified number of cells are marked eligible, then the rate-controlled discipline emulates the hierarchical round-robin discipline described in reference. Similarly, if we use a delay-jitter regulator and an earliest-due-date scheduler, then the rate-controlled discipline emulates jitter-earliest-due-date.

Evaluation

Rate-controlled disciplines have several advantages over other disciplines:

- They are flexible, in that they can emulate a variety of other disciplines.
- With some choices of regulators and schedulers, they can decouple bandwidth and delay assignment.
- With both delay-jitter and rate-jitter regulators, end-to-end delay bounds can be easily computed.
- They do not require complicated schedulers to guarantee protection (the scheduler can just be multilevel static priority or even FCFS).
- With properly chosen regulators, they can emulate WFQ delay bounds without the complexity of maintaining a sorted-priority queue.

On the other hand, they require a per-scheduler calendar queue, and they achieve a delay-jitter bound only at the expense of increasing the mean delay. Besides, if the scheduler uses a delay-jitter regulator, it must, in addition, police all connections, because a delay-jitter regulator does not automatically restrict a connection's bandwidth share. Nevertheless, in view of their advantages and flexibility, rate-controlled disciplines may be the best available disciplines for serving connections that require bandwidth, delay, and delay-jitter bounds in high-speed networks.

1.5.4 Summary

In this section, we studied some scheduling disciplines that allow guaranteed-service connections to obtain bandwidth, delay, and delay-jitter bounds. We first showed that weighted fair queuing (WFQ) allows us to obtain bandwidth and worst-case end-to-end delay bounds (Section 1.5.1). However, to obtain a low end-to-end worst-case delay, a connection must reserve a large bandwidth, reducing the potential for temporal statistical multiplexing gain and making the network more expensive to operate. We then studied delay-earliest-due-date and jitter-earliest-due-date, two disciplines that provide bandwidth and delay bounds and do not couple them, at the expense of a reservation for the peak rate of a connection (Section 1.5.2). We generalized the jitter-earliest-due-date discipline to obtain the class of rate-controlled disciplines (Section 1.5.3). Disciplines in this class are composed from a rate regulator and a scheduler. They can provide a variety of performance bounds by appropriate choice of these components.

Réseaux,
Protocoles et applications de l'Internet

INF 586

Walid Dabbous
INRIA Sophia Antipolis

Contenu du cours

- Introduction: le téléphone et l'Internet.
- Les liens de communication et l'accès multiple
- Adressage et routage point à point dans l'Internet
- Contrôle de transmission
- Architecture de protocoles
- Communication de groupe
- Support de la qualité de service dans l'Internet

Références

- Cours inspiré (surtout) du livre de S. Keshav
- An Engineering Approach to Computer Networking, S. Keshav, Addison-Wesley, May 1997, 688 pages, ISBN 0-201-63442-2
- Plusieurs autres livres de référence
- Routing in the Internet, C. Huitema, Prentice-Hall, 1995, 319 pages, ISBN 0-13-132192-7
- Data and Computer Communications, W. Stallings, Prentice Hall International Editions, 6th edition, 2000, 810 pages, ISBN 0-13-086388-2
- Computer Networking, A Top-Down Approach Featuring the Internet, J. Kurose, K. Ross, Pearson Education, 2001, 712 pages, ISBN 0-201-47711-4
- Computer Networks: A Systems Approach, Larry L. Peterson, Bruce S. Davie, Morgan Kaufmann, April 1996, 500 pages, ISBN 1-55-860368-9
- Computer Networks, Andrew S. Tanenbaum, Prentice Hall International Editions, 3rd edition, March 1996, 814 pages, ISBN 0-13-394248-1
- Data Networks, Dimitri P. Bertsekas, Robert Gallager, Prentice Hall, 2nd edition, December 1991, 556 pages, ISBN 0-13-200916-1
- Internetworking with TCP/IP Volume 1: Principles, Protocols, and Architecture, D. E. Comer, Prentice-Hall, 3rd edition, 1995, 613 pages, ISBN 0-13-216987-8

Références (suite)

- Computer Networks and Internets, D. E. Comer, Prentice-Hall, 3rd edition, 2001, 703 pages, ISBN 0-13-091449-5
- Systèmes multimédias communicants, W. Dabbous éditeur, Hermès Science Publications, 2001, 320 pages, ISBN 2-7462-0251-4
- Interconnections: Bridges and Routers, Radia Perlman, Addison-Wesley, May 1992, 400 pages, ISBN 0201563320
- Multicast Networking And Applications, Kenneth C. Miller, Addison-Wesley, 1999, 282 pages, ISBN 0-201-30979-3
- MobileIP: Design Principles and Practices, C. E. Perkins, Addison-Wesley, 1997, 275 pages, ISBN 0-201-63469-4
- TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens, Addison-Wesley, Published January 1994, 600 pages, ISBN 0201633469
- TCP/IP Illustrated, Volume 2: The Implementation W. Richard Stevens, Wright, Gary R., Addison-Wesley, Published January 1995, 832 pages, ISBN 020163354X
- TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the Unix Domain Protocols, W. Richard Stevens, Gary R. Wright, Addison-Wesley, Hardcover, Published January 1996, 325 pages, ISBN 0201634953
- Advanced programming in the UNIX environment, Richard Stevens, Addison-Wesley, 1992, 768 pages, ISBN 0-201-56317-7
- UNIX network programming, W Richard Stevens, Prentice Hall, 1998, 1240 pages, ISBN 0-13-490012-X

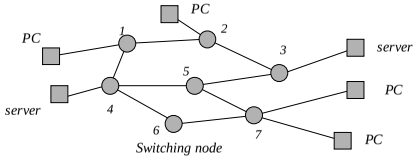
Plan du cours d'aujourd'hui - bloc 1
Le téléphone et l'Internet

- Les réseaux commutés
- Le réseau téléphonique
  - commutation de circuits
- Le réseau ATM
  - commutation de cellules - circuits virtuels
- L'Internet
  - commutation de paquets - datagrammes

Les réseaux commutés

Beyond local area networks

- End systems (stations) send data through a network of intermediate switching nodes
- Some nodes connect only to other nodes (routers, switches)
- usually the network is not fully connected
  - but more that one path from source to destination



Le réseau téléphonique

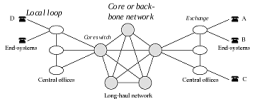
Is it a computer network?

- Specialized to carry voice (more than a billion telephones worldwide)
- But also carries
  - fax
  - modem calls
  - video
- Internally, uses digital *samples*
- Standard end-system/network interface
- Switches and switch controllers are special purpose computers
- Principles in its design apply to more general computer networks

Concepts

- Single basic service: two-way voice
  - low end-to-end delay
  - guarantee that an accepted call will run to completion
- Endpoints connected by a *circuit*
  - like an electrical circuit
  - signals flow both ways (*full duplex*)
  - associated with bandwidth and buffer *resources*

The big picture



- (nearly) Fully connected core
  - simple routing
  - hierarchically allocated telephone number space
  - telephone number is a hint about how to route a call
    - but not for 800/888 (toll-free) / 700 (AT&T Incoming call forwarding) / 900 (pay-per-call) numbers

The components of a telephone network

1. End systems
2. Transmission
3. Switching
4. Signaling

1. End-systems

- Transducers
  - key to carrying voice on wires
- Dialer
- Ringer
- Switchhook at central office interprets tones or pulses
  - place a call
  - or do call forwarding
  - sends ring signal
    - power for ringing provided by central office

Sidetone & Echo

- Transmission circuit needs two wires
- And so does reception circuit
- => 4 wires from every central office to home
- Can we do better?
- Use *same* pair of wires for both transmission and reception
- Two problems: sidetone and echo
  - Sidetone attenuation: balance circuit is required
  - (expensive) Echo cancellation for *long-distance* calls
- Lesson
  - keep end-to-end delays as short as possible

2. Transmission

- Link characteristics
  - information carrying capacity (bandwidth)
    - information sent as *symbols*
    - 1 symbol >= 1 bit (see next course)
  - propagation delay
    - time for electromagnetic signal to reach other end
    - light travels at 0.7c in fiber ~ 5 μs/km
    - Nice to Paris => 5 ms; London to NY => 27 ms ; ~250 ms for earth-sat-earth on GEO satellites
  - attenuation
    - degradation in signal quality with distance
    - long lines need regenerators
    - but recent links need regeneration each 5000 Km and optical amplifiers exist

Transmission: Multiplexing

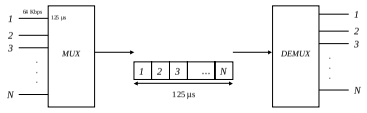
- *Trunks* between central offices carry hundreds of conversations
- Can't run thick bundles!
- Instead, send many calls on the same wire
  - *multiplexing*
- Analog multiplexing (FDM)
  - bandlimit call to 3.4 KHz and frequency shift onto higher bandwidth trunk
  - obsolete, the telephone network is becoming all-digital
- Digital multiplexing
  - first convert voice to *samples*
  - 1 sample = 8 bits of voice
  - 8000 samples/sec => call = 64 Kbps

Transmission: Digital multiplexing

- How to choose a sample?
  - 256 *quantization levels*
    - logarithmically spaced (better resolution at low signal levels)
    - sample value = amplitude of nearest quantization level
  - two choices of quantization levels (μ law (Japan and USA) and A law)
- Time division multiplexing (TDM)
  - (output) trunk carries bits at a faster bit rate than inputs
  - *n* input streams, each with a 1-byte buffer
  - output interleaves samples
  - need to serve all inputs in the time it takes one sample to arrive
  - => output runs *n* times faster than input
  - *overhead* bits mark end of *frame* (synchronize to frame boudary)

Multiplexors and demultiplexors

- Most trunks time division multiplex voice samples
- At a central office, trunk is demultiplexed and distributed to active circuits
- Synchronous multiplexor
  - N input lines (associated with a buffer to store at least one sample)
  - Output runs N times as fast as input



More on multiplexing

- Demultiplexor
  - one input line and N outputs that run N times slower
  - samples are placed in output buffer in round robin order
- Neither multiplexor nor demultiplexor needs addressing information (why?)
  - requires however accurate timing information
- Can cascade multiplexors
  - need a standard
  - example: DS hierarchy in the US and Japan

Digital Signaling hierarchy

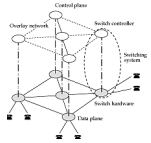
Digital Signal Number	Number of previous level circuits	Number of voice circuits	Bandwidth
DS0		1	64 Kbps
DS1 - T1	24	24	1.544Mbps
DS2	4	96	6.312 Mbps
DS3 - T3	7	672 = 28 T1	44.736 Mbps

Inverse multiplexing : scatter/gather

- Takes a high bit-rate stream and scatters it across multiple trunks
- At the other end, combines multiple streams
  - resequencing to accommodate variation in delays
- Allows high-speed virtual links using existing technology
  - aggregate telephone channels to connect IP routers

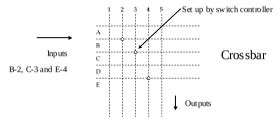
3. Switching

- Problem:
  - each user can potentially call any other user
  - can't have direct lines!
- Switches establish temporary *circuits*
- Switching systems come in two parts: switch and switch controller



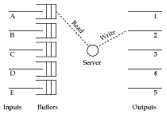
Switching: what does a switch do?

- Transfers data from an input to an output
  - many ports (up to 200,000 simultaneous calls)
  - need high speeds
- Some ways to switch:
  - First way*: space division* (data paths are separated in space)
    - *simplest space division switch is a "crossbar"*
    - if inputs are multiplexed, need a *schedule* (to rearrange crosspoints at each time slot)



Time Division Switching

- Another way to switch
  - *time division (time slot interchange* or *TSI)*
  - also needs (only) a schedule (to write to outputs in correct order)



- Inefficient if long pauses in conversations (idle slots are wasted)
- To build (large) switches we combine space and time division switching larger elements

More details: A circuit switch

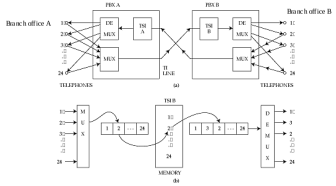
- A switch that can handle N calls has N logical inputs and N logical outputs
  - N up to 200,000
- In practice, input trunks are multiplexed
  - far fewer physical I/O lines
  - example: DS3 trunk carries 672 simultaneous calls
- Multiplexed trunks carry *frames* = set of samples
- Goal: extract samples from frame, and depending on position in frame, switch to output
  - each incoming sample has to get to the right output line and the right slot in the output frame
  - demultiplex, switch, multiplex

Call blocking

- Can't find a path from input to output (reject blocked calls)
- Internal blocking
  - slot in output frame exists, but no path through the switch
- Output blocking
  - no slot in output frame is available (compete for the same output)
- *Line* switch : connect a specific input to a specific output
- *Transit* switch: connect an input to one *several* outputs
- Internal and output blocking is reduced in transit switches
  - need to put a sample in one of *several* slots going to the desired next hop
  - a transit switch acheives same blocking probability as a line switch with less hardware

More on Time division switching

- Key idea: when demultiplexing, position in frame determines output trunk
- Time division switching interchanges sample position within a frame: time slot interchange (TSI)

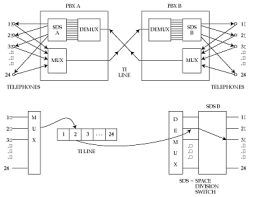


How large a TSI can we build?

- Limit is time taken to read and write to memory
- For 120,000 circuits
  - need to read and write memory 120,000 times every 125 µs (slot duration)
  - each operation takes around 0.5 ns => impossible with current technology
  - with 40-ns memory => 1500 circuits!
- Need to look to other techniques

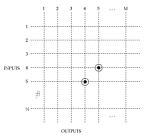
Space division switching

- Each sample takes a different path through the switch, depending on its destination



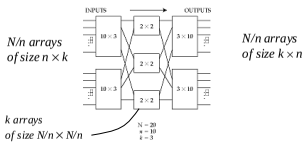
Crossbar

- Simplest possible space-division switch
- *Crosspoints* can be turned on or off
- For multiplexed inputs, need a switching *schedule*
  - as different samples may have different destinations
- Internally nonblocking
  - vulnerable to single faults (only one path between given input output pair)
  - time taken to set crosspoints grows quadratically with N
  - need N^2 crosspoints
- Small switches 8x8 or 64x64



Multistage crossbar

- In a crossbar during each switching time only one crosspoint per row or column is active
- Can *save crosspoints* if a crosspoint can attach to more than one input line
- This is done in a multistage crossbar

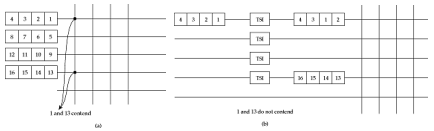


Multistage crossbar

- Can suffer internal blocking
  - unless sufficient number of second-level stages ($k > 2n - 2$)
  - but requires rearranging existing connections as a new call arrives
  - Clos network: *rearrangably nonblocking* switch
- Number of crosspoints < N^2
  - minimize crosspoints for n ~ SQRT(N)
- Finding a path from input to output requires a depth-first-search
  - path stored in switch schedule
- Scales better than crossbar, but still not too well
  - 120,000 call switch needs ~250 million crosspoints
- Unless we accept blocking
  - trade-off between blocking probability and switch cost

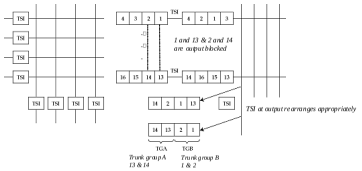
Time-space switching

- Precede each input trunk in a crossbar with a TSI
- "Delay" samples so that they arrive at the right time for the space division switch's schedule
- Allows to build non blocking SDS with fewer crosspoints than a Clos switch



Time-space-time (TST) switching

- Allowed to flip samples both on input and output trunk
- Gives more flexibility => lowers call blocking probability



4. Signaling

- Recall that a switching system has a switch and a switch controller
- Switch controller is in the *control* plane
  - does not touch voice samples
- Manages the network
  - call routing (collect *dialstring* and forward call)
  - alarms (ring bell at receiver)
  - billing
  - directory lookup (for 800/888 calls)

Challenges for the telephone network

- Multimedia
  - simultaneously transmit voice/data/video over the network
  - people seem to want it
  - existing telephone network can't handle it
    - bandwidth requirements
    - *burstiness* in traffic (TSI can't skip input)
      - either peak rate service or very large buffers
    - change in statistical behavior with regard to voice
      - decades of experience for telephone engineers
- Backward compatibility of new services
  - huge existing infrastructure
  - "advantage" of developing countries
- Regulation
  - monopoly stifles innovation

Challenges

- Competition
  - future telephone networks will no longer be monopolies
    - end to good times
  - how to manage the transition?
    - be more responsive to technological innovations
    - at the expense of long term thinking!
- Inefficiencies in the system
  - an accumulation of incompatible systems and formats
  - special-purpose systems of the past (assembly language parts)
  - 'legacy' systems
  - need to change them without breaking the network

Les réseaux ATM

Why ATM networks?

- Different information types require different qualities of service from the network
  - stock quotes vs. USENET
- Telephone networks support a single quality of service
  - and is expensive to boot
- ATM networks are meant to support a range of service qualities at a reasonable cost

Design goals

- Providing “end-to-end” quality of service
- High bandwidth
- Scalability
- Manageability
- Cost-effectiveness

How far along are we?

- Basic architecture has been defined
- But delays have resulted in ceding desktop to IP
- Also, little experience in traffic specification, multicast, and fault tolerance
- We will never see “end-to-end” ATM
  - but its ideas continue to influence design of next-generation Internet - see block 7 (Scheduling)
  - Internet technology + ATM philosophy -- will it work ?
- Note--two standardization bodies
  - ATM Forum
  - International Telecommunications Union-Telecommunications Standardization Sector (ITU-T)

Concepts

1. Virtual circuits
2. Fixed-size packets (*cells*)
3. Small packet size
4. Statistical multiplexing
5. Integrated services

Together

can carry *multiple* types of traffic

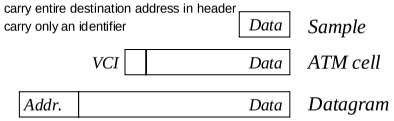
with (ATM) end-to-end quality of service

1. Virtual circuits

- Some background first
- Telephone network operates in *Synchronous Transfer Mode*
  - the destination of a sample depends on where it comes from. Knowing *when* it came is suffcient, no need for a descriptive header
  - example--shared leased link to the same destination
- Problems with STM
  - idle users consume bandwidth
  - Arbitrary schedules result in complicated operation
    - links are shared with a fixed cyclical schedule => quantization of link capacity (corresponds to 64 Kbps circuits in telephone)
    - can't 'dial' bandwidth e.g. 91 Kbps.
  - ➥ STM service is inflexible

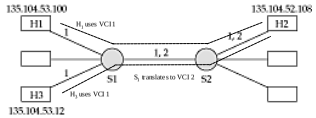
Virtual circuits (contd.)

- STM is easy to overcome
  - use *packets* instead
  - meta-data (header) indicates src/dest
    - allows to store packets at switches and forward them when convenient
    - no wasted bandwidth (identify cell by source address not only order in frame) - more *efficient*
    - arbitrary schedule (cells of same source can occur more than once in frame) - more *flexible*
- Two ways to use packets
  - carry entire destination address in header
  - carry only an identifier



Virtual circuits (contd.)

- Identifiers save on header space
- But need to be pre-established
- We also need to switch Ids at intermediate points
  - VCIs are allocated locally
- Need *translation table* (for VCI swapping) and *connection setup*



Features of virtual circuits

- All packets must follow the same path
  - if any switch along the route fails -> the VC fails
- Switches store per-VC state (entry in translation table)
  - can also store QoS information (priority, reserved bandwidth)
- Call set-up (or signaling) => separation of *data* and *control*
  - control in software over slow time scale, data transfer in hardware
- Virtual circuits do not automatically guarantee reliability
  - possible packet loss
- Small Identifiers can be looked up quickly in hardware
  - harder to do this with IP addresses

More features

- Setup must precede data transfer
  - delays short messages
- Switched vs. Permanent virtual circuits
- Ways to reduce setup latency
  - preallocate a range of VCIs along a path
    - *Virtual Path*
    - *reduces also the size of the translation table*
  - dedicate a VCI to carry datagrams, reassembled at each hop

2. Fixed-size packets

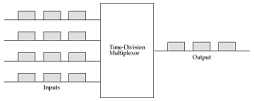
- Pros
  - Simpler buffer hardware
    - packet arrival and departure requires us to manage fixed buffer sizes (easier, no memory fragmentation)
  - Simpler line scheduling
    - each cell takes a constant chunk of bandwidth to transmit -> harder to achieve simple ratios with variable size packets
  - Easier to build large *parallel* packet switches
    - input buffers, parallel switch fabrics, output buffers -> *maximum parallelism if same packet size*
- Cons
  - If the chosen size < ADU => overhead
  - segmentation and reassembly cost
  - last unfilled cell after segmentation wastes bandwidth

3. Small packet size

- At 8KHz, each byte is 125 microseconds
- The smaller the cell, the less an endpoint has to wait to fill it
  - *packetization delay*
- The smaller the packet, the larger the header overhead
- EU and Japan: reduce cell size (32 bytes cell, 4 ms packetization delay)
- US telcos: reduce header cost (existing echo cancellation equipment) (64 bytes cell, 8ms packetization delay)
- Standards body balanced the two to prescribe 48 bytes + 5 byte header = 53 bytes
  - => ATM maximal efficiency of 90.57%

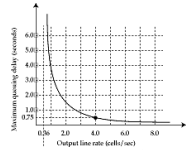
IETF TShirsts

4. Statistical multiplexing



- output rate: 4cells/s. queuing delay <= 3/4s.
- Suppose cells arrive in bursts
  - each burst has 10 cells evenly spaced 1 second apart
  - mean gap between bursts = 100 seconds (average rate = 0.0909 cell/s)
- What should be service rate of output line?
  - No single answer (4c/s? 0.36c/s? 1c/s?)

Statistical multiplexing



- We can trade off *worst-case delay* against *speed of output trunk*
- Statistical Multiplexing Gain = sum of peak input/output rate
  - A cell switch exploits SMG in the same way as a TD multiplexor.
- Whenever long term average rate *differs* from peak, we can trade off service rate for delay (requires buffers for zero loss)
  - key to building packet-switched networks with QoS

Generalized SMG

- *n* bursty source that have *p* peak rate and *a* average rate
- Worst case: simultaneous arrivals -> conservatively serve at *n.p*
- To reduce cost, can serve at *r* with $n.a < r < n.g$
  - Requires buffering -> higher delays
- SMG = *n.p/r*
- general principle:
  - if long-term average rate < peak rate; trade-off service rate for mean delay
- ATM cells can be stored & long distance BW expensive
  - -> SMG applicable
- Not if average rate close to peak rate

5. Integrated service

- Traditionally, voice, video, and data traffic on separate networks
- Integration
  - easier to manage
  - innovative new services (Vconferencing, Venvironments)
- How do ATM networks allow for integrated service?
  - lots of (switching) capacity: hardware-oriented switching
  - support for different traffic types
    - signaling for call set-up
    - admission control, Traffic descriptor, policing
    - resource reservation
    - requires intelligent link scheduling for voice/data integration (more flexible than telephone because of headers)

Challenges

- Quality of service
  - defined, but not used!
  - still needs research
- Scaling
  - little experience
- Competition from other LAN technologies
  - FDDI
  - 100Mbps Ethernet
- Standardization
  - Political (ATM forum is not the IETF)
  - slow

Challenges

- IP
  - a vast, fast-growing, non-ATM infrastructure
  - interoperation is a pain in the neck, because of fundamentally different design philosophies
    - connectionless vs. connection-oriented
    - resource reservation vs. best-effort
    - different ways of expressing QoS requirements
    - routing protocols differ
  - ATM serves as a "leased line" service between IP routers

L'Internet

My how you've grown!

- The Internet has doubled in size every year since 1969
- In 1996, 10 million computers joined the Internet
- By July 1997, 10 million more have joined
- By Jan 2001, 100 million hosts
- By March 2002, 400 million users
- By 2004, 700 to 900 million expected
- Soon, everyone who has a phone is likely to also have an email account

What does it look like?

- Loose collection of networks organized into a multilevel hierarchy
  - 10-100 machines connected to a *hub* or a *router*
    - service providers also provide direct dialup access
    - or over a wireless link
  - 10s of routers on a *department backbone*
  - 10s of department backbones connected to *campus backbone*
  - 10s of campus backbones connected to *regional service providers*
  - 100s of regional service providers connected by *national backbone*
  - 10s of national backbones connected by *international trunks*

Example of message routing

```
# traceroute parmesan.cs.wisc.edu (three probes at each TTL value)
traceroute to parmesan.cs.wisc.edu (128.105.167.16), 30 hops max, 38 byte packets
 1  t4-gw.inria.fr (138.96.32.250)  0.314 ms  0.271 ms  0.332 ms
 2  nice.cssi.renater.fr (195.220.98.117)  7.953 ms  10.770 ms  2.018 ms
 3  nio-n1.cssi.renater.fr (195.220.98.101)  17.489 ms  22.218 ms  14.136 ms
 4  nio-1.cssi.renater.fr (193.51.206.14)  14.080 ms  23.882 ms  18.131 ms
 5  opentransit-nio-i.cssi.renater.fr (193.51.206.42)  22.554 ms  15.353 ms  15.653 ms
 6  P3-0.PASCR2.Pastourelle.opentransit.net (193.251.241.158)  25.020 ms  16.662 ms  20.514 ms
 7  P11-0.PASCR1.Pastourelle.opentransit.net (193.251.241.97)  18.202 ms  15.704 ms  16.216 ms
 8  P12-0.NYKCR2.New-york.opentransit.net (193.251.241.134)  90.137 ms  90.190 ms  89.799 ms
 9  P6-0.NYKBB3.New-york.opentransit.net (193.251.241.238)  96.411 ms  97.740 ms  96.006 ms
10  BBN.GW.opentransit.net (193.251.250.158)  112.554 ms  116.028 ms  110.994 ms
11  p3-0.nycmny1-nbr2.bbnplanet.net (4.24.10.69)  119.815 ms  113.583 ms  108.599 ms
12  * p15-0.nycmny1-nbr1.bbnplanet.net (4.24.10.209)  115.725 ms  115.237 ms
13  so-6-0-0.chcgil2-br2.bbnplanet.net (4.24.4.17)  115.999 ms  124.484 ms  119.278 ms
14  so-7-0-0.chcgil2-br1.bbnplanet.net (4.24.5.217)  116.533 ms  120.644 ms  115.783 ms
15  p1-0.chcgil2-cr7.bbnplanet.net (4.24.8.106)  119.212 ms  117.684 ms  117.374 ms
16  a0.uwisc.bbnplanet.net (4.24.223.22)  123.337 ms  119.627 ms  126.541 ms
17  r-peer-WRNmdison-gw.net.wisc.edu (216.56.1.18)  123.403 ms  127.295 ms  129.175 ms
18  144.92.128.226 (144.92.128.226)  124.777 ms  123.212 ms  131.111 ms
19  144.92.128.196 (144.92.128.196)  121.280 ms  126.488 ms  123.018 ms
20  e1-2.foundry2.cs.wisc.edu (128.105.1.6)  132.539 ms  127.177 ms  122.419 ms
21  parmesan.cs.wisc.edu (128.105.167.16)  123.928 ms *  124.471 ms
```

A closer example

```
# traceroute ultralix.polytechnique.fr
traceroute to ultralix.polytechnique.fr (129.104.11.15), 30 hops max, 38 byte packets
 1  t4-gw.inria.fr (138.96.32.250)  0.550 ms  0.270 ms  0.263 ms
 2  nice.cssi.renater.fr (195.220.98.117)  8.779 ms  6.381 ms  2.391 ms
 3  nio-n1.cssi.renater.fr (195.220.98.101)  19.744 ms  24.804 ms  17.490 ms
 4  nio-n1.cssi.renater.fr (193.51.206.5)  21.975 ms  17.592 ms  13.758 ms
 5  jussieu.cssi.renater.fr (194.214.109.6)  18.938 ms  21.357 ms  15.002 ms
 6  epp-jussieu.cssi.renater.fr (193.51.12.82)  25.117 ms  29.762 ms  21.258 ms
 7  129.104.63.1 (129.104.63.1)  23.580 ms  20.993 ms  25.804 ms
 8  129.104.63.13 (129.104.63.13)  21.973 ms  16.783 ms  23.964 ms
 9  ultralix.polytechnique.fr (129.104.11.15)  19.174 ms *  25.052 ms
```

What holds the Internet together?

- Addressing
  - how to refer to a machine on the Internet
- Routing
  - how to get there
- Internet Protocol (IP)
  - what to speak to be understood at the "inter-network" level

More details : joining the Internet

- How can people talk to you?
  - get an IP address from your administrator
- How do you know where to send your data?
  - if you only have a single external connection, then no problem
  - otherwise, need to speak a routing protocol to decide next hop
- How to format data?
  - use the IP format so that intermediate routers can understand the destination address
- Decentralized and distributed
  - No single authority for addressing
  - No coordination for routing
  - Connectionless IP service
  - scales to millions of hosts

What lies at the heart?

- Two key technical concepts
  - packets
  - store and forward

Packets

- Self-descriptive data
  - packet = data + metadata (header)
- Packet vs. sample
  - samples are not self descriptive
  - to forward a sample, we have to know *where* it came from (in fact order in frame)
  - can't store it!
  - hard to handle bursts of data

Store and forward

- Metadata allows us to forward packets when we want
- E.g. letters at a post office headed for main post office
  - address labels allow us to forward them in batches
- Efficient use of critical resources
  - allows to share the cost of expensive transmission link
- Three problems
  - hard to control delay within network
  - switches need memory for buffers
  - convergence of flows can lead to congestion

Key features of the Internet

- Addressing
- Routing
- Endpoint control

Addressing

- Internet addresses are called IP addresses
- Refer to a *host interface*: need one IP address per interface
- Addresses are structured as a two-part hierarchy
  - network number
  - host number

135.105.53	*100*

An interesting problem

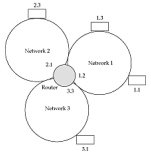
- How many bits to assign to host number and how many to network number?
- If many networks, each with a few hosts, then more bits to network number
- And *vice versa*
- But designer's couldn't predict the future
- Decided three sets of partitions of bits
  - class A: 8 bits network (in fact 7), 24 bits host
  - class B: 16 bits networks (in fact 14), 16 bits host
  - class C: 24 bits network (in fact 21), 8 bits host

Addressing (contd.)

- To distinguish among them
  - use leading bit
  - first bit = 0=> class A, range 1-126 (127 is loopback)
  - first bits 10 => class B, range128-191
  - first bits 110 => class C, range 192-223
- Problem
  - if you want more than 256 hosts in your network, need to get a class B, which allows 64K hosts => wasted address space
- Solution
  - associate *every* address with a *mask* that indicates partition point
  - *CIDR (Classless InterDomain Routing)*
- What about IPv6?

Routing

- How to get to a destination given its IP address?
- We need to know the next hop to reach a particular network number
  - this is called a *routing table*
  - computing routing tables is non-trivial (distributed routing protocol)
- Simplified example



Default routes

- Strictly speaking, need next hop information for every network in the Internet
  - > 80,000 now
- Instead, keep detailed routes only for local neighborhood
- For unknown destinations, use a *default* router
- Reduces size of routing tables at the expense of non-optimal paths

Endpoint control - the end2end argument

- Key design philosophy
  - do as much as possible at the endpoint
  - dumb network
  - exactly the opposite philosophy of telephone network
- Layer above IP compensates for network defects
  - Transmission Control Protocol (TCP)
- Can run over any available link technology
  - but no quality of service
  - modification to TCP requires a change at every endpoint
  - telephone network technology upgrade transparent to users
    - cellular phone introduction does not require fixed telephones upgrade

Challenges

- IP address space shortage
  - because of free distribution of inefficient Class B addresses
  - decentralized control => hard to recover addresses, once handed out
- Decentralized control
  - allows scaling, but makes *reliability* next to impossible
  - cannot “guarantee” that a route exists
  - Corrupted routing messages can cause a major disaster
  - Non-optimal routing
    - each administrative makes a locally optimal decision

Challenges (contd.)

- Decentralized control (contd.)
  - hard to guarantee security
    - end-to-end encryption is a partial solution
    - requires scalable and efficient key distribution scheme
  - no equivalent of white or yellow pages
    - hard to reliably discover a user's email address
  - no uniform solution for accounting and billing
    - can't even reliably identify individual users
    - password based identification does not “scale”
    - -> flat rate billing

Challenges (contd).

- Multimedia
  - requires network to support quality of service of some sort
    - hard to integrate into current architecture
    - store-and-forward => shared buffers => traffic interaction => hard to provide service quality
  - requires endpoint to signal to the network what it wants
    - but Internet does not have a simple way to identify streams of packets
    - nor are routers required to cooperate in providing quality
    - and what about pricing!
  - However, basic Internet multimedia applications exist today

Les liens de communication et l'accès multiple

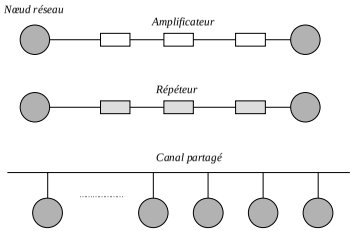
Bloc 2, INF 586

Walid Dabbous
INRIA Sophia Antipolis

Contenu du bloc

- La couche physique
  - notions de base sur la transmission
    - numérique
    - analogique
- L'accès multiple
  - les techniques de base
  - les protocoles
- Le contrôle de liaison
  - fera partie du 4ème bloc

Portée de ce bloc



La couche physique

Transmission: some definitions

- Data: entities that convey information
- Analog: take continuous values in some interval
- Digital: take discrete values
- Analog data: voice, video, temperature, etc.
- Digital data: character strings, digitized audio or video, etc.
- Signal: Electric or electromagnetic representation of data
- Signaling (here): physical propagation of the signal along suitable medium
- Transmission: communication of data by the propagation and *processing* of signals

More definitions

- Analog signal: a continuously varying electromagnetic wave that may be propagated over a variety of (wired or wireless) media
- Digital signal: a discrete or discontinuous signal such as voltage pulses that may be propagated over a *wired* medium
- Wired media: guided transmission media e.g. twisted pair, coaxial cable, optical fiber (only transmits analog signal-encoded beam of light)
- Wireless (unguided) media: radio, terrestrial or satellite microwave, infrared
- Analog signaling: propagation of analog signal
- Digital signaling: propagation of digital signal
- Analog transmission: a means of transmitting analog signals without regard to their content (analog or digital data); amplifiers are used to boost the (analog) signal.

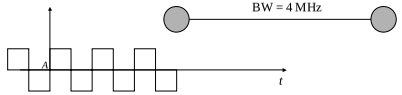
More definitions

- Digital transmission: the transmission of digital data, using either an analog or a digital signal, in which the digital data are recovered and repeated at intermediate points (repeaters) to reduce the effects of noise.
- coder (sampler+quantizer+digitize): transforms analog data/signal to a digital signal
- Modulator : transforms a digital bit stream into an analog signal suitable for transmission over analog media
- absolute bandwidth : width of the frequency spectrum ; F_{max}-F_{min}
- bandwidth: frequency band where « most » of the signal energy is contained (e.g. half-power bandwidth, 3dB below peak value)

Data rate and signal elements

- Signal element: the part of a signal that occupies the shortest interval of a signaling code
  - Digital signal element: a voltage pulse of constant amplitude
  - Analog signal element: a pulse of constant frequency, phase and amplitude
- Data element: a single binary one or zero (bit).
- Data rate (R in bps): rate at which data elements are transmitted
- Signaling or « modulation » rate (D in baud): rate at which signal elements are transmitted.
- Multilevel signal modulation techniques: reduces D because each signal element represents b=Log_2(number of levels) bits.

Relation between data rate and bandwidth (an example)

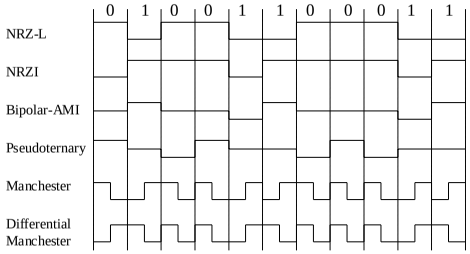


- Square wave s(t) with amplitudes A and -A, period T=1/f
- $s(t) = A.4/\pi.\sum 1/k.\sin(2\pi kf.t)$ (k odd, k=1->inf)
- limit bandwidth to first three frequency components, T = 1 µs
  - Bw = 5MHz -1MHz=4MHz, R = 2 Mbps (1 bit every 0.5 µs)
- limit to first two frequency components, T=0.5 µs
  - Bw = 6MHz - 2MHz = 4MHz, R = 4Mbps (1 bit every 0.25 µs)
- Theortically unbounded, but economical and practical limitations -> distortion

Digital encoding techniques - comparison criteria

- Encoding scheme: mapping of data elements to signal elements
- Signal spectrum
  - lack of high frequency and dc components
  - concentrated spectral density
- Clocking
  - provide synchronisation with suitable encoding
- Error detection
  - built in error detection, faster than data link control
- signal interference and noise immunity
- Cost and complexity (increases with signalling rate)

Digital signal encoding formats



Encoding digital data to digital signals

- NonReturn to Zero-Level (NRZ-L): 0=+V, 1= -V
- NRZ Invert on 1s (NRZI): 0=no transition, 1=transition (differential encoding)
- Bipolar-AMI: 0=no line signal, 1= positive or negative level alternating for successive ones
- Pseudoternary: positive or negative level alternating for successive zeros, 1=no line signal
- Manchester: 0=transition from high to low in middle of bit interval, 1=transition from low to high
- Differential manchester (always a transition in middle of interval): 0: transition at beginning of interval, 1=no transition
- HDB3: based on Bipolar-AMI (four zeros replaced by code violation (signal pattern not allowed))

Encoding schemes characteristics

- NRZI uses differential encoding
  - to detect a transition is easier to compare to a threshold in presence of noise
  - immune to (accidental) polarity inversion
- NRZ codes are easy to enginneer and most energy is between DC and half the bit rate
  - but dc component and lack of synchronization
- Multilevel binary (Alternate Mark Inversion) (mark means 1, space means 0)
  - no dc component & provides better synchronization
    - however long string of 0s (AMI) or 1s (pseudoternary) still a pb
  - less efficient than NRZ (1 signal element carries 1 bit instead of $\log_2 3$ bits)
- Biphase
  - always a transition -> « self clocked » mechanism
  - no dc component, but bandwidth wider than multilevel binary

Encoding schemes characteristics

- Scrambling techniques (e.g. HDB3)
  - no dc component, no long sequence of zero-level line signals
  - no reduction in data rate (suitable for long distance transmission)
  - error detection capabilities
- Normalized signal transition rate of various schemes D =R/b

	Min. Mod Rate	101010…	Max. Mod Rate
NRZ-L	0 (all 0s or 1s)	1.0	1.0
NRZI	0 (all 0s)	0.5	1.0 (all 1s)
Bipolar-AMI	0 (all 0s)	1.0	1.0
Pseudotemary	0 (all 1s)	1.0	1.0
Manchester	1.0 (1010..)	1.0	2.0 (all 0s or 1s)
D-Manchester	1.0 (all 1s)	1.5	2.0 (all 0s)

Digital data to analog signals

- For transmission over the public telephone or microwave links
- Three encoding or modualtion techniques
- Amplitude shift keying (ASK):
  - $s(t) = A.\cos(2\pi f_1 t)$ for binary 1, 0 for binary 0
- Frenquency shift keying (FSK):
  - $s(t) = A.\cos(2\pi f_1 t)$ for binary 1, $A.\cos(2\pi f_2 t)$ for binary 0
- Phase shift keying (PSK):
  - $s(t) = A.\cos(2\pi f_c t + \pi)$ for binary 1, $A.\cos(2\pi f_c t)$ for binary 0
- QPSK (signal element represents 2 bits)
  - $s(t) = A.\cos(b_1 b_2 \pi).\cos(2\pi f_c t + 5\pi/4 + (b_2 - b_1)\pi/2)$ for binary $b_1 b_2$

Digital transmission of analog data

- Digitize data then either:
  - transmit using NRZ-L
  - encode as a digital signal using code other than NRZ-L
  - convert digital data into an *analog* signal using modulation
    - allows digital transmission of voice on analog media e.g. microwave
- PCM
- Delta Modulation

Analog transmission

- Modulation for analog signals is useful
  - higher frequency needed for effective transmission (unguided transmission), baseband signals would require huge antennas
  - allows Frequency division multiplexing
- Input signal $m(t) = n_a.x(t)$
- Amplitude Modulation
  - $s(t) = [1 + m(t)] \cos(2\pi f_c t)$
- Angle modulation
  - $s(t) = A_c \cos[2\pi f_c t + \phi(t)]$
  - Phase modulation: $\phi(t) = n_p.m(t)$
  - Frequency modulation: $\phi'(t) = n_f.m(t)$

L’accès multiple

What is it all about?

- Consider an audioconference where
  - if one person speaks, all can hear
  - if more than one person speaks at the same time, both voices are garbled
- How should participants coordinate actions so that
  - the number of messages exchanged per second is *maximized*
  - time spent waiting for a chance to speak is *minimized*
- This is the *multiple access problem*

Some simple solutions

- A (simple) centralized solution: use a moderator
  - a speaker must wait for moderator to "poll" him or her, even if no one else wants to speak
  - what if the moderator's connection breaks?
- Distributed solution
  - speak if no one else is speaking
  - but if two speakers are waiting for a third to finish, guaranteed collision
- Designing good schemes is surprisingly hard!

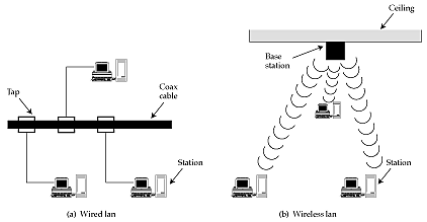
Outline

- Contexts for the problem
- Choices and constraints
- Performance metrics
- Base technologies
- Centralized schemes
- Distributed schemes

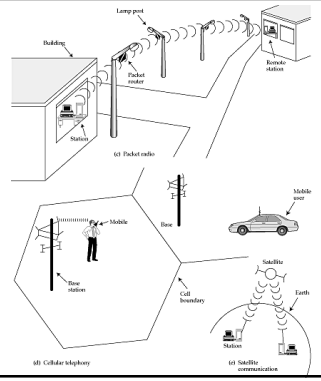
Contexts for the multiple access problem

- *Broadcast* transmission medium
  - message from any "station" is received by all receivers in its listening area
- Colliding messages are garbled
- Goal
  - maximize message throughput
  - minimize mean waiting time
- Shows up in five main contexts

Contexts



Contexts



Solving the problem

- First, choose a *base technology*
  - to isolate traffic from different stations
  - can be in time domain or frequency domain
- But: not enough time slots or frequencies to exclusively dedicate to each user
- Then, choose how to allocate a limited number of transmission resources to a larger set of contending users

Outline

- Contexts for the problem
- Choices and constraints
- Performance metrics
- Base technologies
- Centralized schemes
- Distributed schemes

Choices

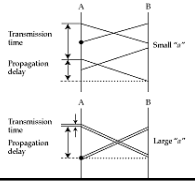
- Centralized vs. distributed design
  - is there a moderator or not?
  - in a centralized solution one of the stations is a *master* (e.g. base station in cellular telephony) and the others are *slaves*
    - master->slave = downlink
    - slave->master = uplink
  - in a distributed solution, all stations are peers
- Circuit-mode vs. packet-mode
  - do stations send steady streams or bursts of packets?
  - with streams, doesn't make sense to contend for every packet
  - allocate resources to streams - analogy to long "speeches"
  - with packets, makes sense to contend for every packet to avoid wasting bandwidth - analogy to brainstorming session

Constraints

- Spectrum scarcity
  - radio spectrum is hard to come by
  - only a few frequencies available for long-distance (few miles) data communication
  - multiple access schemes must be careful not to waste bandwidth
- Radio link properties
  - radio links are error prone
    - fading (signal degradation because of hills, foliage, trucks, etc.)
    - multipath interference
  - hidden terminals
    - transmitter heard only by a subset of receivers
  - capture
    - station with higher power overpowers the other -> no "collision" is detected
    - lower powered station may never get a chance to be heard

The parameter 'a'

- The number of packets sent by a source before the farthest station receives the first bit
- D: max propagation delay between any two stations
- T: time taken to transmit an average size packet
- a = D/T (around 10^{-2} for LANs, 1 for HSLANs, 10^{2} for satellite links)



Outline

- Contexts for the problem
- Choices and constraints
- Performance metrics
- Base technologies
- Centralized schemes
- Distributed schemes

Performance metrics

- Normalized throughput
  - fraction of link capacity used to carry non-retransmitted packets
  - example
    - Ideally (with no collisions) a 1 Mbps link can carry 1000 “125 byte” packets/sec
    - with a particular scheme and workload, we might have 250 packets/sec
    - => goodput = 0.25
- Mean delay
  - amount of time a station has to wait before it successfully transmits a packet
    - depends on the MAC protocol, the load and the characteristics of the medium

Performance metrics

- Stability
  - with heavy load, is all the time spent on resolving contentions?
  - => unstable
  - with a stable algorithm, throughput does not decrease with offered load
  - if huge number of “uncontrolled” stations share a link, then instability is guaranteed
  - but if sources reduce load when overload is detected, can achieve stability
- Fairness
  - no single definition
  - ‘no-starvation’: source eventually gets a chance to send
  - stricter metric: each station gets an equal share of the transmission bandwidth

Limitation of analytical modeling

- Assumptions on source traffic
  - work of queuing theory is only indicative and not an absolute metric
- Other factors may have greater impact than what models predict
  - battery strength
  - weather conditions
  - presence or absence of foliage or plants
  - workload different from assumptions
- Will only focus on *qualitative* description

Outline

- Contexts for the problem
- Choices and constraints
- Performance metrics
- Base technologies
- Centralized schemes
- Distributed schemes

Base technologies

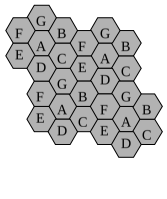
- Isolates data from different sources
- Three basic choices
  - Frequency division multiple access (FDMA)
  - Time division multiple access (TDMA)
  - Code division multiple access (CDMA)

FDMA

- Simplest, best suited for analog links
- Each station has its own frequency band, separated by guard bands
- Receivers tune to the right frequency
- Common for TV and radio broadcast (at most few hundred transmitters in a listening area) - Not in wireless telephony
- Number of frequencies is limited
  - reduce transmitter power; reuse frequencies in non-adjacent cells requiring “complex” handoff
  - trade off complexity for increased number of users
- simplistic example: voice channel = 30 KHz
  - 833 channels in 25 MHz band
  - with 7-cell pattern, partition into 119 channels each
  - but with N cells in a city, can get 119N calls => win if pattern repeated

GSM frequency reuse

- Pattern: smallest cell set containing only once all radio channels (repeated on all zones to cover)
- Regular pattern size: 3,4,7,9, 12 & 21
- D_{ru} is re-use distance
- R is cell radius
- K is pattern size (7 in this example)
- $D_{ru}/R = \sqrt{3.K}$



TDMA

- All stations transmit data on same frequency, but at different times
- Needs time synchronization
- supposes that stations resolve contention for access to a time *slot* and limit their transmission to a single slot
- roughly same number of users than FDMA
- Pros
  - users can be given different amounts of bandwidth (time slots)
  - mobiles can use idle times to determine best base station (and handoff to nearer base station to save battery)
  - can switch off power when not transmitting
- Cons
  - synchronization overhead (one of the stations emit sync signal)
  - greater problems with multipath interference on wireless links (because of wider frequency band -> smaller transmission duration -> need for adaptive equalization circuit in each receiver)

CDMA

- Users separated both by time and frequency
  - Aka spread spectrum techniques
- Colliding “frames” are not necessarily totally garbled
- Send at a different frequency at each time slot (*frequency hopping*)
- Or, convert a single bit to a code (*direct sequence*)
  - receiver can decipher bit by inverse process even in presence of narrowband noise
  - assumes that multiple signals add linearly

CDMA/DS example

- Each station has its « chip-sequence » or codeword (bipolar repsentation)
  - EA: (-1-1-1+1+1-1+1+1) (binary 00011011)
  - EB: (-1-1+1-1+1+1+1-1) (binary 00101110)
  - EC: (-1+1-1+1+1+1-1-1) (binary 01011100)
  - ED: (-1+1-1-1-1-1+1-1) (binary 01000010)
- Send codeword E for a 1 *bit* and its negation -E for a 0 *bit*

A	B	C	D
-	-	1	-
-	1	1	-
1	0	-	-
1	0	1	-
1	1	1	1
1	1	0	1

E1= (EC)	(-1 +1 -1 +1 +1 +1 -1 -1)
E2 = (EB+EC)	(-2 0 0 0 +2 +2 0 -2)
E3 = (EA-EB)	(0 0 -2 +2 0 -2 0 +2)
E4 = (EA-EB+EC)	(-1 +1 -3 +3 -1 -1 -1 +1)
E5 = (EA+EB+EC+ED)	(-4 0 -2 0 +2 0 +2 -2)
E6 = (EA+EB-EC+ED)	(-2 -2 0 -2 0 -2 +4 0)

E1.EC	(1+1+1+1+1+1+1+1)/8 = 1
E2.EC	(2+0+0+0+2+2+0+2)/8 = 1
E3.EC	(0+0+2+2+0-2+0-2)/8 = 0
E4.EC	(1+1+3+3+1-1+1-1)/8 = 1
E5.EC	(4+0+2+0+2+0-2+2)/8 = 1
E6.EC	(2-2+0-2+0-2-4-0)/8 = -1

CDMA

- Pros
  - hard to spy
  - immune from narrow band noise
  - Unlike TDMA, no need for all stations to synchronize (only sender and receiver)
  - no hard limit on capacity of a cell, but noise increases and effective bit-rate per station decreases with the number of stations
  - all cells can use all frequencies, no need for frequency planning
- Cons
  - implementation complexity (receiver perfectly synchronized with senders)
  - need for complicated power control to avoid *capture*
    - hard because of multiple moving receivers
  - need for a large contiguous frequency band (for direct sequence) -> problems installing in the field

FDD and TDD

- Two ways of converting a wireless medium to a duplex channel
- In Frequency Division Duplex, uplink and downlink use different frequencies
- In Time Division Duplex, uplink and downlink use different time slots
- Can combine with FDMA/TDMA
- Examples
  - TDD/FDMA in second-generation cordless phones
  - FDD/TDMA/FDMA in digital cellular GSM phones

Outline

- Contexts for the problem
- Choices and constraints
- Performance metrics
- Base technologies
- Centralized schemes
- Distributed schemes

Centralized access schemes

- One station is master, and the other are slaves
  - slave can transmit only when master allows
- Natural fit in some situations
  - wireless LAN, where base station is the only station that can see everyone
  - cellular telephony, where base station is the only one
    - with a wired connection to the network and
    - capable of high transmit power

Centralized access schemes

- Pros
  - simple
  - master provides single point of coordination
- Cons
  - master is a single point of failure
    - need a re-election protocol => complicates the system!
    - master is involved in every single transfer => added delay
- Circuit mode schemes: cellular telephony
- Packet mode schemes: polling/probing and reservation

Circuit mode: the cellular telephony example

- When station wants to transmit, it sends a message to master using simple (ALOHA) packet mode multiple access protocol
- Master allocates transmission resources to slave
- Slave uses the resources until it is done
- No contention during data transfer
- Used primarily in cellular phone systems
  - EAMPS: analog FDD/FDMA
  - GSM : FDD/TDMA/FDMA
  - IS-95: CDMA

Polling and probing

- Packet-mode: station must contend for medium access for each packet
- Centralized controller mediates this contention
- Polling
  - master asks each station in turn if it wants to send (roll-call polling)
  - inefficient if (a) time to query a station is long, (b) overhead for polling messages is high, or (c) system has many terminals
- Probing
  - stations are numbered with consecutive logical addresses
  - assume station can listen both to its own address and to a set of multicast or "group" addresses
  - master does a binary search to locate next active station
    - skip chunks of address space with no active station
    - But repeated polls in sections with more than one active station
  - Efficient if few stations are active, doubles polls if all active

Reservation-based schemes

- When 'a' is large, collisions are expensive
  - polling overhead is too high
  - better use reservation than polling
  - mainly for satellite links
- Master coordinates access to link using reservations
- Some time slots devoted to reservation messages
  - can be smaller than data slots => *minislots*
- Stations contend for a minislot (PDAMA) (or own one FPODA)
- Master decides winners and grants them access to link
- Packet collisions are only for minislots, so overhead on contention is reduced

Outline

- Contexts for the problem
- Choices and constraints
- Performance metrics
- Base technologies
- Centralized schemes
- Distributed schemes

Distributed schemes

- Compared to a centralized scheme
  - more reliable
  - have lower message delays
  - often allow higher network utilization
  - but are more complicated
    - e.g. to synchronize stations to the same time base
- Almost all distributed schemes are packet mode
  - difficult to establish and maintain circuits without a central controller (the one-time coordination is amortized over many packets for circuit-mode in centralized schemes)

Decentralized polling

- Just like centralized polling, except there is no master
- But, all stations must share a time base
- Each station is assigned a slot that it uses
  - if nothing to send, slot is wasted
  - this is just TDMA :-)

Decentralized probing

- Also called *tree based multiple access*
- All stations in left subtree of root allowed to place packet on medium in first slot
- If a collision, root <- root ->left_son, and try again in next slot
- On success, everyone in root->right_son contend for access etc.
- Works well if 'a' is small
  - otherwise: either introduce idle time to wait for possible collision (inefficient) or roll back state if collision detected later (complex)

Carrier Sense Multiple Access (CSMA)

- Polling/probing may waste time if number of stations is large but number of simultaneously active stations is small
- A fundamental advance: check whether the medium is active before sending a packet (i.e *carrier sensing*)
- Unlike polling/probing a node with something to send doesn't have to wait for a master, or for its turn in a schedule
- If medium idle, then can send
  - just like a participant in a meeting
- If collision happens, detect and resolve
- Works when 'a' is small (0.1 or smaller)
- In slotted version, time *slot* is chosen to be the maximum *propagation* delay (considered small comparing to T)

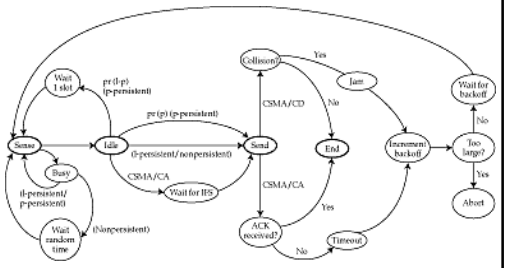
Simplest CSMA scheme

- Send a packet as soon as medium becomes idle
- If, on sensing busy, wait for idle -> *persistent*
- If, on sensing busy, set a timer and try later -> *non-persistent*
- Problem with persistent: two stations waiting to speak will collide

How to solve the collision problem

- Two solutions
- *p-persistent*: when media becomes idle, transmit with probability *p:*
  - hard to choose *p* (< 1/Number of stations waiting)
    - balance message delay with higher utilization under heavy loads
  - if *p* small, then wasted time (if media idle)
  - if *p* large, more collisions
- *exponential backoff*
  - need to detect collisions: explicit ACK or *collision detect circuit* => CSMA/CD
  - on collision detection, choose retransmission timeout randomly from doubled range; on success reset timeout value
  - backoff range adapts to number of contending stations
  - no need to choose *p* (even 1-persistent CSMA with backoff is stable)

Summary of CSMA schemes



Ethernet

- The most widely used LAN
- Standard is called IEEE 802.3
- Uses 1-persistent CSMA/CD with exponential backoff
- Also, on collision, place a *jam* signal on wire, so that all stations are aware of collision and can increment backoff timeout range
- 'a' small =>time wasted in collision is around 50 microseconds
- Ethernet requires packet to be long enough that a collision is detected before packet transmission completes (a <= 1)
  - packet should be at least 64 bytes long for longest allowed segment
- Max packet size is 1500 bytes
  - prevents hogging by a single station

More on Ethernet

- First version ran at 3 Mbps
- Early versions ran at 10 Mbps, and uses 'thick' or 'thin' coax, or twisted pair
- Ethernet types are coded as <Speed><Baseband or broadband><physical medium>
  - Speed = 3, 10, 100 Mbps
  - Baseband = within building, broadband = on cable TV infrastructure
  - Physical medium:
    - "5" is thick coax, up to 500 meters
    - "2" is cheap 50 Ohm cable, up to 200 meters
    - "T" is unshielded twisted pair (also used for telephone wiring)

Enhancing Ethernet

- Ease of maintenance
  - use a hub as in 10BaseT
    - add/remove
- Increase performance
  - divide in multiple « contention domains »
    - use bridges
    - or (even) switches : Switched Ethernet
  - increase speed
    - 100BaseT

Some definitions

- Contention or collision domain: sum total of devices that compete with each other for access to the transmission media
- Hub: a centrally-located device in a star topology that propagates the signal transmitted by each node to ALL other ports. Nodes still constitute a single contention domain. Collision is detected by simultaneous activity on the Data Out (DO) and Receive Data (RD) circuits
- Bridge: device connecting « segments » with level 2 filtering capability. Splits LAN to N contention domains (N=number of ports). Packets are usually stored then forwarded.
- Switch: a « bridge » with N=number of nodes. If switch is full duplex capable, no collision will occur. Each pair gets 20Mbps

Recent developments

- Switched Ethernet
  - each station is connected to switch by a separate UTP wire
    - as in 10BaseT
  - however, line card of switch has a buffer to hold incoming packets
  - fast backplane switches packet from one line card to others
  - simultaneously arriving packets do not collide (until buffers overflow)
  - higher intrinsic capacity than 10BaseT (and more expensive)

Comparison

	Topology	Cable	Max distance	Nodes	Advantages	Half/DpX/FDX
10Base5	bus	Thick coax	500 x5seg	100/seg	backbones	HDX
10Base2	bus	Thin coax	185x5seg	30/seg	cheap	HDX
10BaseT	logical bus	2 UTPs	100 (Hub to Node)	few100s/CD	maintenance	HDX
Bridged	scalable	"bus" / port	No	per segment	multiple CDs	HDX in each CD
Switched	crossbar	UTPs	100 (btw swt.)	__	No contention	FDX

Fast Ethernet variants

- Fast Ethernet (IEEE 802.3u)
  - same as 10BaseT, except that line speed is 100 Mbps
  - spans only 205 m
  - big winner
  - most current cards support both 10 and 100 Mbps cards (10/100 cards) for about $80
- 100VG Anylan (IEEE 802.12)
  - station makes explicit service requests to master
  - master schedules requests, eliminating collisions
  - not a success in the market
- Gigabit Ethernet
  - aims to continue the trend
  - works over 4-pair UTP
- 10Gigabit Ethernet
  - No CSMA, only over optical fiber

Evaluating Ethernet

- Pros
  - easy to setup
  - requires no configuration
  - robust to noise
- Problems
  - at heavy loads, users see large delays because of backoff
  - non-deterministic service
  - doesn't support priorities
  - big overhead on small packets
- But, very successful because
  - problems only at high load
    - loads rarely exceed 30%
  - can segment LANs to reduce load

CSMA/CA

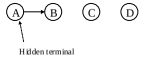
- Used in wireless LANs
- Can't detect collision because transmitter overwhelms colocated receiver
- So, need explicit acks
- But this makes collisions more expensive
  - => try to reduce number of collisions
- Standardized as IEEE 802.11

CSMA/CA algorithm

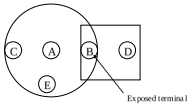
- First check if medium is busy
- If so, wait for medium to become idle
- if idle: wait for inter-frame spacing before contending for a slot (low IFS means higher priority)
- then, set a *contention timer* to an interval randomly chosen in the range [0, CW] (CW predefined contention window)
- On timeout, send packet and wait for ack
- If no ack, assume packet is lost
  - try again, after doubling CW
- If another station transmits while counting down, freeze CW and unfreeze when packet completes transmission
- station will get higher priority in next round of contention

Dealing with hidden terminals

- CSMA/CA works when every station can receive transmissions from every other station
- Not always true
- Hidden terminal
  - some stations in an area cannot hear transmissions from others, though base can hear both (C cannot sense A is sending to B)
- Exposed terminal
  - some (but not all) stations can hear transmissions from stations not in the local area (B should be able to send to D, while A sending to C)



Hidden terminal



Exposed terminal

Dealing with hidden and exposed terminals

- In both cases, CSMA/CA doesn't work
  - with hidden terminal, collision because carrier not detected
  - with exposed terminal, idle station because carrier incorrectly detected
  - what matters is collision "at the receiver"
- Two solutions
- Busy Tone Multiple Access (BTMA)
  - assumes symmetric wireless links
  - uses a separate "busy-tone" channel
  - when station is receiving a message, it places a tone on this channel
  - everyone who might want to talk to a station knows that it is busy
    - even if they cannot hear transmission that this station hears
  - this avoids both problems of hidden and exposed terminals
    - transmitters ignore their carrier-sense circuit and sends only if busy-tone channel is idle

Multiple Access Collision Avoidance

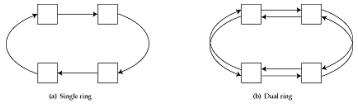
- BTMA requires us to split frequency band
  - more complex receivers (need two tuners)
- Separate bands may have different propagation characteristics
  - scheme fails!
- Instead, use a single frequency band, but use explicit messages to tell others that receiver is busy
- In MACA, before sending data, send a Request to Sent (RTS) to intended receiver
- Station, if idle, sends Clear to Send (CTS)
- Sender then sends data
- If station overhears RTS, it waits for other transmission to end
- Solves both problems

Token passing

- In distributed polling, every station has to wait for its turn
- Time wasted because idle stations are still given a slot
- What if we can quickly skip past idle stations?
- This is the key idea of token ring
- Special packet called 'token' gives station the right to transmit data
  - analogy with "right to speak or microphone"
- When done, it passes token to 'next' station
  - => stations form a logical ring
- No station will starve
- In addition, stations no longer need precise time synchronization

Logical rings

- Can be on a non-ring physical topology


(a) Single ring

(b) Dual ring

(c) Token bus

(d) Hub or star-ring

Ring operation

- During normal operation, copy packets from input buffer to output
- If packet is a token, check if packets ready to send
- If not, forward token
- If so, delete token, and send packets
- Receiver copies packet and sets 'ack' flag
- Sender removes packet from the ring
- When done, reinserts token
- If ring idle and no token for a long time, regenerate token

Single and double rings

- With a single ring, a single failure of a link or station breaks the network => fragile
- With a double ring, on a failure, go into *wrap mode*
- Used in FDDI

Hub or star-ring

- Simplifies wiring
- Active hub is predecessor and successor to every station
  - can monitor ring for station and link failures
- Passive hub only serves as wiring concentrator
  - but provides a single test point
- Because of these benefits, hubs are practically the only form of wiring used in real networks
  - even for Ethernet

Evaluating token ring

- Pros
  - medium access protocol is simple and explicit
  - no need for carrier sensing, time synchronization or complex protocols to resolve contention
  - guarantees zero collisions
  - can give some stations priority over others
- Cons
  - token is a single point of failure
    - lost or corrupted token trashes network
    - need to carefully protect and, if necessary, regenerate token
  - all stations must cooperate
    - network must detect and cut off unresponsive stations
  - stations must actively monitor network
    - to detect token loss and duplication
    - usually elect one station as *monitor*

Fiber Distributed Data Interface

- FDDI is the most popular token-ring base LAN
- Dual counter-rotating rings, each at 100 Mbps
- Uses both copper (CDDI) and fiber links
- Supports both non-realtime and realtime traffic
  - token is guaranteed to rotate once every Target Token Rotation Time (TTRT)
  - station is guaranteed a *synchronous allocation* within every TTRT
- Supports both *single attached* and *dual attached* stations
  - single attached (cheaper) stations are connected to only one of the rings

ALOHA and its variants

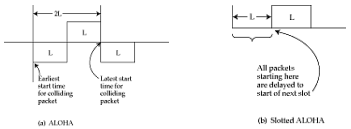
- ALOHA is one of the earliest multiple access schemes
- Just send it!
- Wait for an ack
- If no ack, try again after a random waiting time
  - no backoff

Evaluating ALOHA

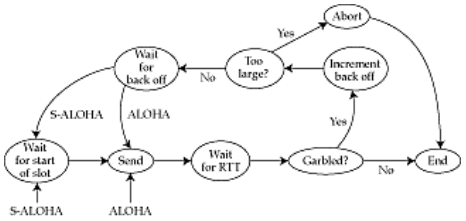
- Pros
  - useful when 'a' is large, so carrier sensing doesn't help
    - satellite links
  - simple
    - no carrier sensing, no token, no timebase synchronization
  - independent of 'a'
- Cons
  - under some mathematical assumptions, goodput is at most .18
    - much higher goodput is achievable (e.g. a single user Trx)
  - at high loads, collisions are very frequent
  - sudden burst of traffic can lead to instability
    - unless backoff is exponential

Slotted ALOHA

- A simple way to double ALOHA's capacity
- Make sure transmissions start on a slot boundary
  - synchronize to a broadcast pulse
- Halves *window of vulnerability*
- Used in cellular phone uplink (to request a frequency)
  - stations need to synchronize for TDMA anyway -> little additional overhead



ALOHA schemes summarized



Reservation ALOHA

- Similar to FPODA but no master, slot time = time to transmit a (constant length) packet
- Contend for implicit reservation using slotted ALOHA
- Stations independently examine reservation requests and come to consistent conclusions
  - all stations have same priority
- Simplest version
  - divide time into frames = fixed length set of slots (spans prop delay)
  - station that successfully transmit in a slot using S-ALOHA (this is known implicitly in satellite Aloha and with explicit ACK with cellular Aloha) makes implicit reservation and can keep slot as long as it wants
  - station that loses keeps track of idle slots and contends for them in next frame

Evaluating R-ALOHA

- Pros
  - supports both circuit and packet mode transfer
  - works with large 'a'
  - simple
- Cons
  - arriving packet has to wait for entire frame before it has a chance to send (frame length is at least 'a')
  - cannot preempt hogs
  - variants of R-ALOHA avoid these problems

Classification of multiple access schemes

Base technologies	FDMA				
	TDMA				
	CDMA				
	FDD				
	CDD				
Access schemes	Centralized	Circuit mode		FAMPS	Cellular telephony
				GSM	
				IS-95	
		Packet mode	Polling and probing		Wired LAN
			Reservation based	FPODA	satellite
				FDAMA	
	Distributed	Packet mode	Polling and probing		Wired LAN
			CSMA	CSMA/CD	
				CSMA/CA	Wireless LAN
			BTMA	MACA	
				MACAW	
			Token ring	FDDI	Wired LAN
			ALOHA	P-ALOHA	satellite
				S-ALOHA	
				R-ALOHA	

Adressage et Routage point à point dans l'Internet

Bloc 3, INF 586

Walid Dabbous
INRIA Sophia Antipolis

Nommage et adressage

Outline

- Naming and Addressing
  - Names and addresses
  - Hierarchical naming
  - Addressing
  - Addressing in the Internet
  - Name resolution
  - Finding datalink layer addresses

Names and addresses

- Names and addresses both uniquely identify a host (or an interface on the host)
- `%nslookup`
  - `Default Server: euryale101.inria.fr`
  - `Address: 138.96.80.222`
  - `> lix.polytechnique.fr`
  - `Name: lix.polytechnique.fr`
  - `Address: 129.104.11.2`
- *Resolution*: the process of determining an address from a name

Why do we need both?

- Names are long and human understandable
  - wastes space to carry them in packet headers
  - hard to parse
- Addresses are shorter and machine understandable
  - if fixed size, easy to carry in headers and parse
- Indirection
  - multiple names may point to same address
  - can move a machine in same domain and just update the resolution table

Hierarchical naming

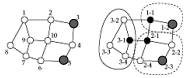
- Goal: give a globally unique name to each host
- Naïve approach: ask every other naming authorities before choosing a name
  - doesn't scale
  - not robust to network partitions
- Instead carve up *name space* (the set of all possible names) into mutually exclusive portions => hierarchy

Hierarchy

- A wonderful thing!
  - simplifies distributed naming
  - guarantees uniqueness
  - scales arbitrarily
- Example: Internet names
  - use *Domain name system (DNS)*
  - global authority (Network Solutions Inc.) assigns top level domains to naming authorities (e.g. .edu, .net, .cz etc.)
  - naming authorities further carve up their space
  - all names in the same domain share a unique *suffix*

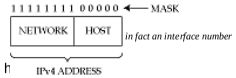
Addressing

- Addresses need to be globally unique, so they are also hierarchical
- Another reason for hierarchy: *aggregation*
  - reduces size of routing tables
    - impractical to have one entry per destination for the Internet
  - at the expense of longer routes



Addressing in the Internet

- Every *host interface* has its own IP address
- Routers have multiple interfaces, each with its own IP address
- Current version of IP is version 4, addresses are IPv4 addresses



- 4 bytes long, two-part h
  - network number and host number
  - boundary identified with a *subnet* mask
  - can aggregate addresses within subnets (network number based routing)

Address classes

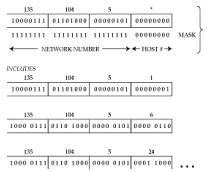
- First cut
  - fixed network-host partition, with 8 bits of network number
  - too few networks!
- Generalization
  - Class A addresses have 8 bits of network number
  - Class B addresses have 16 bits of network number
  - Class C addresses have 24 bits of network number
- Distinguished by leading bits of address
  - leading 0 => class A (first byte < 128)
  - leading 10 => class B (first byte in the range 128-191)
  - leading 110 => class C (first byte in the range 192-223)

Address evolution

- This scheme to allocate *scarce* resources was too *inflexible*
- Three extensions
  - subnetting
  - CIDR
  - dynamic host configuration

Subnetting

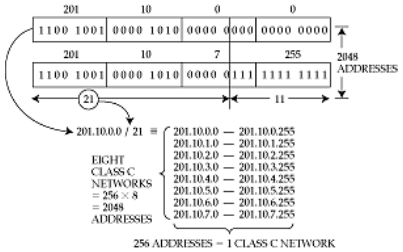
- Allows administrator to cluster IP addresses *within* its network
  - 256 subnet of 256 addresses (e.g. an Ethernet segment)
  - saves space and computation time in subnet routing tables
  - subnet masks are not visible outside the network



CIDR : Classless Interdomain Routing

- Scheme forced medium sized nets to choose class B addresses, which wasted space
- Address space exhaustion (2^{14} = 16382 class B addresses)
- Solution
  - allow ways to represent a contiguous set of class C addresses as a block, so that class C space can be used
  - use a CIDR mask
  - idea is very similar to subnet masks, except that all routers must agree to use it
  - carry a prefix indication: the number of bits of the network number part

CIDR (contd.)



Dynamic host configuration

- Allows a set of hosts to share a pool of IP addresses
- Dynamic Host Configuration Protocol (DHCP)
- Newly booted computer broadcasts *discover* to subnet
- DHCP servers reply with *offers* of IP addresses
- Host picks one and broadcasts a *request* with the name of a particular server
- All other servers "withdraw" offers, and selected server sends an *ack*
- When done, host sends a *release*
- IP address has a *lease* which limits time it is valid
- Server reuses IP addresses if their lease is over (LRU is wise)
- Similar technique used in *Point-to-point* protocol (PPP)
  - to allocate addresses by ISPs to dial-up hosts

IPv6

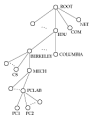
- 32-bit address space is likely to eventually run out
- IPv6 extends size to 128 bits (16 bytes)
- Main features
  - classless addresses (longest prefix match like CIDR)
  - multiple levels of aggregation are possible for *unicast* (IPv6 aggregatable global unicast address RFC[2374])
    - Top level aggregation
    - Next-level aggregation
    - Site-level aggregation
  - several flavors of *multicast*
  - *anycast* (e.g. for partial routes), same syntax as unicast
  - interoperability with IPv4

Name resolution

- Translation done by name servers
- Application send query to a name server:
  - essentially look up a name and return an address
- Centralized design
  - consistent
  - single point of failure
  - concentrates load
- Thus compose name servers from a set of distributed agents
  - that coordinate their action

DNS (Domain Name System)

- Distributed name server
- A name server is responsible (an *authoritative server)* for a set of domains (a subtree of the name space)
- May delegate responsibility for part of a domain to a child
  - things organized so that a name is correctly translated by at least one authoritative Server
- Query is sent to the root of name space
- Parses it and passes to the responsible server

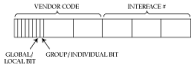


DNS

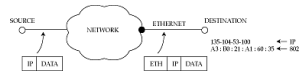
- Heavy load on root servers
  - Root servers are *replicated*
  - requires coordination among servers
  - name resolution query can be made to any replicated server
- *Caching* is also used to reduce load on root servers
- End systems cache and timed out
  - result of the query
  - address of authoritative servers for common domains
- If local server cannot answer a query, it asks root, which delegates reply

Finding datalink layer addresses

- Datalink layer address: most common format is IEEE 802



- Need to know datalink layer address typically for the last hop (in broadcast LANs)



ARP

- To get datalink layer address of a machine on the local subnet
- Broadcast a query with IP dest address onto local LAN
- Host that owns that address (or proxy) replies with address
- All hosts are required to listen for ARP requests and reply
  - including laser printers!
- Reply stored in an ARP cache and timed out
- In point-to-point LANs, need an ARP server
  - register translation with server
  - ask ARP server instead of broadcasting

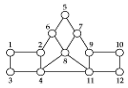
Le routage dans l'Internet

What is it?

- Process of finding a (the best?) path from a source to every destination in the network
- Suppose you want to connect to Antarctica from your desktop
  - what route should you take?
  - does a shorter route exist?
  - what if a link along the route goes down?
- Routing deals with these types of issues

Basics

- A *routing protocol* sets up a *routing table* in *routers* and *switch controllers*



ROUTING TABLE AT 1

Destination	Next hop	Destination	Next hop
1	—	7	2
2	2	8	2
3	2	9	2
4	2	10	2
5	2	11	2
6	2	12	3

- A node makes a *local* choice depending on *global* topology: this is the fundamental problem

Key problem

- How to make correct local decisions?
  - each router must know *something* about global state
- Global state
  - hard to collect
  - inherently large
  - dynamic
- *A routing protocol must intelligently summarize relevant information*

Requirements

- Minimize routing table space
  - fast to look up
  - less to exchange (for some routing protocols)
- Minimize number and frequency of control messages
- Robustness: avoid
  - black holes
  - loops
  - oscillations
- Use optimal path ("best" may be SP, least delay, secure, balances load, lowest monetary cost)
- Trade-offs:
  - robustness vs number of control messages or routing table size
  - reduce table size for slightly "longer" path

Choices

- Centralized vs. distributed routing
  - centralized is simpler, but prone to failure and congestion
- Source-based vs. hop-by-hop (destination address based)
  - how much is in packet header?
  - Intermediate: *loose source route*
- Stochastic vs. deterministic
  - stochastic spreads load, avoiding oscillations, but misorders
- Single vs. multiple path
  - primary and alternative paths (compare with stochastic)
  - not on the Internet (path scarcity and routing table space)
- State-dependent or "dynamic" vs. state-independent
  - do routes depend on current network state (e.g. delay), but risk of oscillations

Outline

- Distance-vector routing
- Link-state routing
- Choosing link costs
- Hierarchical routing
- Internet routing protocols
- Routing within a broadcast LAN

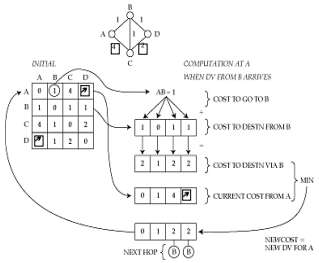
Distance vector routing

- "Internet" environment
  - links and routers unreliable
  - alternative paths scarce
  - traffic patterns can change rapidly
- Two key algorithms
  - distance vector
  - link-state
- Both algorithms assume router knows
  - address of each neighbor
  - cost of reaching each neighbor
- Both allow a router to determine global routing information by exchanging routing information

Basic idea for DV

- Node tells its neighbors its best idea of distance to *every* other node in the network (node identities considered known a priori)
- Node receives these *distance vectors* from its neighbors
  - DV: a list of [destination, cost]-tuples, (next hop info in table)
- Updates its notion of best path to each destination, and the next hop for this destination
- Features
  - distributed
  - adapts to traffic changes and link failures

Example

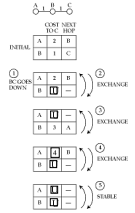


Why does it work?

- Each node knows its true cost to its neighbors
- This information is spread to its neighbors the first time it sends out its distance vector
- Each subsequent dissemination spreads the “truth” one hop
- Eventually, it is incorporated into routing table everywhere in the network
- Proof: **Bellman and Ford**, 1957
- Used in the Routing Information Protocol (RIP)

Problems with distance vector

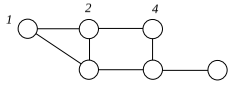
- Works well if nodes are always up
  - problems when links go down or come up
  - DV approach hides details to compute the vector
- Count to infinity



Dealing with the problem

- Path vector
  - DV carries path to reach each destination
  - Trade larger rtg table & extra control overhead **for** robustness
- Split horizon
  - never tell neighbor cost to X if neighbor is next hop to X
  - with poisonous reverse: tell neighbor cost is infinity (faster convergence in some cases)
  - doesn’t work for 3-way count to infinity (assume BA then CA go down in slide 31)
- Triggered updates
  - exchange routes on link failure, instead of on timer
  - faster count up to infinity
- More complicated
  - source tracing (same information as path vector with little additional space)
  - DUAL (Distributed Update ALgorithm)

Source tracing



Destination	Next	Last
1	–	–
2	2	1
3	3	1
4	2	2
5	2	4
6	2	5

DUAL (Distributed Update ALgorithm)

- Avoids loops even in presence of rapid changes
- Router keeps a pared down *topology*
  - sorted union of DVs
- Upon reception of a DV
  - Updates table only if cost decreases (no loop may occur in this case)
  - If cost increases (link’s cost or link failure)
    - check in topology table if shorter path exists
    - if not
      - freeze routing table
      - distribute new DV to all neighbors (recursively) (*expand* until all affected routers know of change)
    - unfreeze and inform “previous” router
    - contract until first router knows that all affected are aware
- Used in EIGRP (Cisco)

Outline

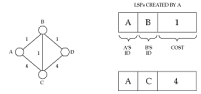
- Distance-vector routing
- Link-state routing
- Choosing link costs
- Hierarchical routing
- Internet routing protocols
- Routing within a broadcast LAN

Link state routing

- In distance vector, router knows only *cost* to each destination
  - hides information, causing problems
- In link state, router knows entire network topology, and computes shortest path by itself
  - independent computation of routes
  - loop free if same view of topology and same algorithm
- Key elements
  - topology dissemination
  - computing shortest routes

Topology dissemination

- A router describes its neighbors with a *link state packet (LSP)*



- Use *controlled flooding* to distribute this everywhere
  - store an LSP in an *LSP database*
  - if *new*, forward to every interface other than incoming one
  - a network with E edges will copy at most 2E times

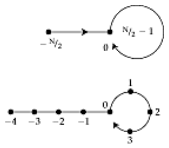
Sequence numbers

- How do we know an LSP is new?
  - Needed to purge “old” information (e.g. after a link failure)
- Use a sequence number in LSP header
- Greater sequence number is newer
- What if sequence number wraps around?
  - smaller sequence number is now newer!
  - Use a large sequence space + comparison on the circle
- But, on boot up, what should be the initial sequence number?
  - have to somehow purge old LSPs
  - two solutions
    - aging
    - lollipop-space sequence numbers

Aging

- Source of LSP puts timeout value in the header
- Router removes LSP when it times out
  - also floods this information to the rest of the network
- So, on booting, router just has to wait for its old LSPs to be purged
- But what age to choose?
  - if too small
    - old LSP could be purged before new LSP fully flooded
    - needs frequent updates
  - if too large
    - router waits idle for a long time on rebooting

A better solution



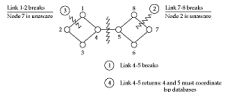
- Need a *unique* start sequence number
- a is older than b if:
  - a < 0 and a < b
  - a > 0, a < b, and b-a < N/4
  - a > 0, b > 0, a > b, and a-b > N/4

More on lollipops

- Additional rule: if a router gets an older LSP, it tells the sender about the newer LSP sequence number
- So, newly booted router quickly finds out its most recent sequence number
- It jumps to one more than that
- -N/2 is a *trigger* to evoke a response from “community memory”

Recovering from a partition

- On partition, LSP databases can get out of synch (inconsistent)



- Databases described by database descriptor records
  - descriptor is link id + version number
- Routers on each side of a newly restored link exchange database descriptors to update databases (determine missing and out-of-date LSPs)

Link or router failure

- *Link* failure easy to detect and recover from
  - Router floods this information
- How to detect *router* failure?
  - HELLO protocol
  - Neighbor floods information about router failure if no response to *N* HELLO packet
- HELLO packet may be corrupted (dead router considered alive!)
  - so age anyway (even with lollipop-space sequence numbers)
  - on a timeout, flood the information

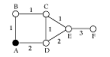
Securing LSP databases

- LSP databases *must* be consistent to avoid routing loops
- Malicious agent may inject spurious LSPs
- Routers must actively protect their databases
  - checksum LSPs even when stored in the database
    - detects corruption on *link* or *disk*
  - ack LSP exchanges
  - authenticate LSP exchanges using passwords

Computing shortest paths

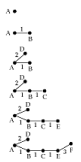
- Based on Dijkstra’s shortest path algorithm
  - computes SP from a “root” to every other node
- Basic idea
  - maintain a set of nodes P to whom we know shortest path
  - initialize P to root
  - consider set {every node one hop away from nodes in P} = T
  - find every way in which to reach a given node in T from root, and choose shortest one
  - then add this node to P

Example



B(A,1) means B was reached by A, cost 1

PERMANENT	TEMPORARY	COMMENTS
A	B(A,1), D(A,2)	ROOT AND ITS NEIGHBORS
A, B(A,1)	D(A,2), C(B,2)	ADD C(B,2)
A, B(A,1) D(A,2)	E(D,4), C(B,2)	C(E,2) DIDN’T MAKE IT
A, B(A,1) D(A,2), C(B,2)	E(C,3)	E(E,8) TOO LONG
A, B(A,1) D(A,2), C(B,2) E(C,3)	F(E,6)	
A, B(A,1) C(B,2), D(A,2) E(C,3), F(E,6)	NULL	STOP



Link state vs. distance vector

- Criteria
  - stability and loop freeness (+LS)
    - in LS routers know entire topology, but transient loops can form (during topology changes flooding)
    - simple modification to vanilla DV algorithm can prevent loops
  - multiple routing metrics (+LS)
    - requires *all* routers agree to report same metrics
  - convergence time after a change (+LS)
    - DV with triggered updates + DUAL has also fast convergence
  - communication overhead (+DV)
    - Nodes are not required to independently compute consistent routes in DV (in LS high overhead to ensure database consistency)
  - memory overhead (+DV)
    - Advantage lost if we use path vector
- Both are evenly matched
- Both widely used (OSPF, BGP)

Outline

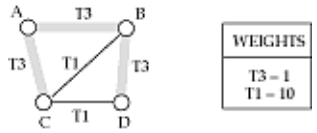
- Distance-vector routing
- Link-state routing
- Choosing link costs
- Hierarchical routing
- Internet routing protocols
- Routing within a broadcast LAN

Choosing link costs

- Shortest path uses link costs
- Can use either static of dynamic costs
- In both cases: cost determine amount of traffic on the link
  - lower the cost, more the expected traffic
  - if dynamic cost depends on load, can have oscillations

Static metrics

- Simplest: set all link costs to 1 => min hop routing
  - but 56K modem link is not the same as a T3!
- Enhancement: give links weight inversely proportional to capacity
- But therefore BC and CD are not used even if T3 are congested



WEIGHTS
T3 = 1 T1 = 10

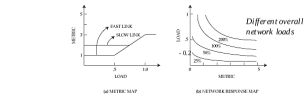
Dynamic metrics

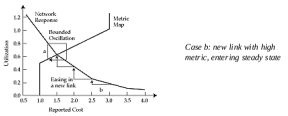
- A first cut (ARPAnet original)
- Cost proportional to length of router queue
  - independent of link capacity
- Unintended consequences of complex design!
  - Many problems when network is loaded
    - queue length averaged over a too small time (10 s) : transient spikes in queue length caused major rerouting
    - cost had wide dynamic range => network completely ignored paths with high costs
    - queue length assumed to predict future loads => opposite is true
    - no restriction on successively reported costs => large oscillations
    - all tables computed simultaneously => low cost links flooded

Modified metrics

- queue length averaged over a small time
- wide dynamic range queue
- queue length assumed to predict future loads
- no restriction on successively reported costs
- all tables computed simultaneously
- queue length averaged over a longer time
- dynamic range restricted (3:1), cost hop normalized
- cost also depends on intrinsic link capacity
  - on low load cost depends only on capacity
- restriction on successively reported costs (1/2 hop)
- attempt to stagger table computation

Routing dynamics





Case a: stable state cost 1.25, U 0.8, m 1.75, U 0.6, m 1.25, ...

Case b: new link with high metric, entering steady state

Are dynamic metrics used?

- Not widely used in today's Internet
- hard to control amount of routing updates a priori
  - dependent on network traffic
- Still can cause oscillations

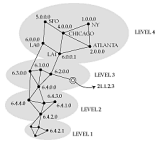
Outline

- Distance-vector routing
- Link-state routing
- Choosing link costs
- Hierarchical routing
- Internet routing protocols
- Routing within a broadcast LAN

Hierarchical routing

- Large networks need large routing tables
  - more computation to find shortest paths
  - more bandwidth wasted on exchanging DVs and LSPs
- Solution:
  - hierarchical routing
- Key idea
  - divide network into a set of domains
  - gateways connect domains
  - computers within domain unaware of outside computers
  - gateways know only about other gateways

Example



- Features
  - only a few routers in each level
  - not a strict hierarchy (both LA_i carry packets to 6.*)
  - gateways participate in multiple routing protocols
  - non-aggregable routes increase core table space (21.1.2.3)

Hierarchy in the Internet

- Three-level hierarchy in addresses
  - network number
  - subnet number
  - host number
- Core advertises routes only to networks, not to subnets
  - e.g. 135.104.*, 192.20.225.*
- Even so, about 80,000 networks in core routers (1996)
- Gateways talk to backbone to find best next-hop to every other network in the Internet

External and summary records

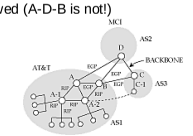
- If a domain has multiple gateways
  - *external* records tell hosts in a domain which one to pick to reach a host in an external domain
    - e.g allows 6.4.0.0 to discover shortest path to 5.* is through 6.0.0.0
  - *summary* records tell backbone which gateway to use to reach an internal node
    - e.g. allows 5.0.0.0 to discover shortest path to 6.4.0.0 is through 6.0.0.0
- External and summary records contain distance from gateway to external or internal node

Interior and exterior protocols

- Internet has three levels of routing
  - highest is at *backbone* level, connecting *autonomous systems (AS)*
  - next level is within AS
  - lowest is within a LAN
- Protocol between AS gateways: exterior gateway protocol
- Protocol within AS: interior gateway protocol

Exterior gateway protocol

- Between untrusted routers
  - mutually suspicious
- Must tell a *border gateway* who can be trusted and what paths are allowed (A-D-B is not!)



- *Transit* over *backdoors* is a problem (A2-C1 should not be summarized)

Interior protocols

- Much easier to implement
  - free of administrative “problems” : no manual configuration
- Typically partition an AS into *areas*
- Exterior and summary records used between areas

Issues in interconnection EGPs and IGPs

- May use different schemes (DV vs. LS)
- Cost metrics may differ
  - 5 hops for an IGP ≠ 5 hops inter-AS
- Need to:
  - convert from one scheme to another
  - use the least common denominator for costs
    - Hop-count metric!
  - manually intervene if necessary

Outline

- Distance-vector routing
- Link-state routing
- Choosing link costs
- Hierarchical routing
- Internet routing protocols
- Routing within a broadcast LAN

Common routing protocols

- Interior
  - RIP
  - OSPF
- Exterior
  - EGP
  - BGP

RIP

- Distance vector
- Cost metric is hop count
- Infinity = 16
- Exchange distance vectors every 30 s
- Split horizon with poisonous reverse
- Useful for small subnets
  - easy to install

OSPF

- Link-state
- Uses areas to route packets hierarchically within AS
- Complex
  - LSP databases to be protected
- Uses *designated routers* to reduce number of endpoints on a broadcast LAN

EGP

- Original exterior gateway protocol
- Distance-vector
- Costs are either 128 (reachable) or 255 (unreachable)
  - only propagates reachability information
  - → backbone must be structured as a tree to ensure loop free
- Allows administrators to pick neighbors to peer with
- Allows backdoors (by setting backdoor cost < 128)
  - not visible to outside systems
- No longer widely used
  - need for loop free topology

BGP

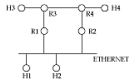
- Path-vector
  - distance vector annotated with entire path
  - also with policy attributes (no cost information)
  - guaranteed loop-free
- Can use non-tree backbone topologies
  - uses true cost (not like EGP)
- Uses TCP to communicate between routers
  - reliable
  - but subject to TCP flow control
- BGP provides the mechanisms to distribute path information
- But leaves (complex) policies to network administrator

Outline

- Distance-vector routing
- Link-state routing
- Choosing link costs
- Hierarchical routing
- Internet routing protocols
- Routing within a broadcast LAN

Routing within a broadcast LAN

- What happens at an endpoint?
- On a point-to-point link, no problem
- On a broadcast LAN
  - is packet meant for destination within the LAN?
  - if so, what is the datalink address ?
  - if not, which router on the LAN to pick?
  - what is the router's datalink address?



Internet solution

- All hosts on the LAN have the same subnet address
- So, easy to determine if destination is on the same LAN
- Local destination's datalink address determined using ARP
  - broadcast a request
  - owner of IP address replies
- To discover routers (default for non local packets)
  - routers periodically sends router advertisements
    - with preference level and time to live (typ. 30 min)
  - pick most preferred router
  - flush when TTL expires
  - can also force routers to reply with *solicitation message* (after a boot)

Redirection

- How to pick the best router?
- Send message to arbitrary router
- If that router's next hop is another router on the same LAN, host gets a *redirect* message
- It uses this for subsequent messages

Contrôle de transmission

Bloc 4, INF 586

Walid Dabbous
INRIA Sophia Antipolis

Plan

- Contrôle d 'erreur
  - de bit (au niveau liaison)
  - de paquet (au niveau transport)
- Contrôle de flux
  - transport
  - en particulier TCP

Le contrôle d 'erreur

Error control

- Error detection & Correction
- Basic idea is to add redundancy to detect or correct errors
- Block code (n,k)
  - add n-k redundancy bits to k data bits to form n bits *codeword*
  - e.g. parity code (k+1, k) *detects* odd number of bit errors
  - rectangular code (parity along rows and column of an array) *corrects* one bit error, with coding delay
- Hamming code
  - valid codewords are « different » enough, so that errored codeword do not ressemble valid codewords
    - *distance*: minimim number of bit inversions to transform VCW1 to VCW2
    - to *detect* E errors : minimal distance is E+1
    - to *correct* E errors : minimal distance is 2E+1
- Interleaved codes
  - transmit column wise a matrix of m consecutive CWs
  - convert burst errors to « bit » errors
  - add memory cost and delay

CRC

- Right n-k bits are the remainder of dividing (n-k)-left shifted "message" by a generator polynomial G(x) of degree (n-k)
- Adequate choice of G(x) allows to detect
  - all single bit errors ($E(x)=x^i$, G has more than two terms)
  - almost all 2-bit errors ($E(x) = x^i+x^j$; G has a factor with at least three terms, chosen not to divide neither x nor $x^{max(i-j)}$)
  - any odd number of errors (E has odd number of terms and G has factor x+1)
  - all bursts up to n-k, where generator bit sequence length is n-k+1 (i.e. n-k check bits)
  - longer bursts with probability $1-2^{-(n-k)}$,if bursts are randomly distributed

Hw/Sw Implementation

- Hardware
  - on-the-fly with a shift register
  - easy to implement with Application Specific Integrated Circuit / Field Programmable Gate Array
- Software
  - Efficiency is important
  - touch each data byte only once
  - rectangular and convolutional not suitable
  - CRC

Software schemes

- TCP/UDP/IP
  - all use same scheme
  - treat data bytes as 16-bit integers
  - add with end-around carry (add 1 to the sum)
  - 16-bit one's complement of the sum = checksum
  - needs only one lookup per 16-bit block
  - catches all 1-bit errors
  - incorrectly validates (uniformly distributed) errors with probability 1/65536

Packet errors

- Different from bit errors
  - causes of packet errors
    - not just erasure, but also duplication, insertion,etc.
  - detection and correction
    - retransmission, instead of redundancy

Causes of packet errors

- Loss
  - due to uncorrectable bit errors (e.g. in wireless environment)
  - buffer loss on overflow
    - especially with bursty traffic
      - for the same load, the greater the burstiness, the more the loss
    - packet losses are bursty (correlation btw consecutive losses)
    - loss rate depends on burstiness, load, and buffer size
  - fragmented packets can lead to error multiplication (TCP>ATM)
    - packet loss rate versus cell loss rate
    - longer the packet, more the loss
      - drop the entire packet at switch

Causes of packet errors (cont.)

- Duplication
  - same packet received twice (2nd is out of sequence)
    - usually due to retransmission
- Reordering
  - packets received in wrong order
    - usually due to retransmission
    - some routing techniques may also reorder
- Insertion
  - packet from some other conversation received
    - (undetectable) header corruption from another active connection (with different TC-identifier)
    - delayed packet from closed connection (with same TC-identifier)

Packet error detection and correction

- Detection
  - Sequence numbers
  - Timeouts
- Correction
  - Retransmission
- Bit level mechanisms active
  - no errors on header

Sequence numbers

- In each header
- Incremented for non-retransmitted packets
- *Sequence space*
  - set of all possible sequence numbers
  - for a 3-bit seq #, space is {0,1,2,3,4,5,6,7}

Using sequence numbers to detect errors

- Reordering & duplication (straightforward)
- Loss
  - gap in sequence space allows *receiver* to detect loss
    - e.g. received 0,1,2,5,6,7 => lost 3,4
  - acks carry *cumulative* seq #
  - redundant information
  - if no ack for a while, *sender* suspects loss
- Insertion
  - if the received seq # is “very different” from what is expected
    - more on this later
- Two important considerations
  - choosing sequence number length
  - choosing initial sequence number

Sequence number bit length - *s*

- Long enough so that sender does not confuse sequence numbers on acks
- E.g, sending at 100 packets/sec (R)
  - sender waits for 200 secs before giving up retransmitting (T)
  - *receiver* may wait up to 100 sec (A) before sending Ack
  - packet can live in the network up to 5 minutes (300 s) (*maximum packet lifetime* or MPL*)*
  - can get an ack as late as 900 seconds after packet sent out
  - sent out 900*100 = 90,000 packets
  - if sequence space smaller, then can have confusion
  - so, s > log (90,000), at least 17 bits
- In general 2^s should be > R(2 MPL + T + A)

MPL

- Lower bound on *s* requires a bound on MPL
- How can we bound it?
- Generation time in header
  - additional space and computation
- Counter in header decremented per hop
  - the Time To Live (TTL)
  - crafty, but works
  - used in the Internet
  - assumes max. diameter, and a limit on forwarding time

Sequence number size (cont.)

- If no retransmissions and acks, size can be smaller: only to detect losses and reordering
- then size depends on two things
  - reordering span: how much packets can be reordered
    - e.g. span of 128 => seq # > 7 bits
  - burst loss span: how many consecutive pkts. can be lost
    - e.g. possibility of 16 consecutive lost packets => seq # > 4 bits
  - both bounds are smaller than the retrx case
  - In practice, do worst case design & hope that technology becomes obsolete before worst case hits!
- Datalink level sequence number shorter than transport
  - usually no retransmission
  - delays are smaller

Packet insertion in CO mode

- Receiver should be able to distinguish packets from other connections
- Why?
  - receive packets on VCI 1
  - connection closes
  - new connection also with VCI 1
  - delayed packet arrives
  - could be accepted
- Solution
  - flush packets on “connection closing”
  - can’t do this for connectionless networks like the Internet
    - need for more sophisticated schemes

Packet insertion (cont.)

- Packets carry source IP, dest IP, *source port number, destination port number*
- How we can have insertion?
  - host A opens connection to B, source port 2345, dest port 6789
  - transport layer connection terminates
  - new connection opens, A and B assign the same port numbers
  - delayed packet from old connection arrives with sequence number in the range used by the newer connection
  - insertion!

Solutions

- Per-connection *incarnation number*
  - incremented for each connection from each host
  - - takes up header space
  - - on a crash, incarnation numbers must be remembered
    - need stable storage, which is expensive
    - not popular in practice
- Reassign port numbers only after 1 MPL
  - remember *time* each port was assigned
  - - needs stable storage to survive crash

Solutions (cont.)

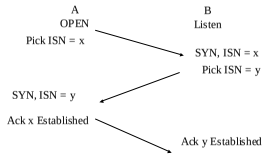
- Assign port numbers serially: new connections have new ports
  - Unix starts at 1024
  - this fails if we wrap around within 1 MPL
  - also fails if computer crashes and we restart with 1024
- chose initial sequence numbers from a clock
  - new connections may have same port, but seq # differs
  - fails on a crash
- Wait 1 MPL after boot up (30s to 2 min)
  - this flushes old packets from network
  - used in most Unix systems

Exchange of Initial Sequence Numbers

- Standard solution, then, is
  - choose port numbers serially (unless specified by user)
  - choose initial sequence numbers from a clock
  - wait 1 MPL after a crash
- Needs communicating ends to tell each other initial sequence number
- Easiest way is to tell this in a *SYNchronize* packet (TCP) that starts a connection
- 2-way handshake
  - does not protect against delayed SYN packets

3-way handshake

- Problem really is that SYNs themselves are not protected with sequence numbers
- 3-way handshake protects against delayed SYNs



Loss detection

- At receiver, from a gap in sequence space
  - send a *nack* to the sender
- At sender, by looking at cumulative acks, and timing out if no ack for a while
  - need to choose timeout interval

Nacks

- Sounds good, but does not work well
  - extra load during loss, even though in reverse direction
- If nack is lost, receiver must retransmit it
  - moves timeout problem to receiver
- So we need timeouts anyway

Timeouts

- Set timer on sending a packet
- If timer goes off, and no ack, resend
- How to choose timeout value?
- Intuition is that we expect a reply in about one round trip time (RTT)

Timeout schemes

- Static scheme
  - know RTT *a priori*
  - timer set to this value
  - works well when RTT changes little (special purpose systems)
- Dynamic scheme
  - measure RTT
  - timeout is a function of measured RTTs
    - larger than RTT to deal with delay variation

Old TCP scheme

- RTTs are measured periodically
- Smoothed RTT (*srtt*)
- *srtt(i) = a * srtt(i-1) + (1-a) * RTT(i)*
- timeout = *b * srtt*
- a = 0.9, b = 2
- sensitive to choice of a
  - a = 1 => timeout = 2 * initial srtt
  - a = 0 => no history
- doesn't work too well in practice

New TCP scheme (Jacobson)

- introduce new error term = *m*
- its smoothed estimate *sm* : mean deviation from mean
- *m(i) = | srtt(i) - RTT(i) |*
- *sm(i) = a * sm(i-1) + (1-a) * m(i)*
- *timeout = srtt + b * sm*
- Different values of *b* give different confidence intervals

Intrinsic problems

- Hard to choose proper timers, even with new TCP scheme
  - What should initial value of srtt be?
    - Particularly hard if a is close to 1 (strong memory)
  - Measuring RTT is hard in presence of losses
    - Ack may acknowledge more than one packet -> hard to determine the packet to derive RTT from
  - Timeout => loss, delayed ack, or lost ack
    - hard to distinguish
- Lesson: use timeouts rarely

Retransmissions

- Sender detects loss on timeout
  - or other "signal"
- Which packets to retransmit?
- Need to first understand concept of error control window

Error control window

- Set of packets sent, but not acked
- 1 2 3 4 5 6 7 8 9 (original window)
- 1 2 3 4 5 6 7 8 9 (recv ack for 3)
- 1 2 3 4 5 6 7 8 9 (send 8)
- May want to restrict max size = window size
- Sender blocked until ack comes back

Go back N retransmission

- On a timeout, retransmit the entire error control window
- Receiver only accepts in-order packets
- + simple
- +conservative: on loss signal, retx every possible lost packet
- + no buffer at receiver
- - can add to congestion
- - wastes bandwidth
- *p* the packet loss probability, *W* the window
  - efficiency = $(1-p)/(1-p+p.W)$
  - low efficiency for high *W* and/or *p*
- used in TCP

Selective retransmission

- Somehow find out which packets lost, then only retransmit them
- How to find lost packets?
  - each ack has a bitmap of received packets
    - e.g. cum_ack = 5, bitmap = 101 => received 5 and 7, but not 6
    - wastes header space
  - sender may therefore periodically ask receiver for bitmap
  - or do fast retransmit (guess that a loss occured)
- requires more complex procedures at both sender and receivers
  - and requires to buffer **W**-1 packets

Fast retransmit

- Assume cumulative acks
- If sender sees repeated cumulative acks, packet likely lost
- 1, 2, 3, 4, 5 , 6
- 1, 2, 3 3 3
- Send cumulative_ack + 1 = 4
- Used in TCP
- Provides partial “selective” information for free
- does not work well in case of multiple error within a window

SMART

- Ack carries cumulative sequence number
- Also sequence number of packet causing ack
- 1 2 3 4 5 6 7 8 9 10
- 1 2 3 4 5 5 5 5
- 1 2 3 4 5 x 7 8 x 10
- (5,5) (5,7) (5,8) (5,10)
- Sender creates bitmap
- Does not use timers
- Not effective if retransmitted packet lost,
  - sender periodically check if cumulative ack increased, and retx N+1
  - on worst case, retrx entire window as in go-back-N

FEC

- Forward Error Correction can also be performed at packet level
- Sends « parity check » packets
- does not require retransmission
- adequate for real time application
  - audio/video conferencing
- But increases load and error rate!
- Not effective if « burst » packet losses
- increases end to end delay
  - wait for entire FEC block before processing

Le contrôle de flux

Flow control problem

- Consider file transfer
- Sender sends a stream of packets representing fragments of a file
- Sender should try to match rate at which receiver and network can process data
- Can't send too slow or too fast
- Too slow
  - wastes time
- Too fast
  - can lead to buffer overflow
- Main objective of flow control
  - how to find the correct rate?

Other considerations

- Simplicity
- Low overhead
  - use of network (bandwidth and buffers) resources
- Scaling to many sources
- Fairness
  - if scarcity of resources, each source gets its "fair" share
- Stability
  - for fixed number of sources, transmission rate for each source settles down to an equilibrium value
- Many interesting tradeoffs
  - low overhead for stability
  - simplicity for fairness

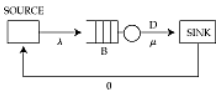
Where?

- Can be at
  - application level
  - transport
  - network
  - link
- At transport layer for end2end flow control
- At datalink layer for hop by hop flow control

- Terminology
  - Flow control vs congestion control
  - congestion is overload of *intermediate* network elements

Model

- Source sending at λ packet/s, sink acks every packet, intermediate servers, (variable) service rate μ packet/s (allocated or available), bottleneck is the slowest server, buffer size at bottleneck B, round trip time (D)



- Flow control: rate-matching with delays
- For flow control purpose: Ignore all but the bottleneck server

Classification

- Open loop
  - Source describes its desired flow rate
  - Network *admits* call and *reserves* resources
  - Source sends at this rate
- Closed loop
  - Source monitors available service rate
    - Explicit or implicit feedback
  - Sends at this rate
  - Due to speed of light delay, errors are bound to occur
- Hybrid
  - Source asks for some minimum rate
  - But can send more, if available

Open loop flow control

- Two phases to flow control, during:
  - Call setup
  - Data transmission
- Call setup
  - Network prescribes traffic descriptor parameters
  - User chooses parameter values
  - Network admits (may negotiate) or denies call
    - if OK, bandwidth and buffers are reserved
- Data transmission
  - User shapes its traffic within parameter range
  - Network *polices* users
  - Scheduling policies give user QoS

Hard problems

- Choosing a descriptor at a source
  - capture future behavior in a set of parameters
- Choosing a scheduling discipline at intermediate network elements (see block 7 - scheduling)
- Admitting calls so that their performance objectives are met (*call admission control*) (not studied in this course, chap 14 in Keshav's book).
- Or just ignore :-)

Traffic descriptors

- Set of parameters that describes behavior of a data source
- It is typically a behavior *envelope*
  - Describes in fact worst case behavior
- Three uses besides describing source behavior
  - Basis for traffic contract
    - if not violated by source, network "guarantees" QoS
  - Input to *regulator*
    - where source delays traffic
  - Input to *policer*
    - where operator delays or drops excess traffic

Descriptor requirements

- Representativity
  - adequately describes flow, so that network does not reserve too little or too much resource
- Verifiability
  - network able easily to verify that descriptor holds
- Usability
  - Easy to describe and use for admission control

Examples

- Representative, verifiable, but not useable
  - Time series of interarrival times
    - potentially very long and unknown for interactive sources
    - network may add jitter
- Verifiable, and useable, but not representative
  - peak rate
    - may send at less than peak rate -> waste resources

Some common descriptors

- Peak rate
- Average rate
- Linear bounded arrival process

- will study each with the corresponding regulator

Peak rate

- Highest 'rate' at which a source can send data
  - trivial bound: the link capacity but does not give a true picture
- Two ways to compute it
- For networks with fixed-size packets
  - (min inter-packet spacing)$^{-1}$
- For networks with variable-size packets, time window *t*
  - bounds total data generated over *all* intervals of duration *t*
- Regulator for fixed-size packets: buffer +
  - timer set on packet transmission to min inter-packet spacing
  - if timer expires, send buffered packet, if any
- Problem
  - sensitive to extremes: a single "drift" may result in a radical change

Average rate

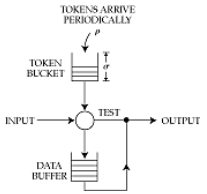
- Measure rate over some time period (*window*) '*t*'
- Less susceptible to outliers
- Parameters: *t* and *a* (number of bits to send during *t*)
- Two types: jumping window and moving window
- Jumping window
  - over consecutive intervals of length *t*, only *a* bits sent
  - sensitive to the choice of the starting time of 1st window
  - regulator *reinitializes* every interval
- Moving window
  - over all intervals of length *t*, only *a* bits sent
  - regulator *forgets* packets sent more than *t* seconds ago
  - removes dependency on starting time

Linear Bounded Arrival Process

- Source bounds # bits sent in any time interval by a linear function of time
- the number of bits transmitted in any active interval of length t is less than or equal to $\rho.t + \sigma$
- ρ is the long term rate *allocated* by network to source
- σ is the burst limit (max burst a source may send)
- a generalization of average rate descriptor
  - also insensitive to outliers

Leaky bucket

- A regulator for an LBAP
- Token bucket fills up at rate ρ
- Largest # tokens = σ



Leaky bucket regulator

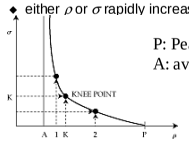
- Leaky bucket can be used as both:
  - a peak rate regulator (ρ = peak rate, σ = 1)
  - or a moving-window average regulator (ρ = average rate)
- Variant
  - Token bucket + peak rate regulator
    - allows to control: average rate, peak rate and max burst
- Has both token and data buckets
  - Sum of sizes is what matters
  - a larger token bucket offsets a smaller data buffer

Choosing LBAP parameters

- How to choose ρ and σ (e.g. for a stored video source)
- Minimal descriptor
  - no other descriptor has both a smaller ρ and a smaller σ
  - presumably costs less
- How to choose minimal descriptor?
  - Not unique
  - tradeoff between ρ and σ
    - for given size of data buffer and max loss rate, for each ρ there is a min σ so that loss rate is met
- Three way tradeoff
  - choice of σ (data bucket size)
  - loss rate
  - choice of ρ

Choosing minimal parameters

- Keeping loss rate the same
  - if σ is more, ρ is less (smoothing)
  - for each ρ in the [A,P] range, we have minimum σ
- For "common" sources choose knee of curve (K)
  - either ρ or σ rapidly increases when moving away from knee



P: Peak rate
A: average rate over a long interval

LBAP

- "Popular" in practice (ATM) and in academia
  - verifiable
  - sort of usable
- BUT do not accurately represent sources with large bursts
  - otherwise σ would be too large
  - makes network expensive
  - what about renegotiating ρ before bursts
    - possible for stored video!
    - Or just after the start of a burst in the case of long bursts
      - buffer still fills while renegotiation

Open loop vs. closed loop

- Open loop
  - describe traffic
  - network admits/reserves resources
  - regulation/policing
- Closed loop
  - can't *describe* traffic or
  - network doesn't support *reservation*
    - resources are overbooked for higher multiplexing gain (SMG)
  - source monitors *available bandwidth*
    - perhaps allocated using GPS-emulation in routers
  - *adapts* to it in order not to overload network
  - if not done properly either
    - excessive packet loss (higher "rate" than bottleneck)
    - underutilize network resources (much slower than bottleneck)

Taxonomy

- First generation (on-off, stop-and-wait, static-window)
  - ignores network state
  - only match receiver
- Second generation
  - responsive to both sink and network states
  - three choices
    - State measurement
      - explicit or implicit
    - Control
      - flow control window size or rate
    - Point of control
      - endpoint or within network

Explicit vs. Implicit

- Explicit
  - Network tells source its current rate
  - Better control
  - More communication and computation overhead
- Implicit (only in end to end schemes)
  - Endpoint figures out rate by looking at network
  - Less overhead
- Ideally, want overhead of implicit with effectiveness of explicit

Flow control window

- Recall error control window
  - Largest number of packet outstanding (sent but not acked)
- If endpoint has sent all packets in window, it must wait => slows down its rate
- Thus, window provides *both* error control and flow control
- Flow control window is also called *transmission* window
- indirectly control a source' rate by modifying transmission window
- but this coupling of error and flow control can be a problem
  - Few buffers at receiver => small window (if selective repeat) => slow rate!

Adaptive window or adaptive rate

- In adaptive rate, we directly control rate
- Needs a fine grain timer per connection
  - set after a packet trx to inverse of trx rate
- Plusses for window
  - easier to implement: no need for fine-grained timer
  - self-limiting
- Plusses for rate
  - better control (finer grain)
  - no coupling of flow control and error control
- Rate control must be carefully engineered to avoid overhead and sending too much (in case of loss of rate limiting packet)

Hop-by-hop vs. end-to-end

- Hop-by-hop
  - make first generation flow responsive to network state
    - control at each link, next server = sink
  - easy to implement
- End-to-end
  - sender matches all the servers on its path
- Plusses for hop-by-hop
  - simpler mechanisms
  - better control
  - distributes buffer usage
- Plusses for end-to-end
  - cheaper, does not require complexity in routers

Closed loop flow control schemes

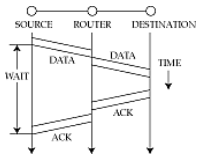
	Explicit		Implicit	
	Dynamic window	Dynamic rate	Dynamic window	Dynamic rate
End2end	DECbit	ATM EERC	TCP	NetBLT, pp
Hop-by-hop	Credit-based	Mishra/Kanakia	_	_

On-off flow control

- Receiver gives ON and OFF signals
- If ON, send at full speed
- If OFF, stop
- OK when RTT is small
- What if OFF is lost?
- Generates bursty traffic
  - packet losses in intermediate elements
- Used in serial lines or LANs
  - delays are small
  - packet loss rare
  - basis of the XON/XOFF protocol used to control serial I/O devices (printers, mice)

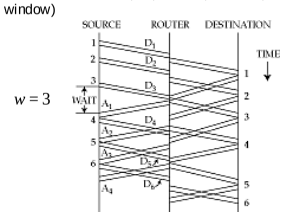
Stop and Wait

- Send a *single* packet
- Wait for ack before sending next packet
- provides error and flow control
- inefficient if delay is large
- Max throughput
  - 1 packet per RTT



Static window

- Stop and wait can send at most one pkt per RTT
- Here, we allow multiple packets per RTT (*w* = transmission window)



What should window size be?

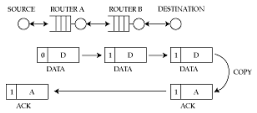
- Let bottleneck service rate along path = μ pkts/sec
- Let round trip time = R sec
- Let flow control window = *w* packet
- Sending rate is *w* packets in R seconds = w/R packets/s
- To keep bottleneck fully utilized
  - w/R > μ => *w* > Rμ
- This is the *bandwidth delay product* or *optimal window size*

Static window

- Works well if μ and R are fixed
- Even for a specific bottleneck, the rate changes with time!
- Static choice of *w* can lead to problems
  - too small
    - bottleneck underutilized
  - too large
    - *w* - Rμ packets buffered at bottleneck
- So, need to adapt window
- Always try to get to the *current* optimal value

DECbit flow control (dynamic window)

- Intuition
  - every packet has a bit in header
  - intermediate routers set bit if queue has built up => source window is too large
  - sink copies bit to ack
  - if bits set, source reduces window size
  - in steady state, oscillate around optimal size



DECbit evaluation

- Only 1 bit is required
- does not require per-connection *queuing* at routers
- can adapt and oscillates around stable optimal window value
  - (Additive Increase Multiplicative Decrease policy)
- Requires per-connection router *actions*
- Increase policy is conservative
  - increase by 1 every two RTTs
  - bad performance on eLePHaNts (Long and Fat pipe Networks)

TCP Flow Control

- Implicit
- Dynamic window
- End-to-end
- Very similar to DECbit, but
  - no support from routers
  - increase if no loss (usually detected using timeout)
  - window decrease on a timeout (or 3 duplicate acks)
  - additive increase multiplicative decrease

TCP details

- Window starts at 1
- Increases exponentially for a while, then linearly
- Exponentially => doubles every RTT
- Linearly => increases by 1 every RTT
- During exponential phase, every ack results in window increase by 1
- During linear phase, window increases by 1 when # acks = window size
- Exponential phase is called *slow start*
- Linear phase is called *congestion avoidance*

More TCP details

- On a loss, current window size is stored in a variable called *slow start threshold* or *ssthresh*
- Switch from exponential to linear (slow start to congestion avoidance) when window size reaches threshold
- Loss detected either with timeout or duplicate cumulative acks (*fast retransmit*)
- Two (early) versions of TCP
  - Tahoe: in both cases, drop window to 1
  - Reno: on timeout, drop window to 1, and on fast retransmit drop window to half previous size (also, do *fast recovery*: increase window by 1 for each duplicate ack, until new data acked)

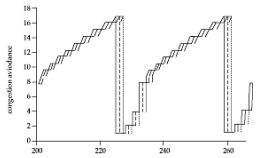
TCP vs. DECbit

- Both use dynamic window flow control and (stable) additive-increase multiplicative decrease policy
- TCP uses implicit measurement of congestion
  - probe a "black box"
- TCP source does not filter information
  - each packet loss indicates congestion
  - necessary because network operated at the *cliff* (close to overload)

Evaluation

- Effective over a wide range of bandwidths
- A lot of operational experience
- Weaknesses
  - loss => overload? (wireless)
    - link level retrx or FEC to make wireless link appear loss-free
    - link level "informs" TCP of link losses
  - loss => self-blame, problem with malicious users on FCFS
  - overload detected only on a loss
    - in steady state, source *induces* loss
  - sensitive to choice of ssthresh for short transfers
    - if large can lead to *multiple* packet losses, FastRTx will not help
  - needs per connection buffering at bottleneck

Sample trace



TCP Vegas

- Source computes Expected throughput = transmission_window_size/propagation_delay
- Numerator: known
- Denominator: measure *smallest* RTT
- Also know *actual* throughput
- Difference = how much to reduce/increase rate
- Algorithm
  - send a special packet
  - on ack, compute expected and actual throughput
  - if expected < actual, adjust propagation_delay
  - (expected - actual)* RTT packets are still in bottleneck buffer
  - adjust sending rate if this is out of L&H watermarks
- "performs better" than TCP Reno
  - but rate based and not TCP reno-fair

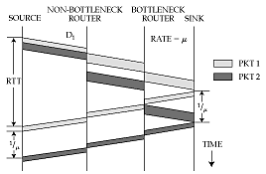
NETBLT

- "First" rate-based flow control scheme
- Separates error control (window) and flow control (no *coupling*)
- So, losses and retransmissions do not affect the flow rate
- Application data sent as a series of buffers, each at a particular rate
- Rate expressed as a burst size and a burst rate
  - so granularity of rate control = burst
- In the original scheme, no rate adjustment
- Later, if received rate < sending rate, multiplicatively decrease rate, otherwise linearly increase
- Change rate only once per buffer => slow

Packet pair

- Improves basic ideas in NETBLT
  - better measurement of bottleneck
  - control based on prediction
  - finer granularity
- Assume all bottlenecks serve packets in *round robin* order
- Then, spacing between 2 packets of same connection at receiver (= ack spacing) = 1/(rate of slowest server)
- If *all* data sent as paired packets, no distinction between data and probes
- Implicitly determine service rates if routers are round-robin-like

Packet pair

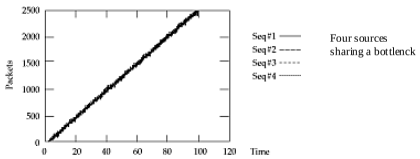


Packet-pair details

- Acks give time series of service rates in the past
- We can use this to predict the next rate

- but requires round robin in routers!

Sample trace



Comparison among closed-loop schemes

- On-off, stop-and-wait, static window, DECbit, TCP, NETBLT, Packet-pair, ATM Forum EERC (End2End Rate based flow Control)
- Which is best? No simple answer
- Some rules of thumb
  - flow control easier with Round Robin scheduling
    - otherwise, assume cooperation, or police allocated rates
  - explicit schemes are more robust
  - hop-by-hop schemes are more responsive, but more complex
  - try to separate error control and flow control
  - rate based schemes are inherently unstable unless well-engineered

Architecture de protocoles haute performance

Bloc 5, INF 586

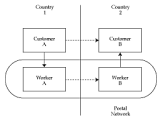
Walid Dabbous
INRIA Sophia Antipolis

Outline

- Protocol Layering
- Layering considered harmful
- Enhancing Protocol performance
- Adaptive applications
- Application Level Framing

Protocol Layering

Peer entities



- Customer A and B are *peers*
- Postal worker A and B are *peers*

Protocols

- A *protocol* is a set of rules and formats that govern the communication between communicating peers
  - set of valid messages
  - meaning of each message
- A protocol is necessary for any function that requires cooperation between peers

Example

- Exchange a file over a network that corrupts packets
  - but doesn't lose or reorder them
- A simple protocol
  - send file as a series of packets
  - send a *checksum*
  - receiver sends OK or not-OK message
  - sender waits for OK message
  - if no response, resends entire file
- Problems
  - single bit corruption requires retransmission of entire file
  - what if link goes down?
  - what if not-OK message itself is corrupted?

What does a protocol tell us?

- *Syntax* of a message
  - what fields does it contain?
  - in what format?
- *Semantics* of a message
  - what does a message mean?
  - for example, not-OK message means receiver got a corrupted file
- *Actions* to take on receipt of a message
  - for example, on receiving not-OK message, retransmit the entire file
- The three above called: protocol specification

Another way to view a protocol

- As providing a *service*
- The example protocol provides *reliable file transfer service*
- Peer entities use a protocol to provide a service to a higher-level peer entity
  - for example, postal workers use a protocol to present customers with the abstraction of an *unreliable letter transfer* service

Protocol layering

- A network that provides many services needs many protocols
- Turns out that some services are independent
- But others depend on each other
- Protocol A may use protocol B as a *step* in its execution
  - for example, packet transfer is one step in the execution of the example reliable file transfer protocol
- This form of dependency is called *layering*
  - reliable file transfer is *layered* above packet transfer protocol
  - like a subroutine

Some terminology

- *Service access point (SAP)*
  - interface between an upper layer and a lower layer
- *Protocol data units (PDUs)*
  - packets exchanged between peer entities
- *Service data units (SDUs)*
  - packets handed to a layer by an upper layer
- PDU = SDU + optional header or trailer
- Example
  - letter transfer service
  - protocol data unit between customers = letter
  - service data unit for postal service = letter
  - protocol data unit = mailbag (aggregation of letters)

Protocol stack

- A set of protocol layers
- Each layer uses the layer below and provides a service to the layer above
- Key idea
  - once we define a service provided by a layer, we need know nothing more about the details of *how* the layer actually implements the service
  - information hiding
  - decouples changes
  - but reduces system performance!

The importance of being layered

- Breaks up a complex problem into smaller manageable pieces
  - can compose simple service to provide complex ones
  - for example, WWW (HTTP) is Java layered over TCP over IP (and uses DNS, ARP, DHCP, RIP, OSPF, BGP, PPP, ICMP)
- Abstraction of implementation details
  - separation of implementation and specification
  - can change implementation as long as service interface is maintained (long distance telephone migration from copper to fiber)
- Can reuse functionality
  - upper layers can share lower layer functionality
  - example: WinSock on Microsoft Windows

Problems with layering

- Layering hides information
  - if it didn't then changes to one layer could require changes everywhere
    - *layering violation*
- But sometimes hidden information can be used to improve performance
  - for example, flow control protocol may think packet loss is always because of network congestion
  - if it is, instead, due to a lossy link, the flow control breaks
  - this is because we hid information about reason of packet loss from flow control protocol

Layering

- There is a tension between information-hiding (abstraction) and achieving good performance
- Art of protocol design is to leak enough information to allow good performance
  - but not so much that small changes in one layer need changes to other layers

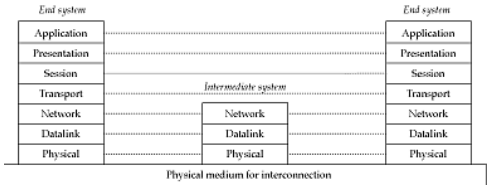
ISO OSI reference model

- A set of protocols is *open* if
  - protocol details are publicly available
  - changes are managed by an organization whose membership and transactions are open to the public
- A system that implements open protocols is called an *open system*
- Any vendor can implement open standard compliant systems
- International Organization for Standards (ISO) prescribes a standard to connect open systems
  - *open system interconnect (OSI)*
- Has greatly influenced thinking on protocol stacks

ISO OSI

- *Reference model*
  - formally defines what is meant by a layer, a service etc.
- *Service architecture*
  - describes the services provided by each layer and the service access point
- *Protocol architecture*
  - set of protocols that implement the service architecture
  - compliant service architectures may still use non-compliant protocol architectures

The seven layers



Physical layer

- Moves bits between physically connected end-systems
- Standard prescribes
  - coding scheme to represent a bit
  - shapes and sizes of connectors
  - bit-level synchronization
- Postal network
  - technology for moving letters from one point to another (trains, planes, vans, bicycles, ships…)
- Internet
  - technology to move bits on a wire, wireless link, satellite channel etc.

Datalink layer

- Introduces the notion of a *frame*
  - set of bits that belong together
- *Idle* markers tell us that a link is not carrying a frame
- *Begin* and *end* markers delimit a frame
- On a broadcast link (such as Ethernet)
  - end-system must receive only bits meant for it
  - need datalink-layer address
  - also need to decide who gets to speak next
  - these functions are provided by *Medium Access sublayer (MAC)*
- Some data links also retransmit corrupted packets and pace the rate at which frames are placed on a link
  - part of *logical link control sublayer*
  - layered over MAC sublayer

Datalink layer (contd.)

- Datalink layer protocols are the first layer of *software*
- Very dependent on underlying physical link properties
- Usually bundle both physical and datalink layer on *host adaptor card*
  - example: Ethernet
- Postal service
  - mail bag 'frames' letters
- Internet
  - a variety of datalink layer protocols
  - most common is Ethernet
  - others are FDDI, SONET, HDLC

Network layer

- Logically concatenates a set of links to form the abstraction of an end-to-end link
- Allows an end-system to communicate with any other end-system by computing a route between them
- Hides specificities of datalink layer
- Provides unique network-wide addresses
- Found both in end-systems and in intermediate systems
- At end-systems primarily hides details of datalink layer
  - segmentation and reassembly
  - some error detection (e.g header check)

Network layer (contd.)

- At intermediate systems
  - participates in routing protocol to build routing tables
  - responsible for forwarding packets
  - scheduling the transmission order of packets (Not implemented)
  - choosing which packets to drop (Not deployed)
- IP only provides "best effort"
  - it runs an "all" underlying technologies

Transport layer

- Network provides a 'raw' end-to-end service
- Transport layer provides the abstraction of an *error-controlled*, *flow-controlled* and *multiplexed* end-to-end link
- Error control
  - message will reach destination despite packet loss, corruption and duplication
  - retransmit lost packets; detect, discard, and retransmit corrupted packets; detect and discard duplicated packets
- Flow control
  - match transmission rate to rate currently sustainable on the path to destination, and at the destination itself

Transport layer (contd.)

- Multiplexes multiple applications to the same end-to-end connection
  - adds an application-specific identifier (*port number)* so that receiving end-system can hand in incoming packet to the correct application
- Some transport layers provide fewer services
  - e.g. simple error detection, no flow control, and no retransmission
  - *lightweight transport layer*

Transport layer (contd.)

- Postal system
  - doesn't have a transport layer
  - transport level functionality is implemented, if at all, by customers
  - detect lost letters (how?) and retransmit them
- Internet
  - two popular protocols are TCP and UDP
  - TCP provides error control, flow control, multiplexing
  - UDP provides only multiplexing

Session layer

- Not common
- Provides *full-duplex service, expedited data delivery,* and *session synchronization*
- Duplex
  - if transport layer is simplex, concatenates two transport endpoints together
- Expedited data delivery
  - allows some messages to skip ahead in end-system queues, by using a separate low-delay transport layer endpoint
- Synchronization
  - allows users to place marks in data stream and to roll back to a pre-specified mark

Presentation layer

- Unlike other layers which deal with *headers* presentation layer touches the application data
- Hides data representation differences between applications
  - e.g. *endian-ness*
- Can also encrypt data
- Usually *ad hoc*
- Internet
  - no standard presentation layer
  - only defines network byte order for 2- and 4-byte integers

Application layer

- The set of applications that use the network
  - File transfer
  - E-mail,
  - Web access,
  - Audio and video conferencing
  - Distributed games
  - Shared virtual environments
- Doesn't provide services to any other layer

Layering

- We have broken a complex problem into smaller, simpler pieces
- Provides the application with *sophisticated* services
- Each layer provides a clean abstraction to the layer above

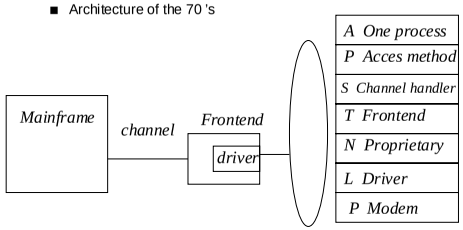
Layering considered harmful

Why seven layers?

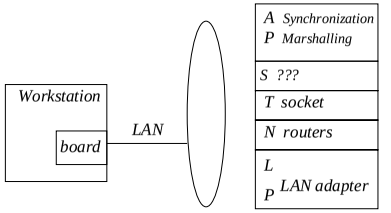
- Need a top and a bottom -- 2
- Need to hide physical link, so need datalink -- 3
- Need both end-to-end and hop-by-hop actions; so need at least the network and transport layers -- 5
- Session and presentation layers are not so important, and are often ignored
- So, we need at least 5, and 7 seems to be excessive
- Note that we can place functions in different layers
- Will study the impact on performance

Layering considered harmful

- Architecture of the 70 's



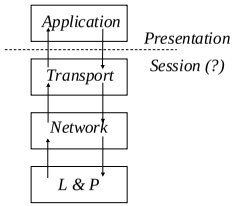
Architecture of the 90 's



Layering considered harmful

- A layer is considered as an asynchronous entity
- Independent « vendors » to develop layers hw/sw
- How to implement an asynchronous entity on a workstation
  - A subroutine?
    - No asynchrony
  - A process
    - several kind of overhead
      - process scheduling
      - asynchronous interfacing
      - passing control and data

What about persentation and session?



How to pass application control?

- « Transport » should be aware of the « Application »
  - e.g. VMTP replies serving as ACKs
- This avoids « bad » decisions that may be taken by the protocol
  - ACK a packet when a reply is waiting
  - After a packet loss during a video transfer
    - close the window
    - retransmit the packet

Layering, the lessons

- Asynchronous interfaces result in reduced efficiency
  - reduce asynchrony
- Avoid « artificial separation » in layers
- Have such interfaces only when necessary i.e. between
  - transmission networks
  - end system hardware and operating system
  - end system communications stack
- Specific case for OSI failure:
  - slow standardization process (political done by the “goers”)
  - done before technology was mature

Enhancing protocol performance

“Enhancing Protocol Performance”

- Impact of the environment
- Parameter tuning & adaptive algorithms
- Special purpose protocols
- Can applications “share” performance?

Impact of the environment

- Implementing protocols in software inside the OS.
- The OS executes the protocol code
- For each packet
  - take an interrupt or two,
  - reset a timer or two
  - allocate a buffer
  - schedule a process

Operating system support

- An O.S. Wish List:
  - Shared memory among processes
  - Blocking on multiple events
  - Good I/O buffer management
  - Good resource management scheduler
  - Low overhead timers
  - High resolution clocks
- Over the last years, it has gotten much better

Generic protocol enhancements

- Protocol parameter tuning
  - Acknowledgments
    - ACK grouping
    - Nacks
  - Adaptive timer values
- New adaptation algorithms
  - TCP slow-start

Special purpose protocols

- Target a specific application
  - File transfer (NETBLT)
  - (distributed) Intra-system communication (VMTP)
- Multiple communication modules overhead
- Provide a toolkit
  - XTP
  - merges layers 3 and 4
  - not a good choice

Can we share performance?

- Two views of performance :
- The explicit approach -- classical telecommunications design.
  - Voice channels shall have a digital bandwidth of 64Kb/s
  - Direct match to application
- The implicit approach -- classical computer design.
  - Applications don't seem to have real requirements
  - Live in a « virtual » world of performance.

The « virtual » world of computers

- Most computer systems attempt to hide the real performance limits of the hardware.
- Real memory limits -> virtual memory.
  - The bigger the program, the slower it runs.
- One CPU -> multiple processes.
  - The more processes, the slower each runs.
- One disk -> multiple files.
  - Can run out of disk space, but not file space
- In the virtual world:
  - Performance gets subdivided
  - Logical entities are not bounded a priori

Connectivity

- In the network world, connectivity is the logical entity
- Degree of connectivity is critical parameter
  - Not like telephone, but many conversations at once
  - Even a small workstation (a client) may have many simultaneous conversations
  - Patterns of connectivity are highly variable
- The computer is a programmable device
  - Network designers should live with this!
- Connectivity has nothing to do with bandwidth
  - many computer applications have very minimal demand for bandwidth. Especially high-connectivity applications

The « virtual » network

- The most natural model of a network (to a computer programmer)
- Bandwidth is a performance parameter, and just gets subdivided.
- Connectivity is a a logical construct, and should be unbounded
- A bit of a shock to the voice specialists
- Voice nets are exactly backwards from this
- Video nets as well.

Two examples

- Ethernet: a success
  - Very high connectivity (no set-up)
  - Fixed total bandwidth, arbitrary allocation (No allocation).
- ISDN circuits: Not a success
  - One logical connection per physical
  - Fixed bandwidth per connection
- Ethernet, with very poor bandwidth allocations tools, has supported effectively a wide range of applications

Can networks be virtual?

- In the world of virtual performance:
  - Add performance in needed quantity by system configuration
- Do networks work this way?
- In the past: we have not built network technology with « scalable » performance
  - Initial solution: gross over-design
  - Problem: it does not last.
- Current products are addressing this need
  - Switching hubs

Can protocols be virtual?

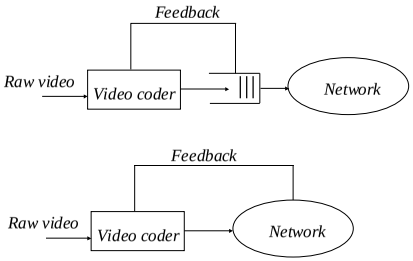
- Protocols like TCP are from the virtual school of performance. They assume that performance will be added *later*, in the proper quantities
- But when is « later »?
  - Protocol implementation time may be « too soon ». Performance gets frozen before the application is defined
  - the telephone system was designed for ONE application. Computer systems are general; we don't know the application
- Can we build software that scales?

Adaptive applications

Adaptive applications

- Can network applications *adapt?*
- Should network applications adapt?

Video over the Internet



A/Video applications can adapt to bandwidth

- Video codec is controllable
  - Control the frame rate
  - Or the frame quality
    - quantization granularity
    - movement detection threshold
- What about audio?
  - Use of multiple codec
  - covers a « wide » range of adaptability

A/Video application can adapt to packet losses

- Add redundant information in packets
- Each packet may contain a sample of the previous packet
  - on a combination of the *k* previous packets
- Increases bandwidth!
- Redundancy should be added within a total fixed budget
  - how to best allocate the channel capacity

Applications « should » adapt

- Underlying networks are heterogeneous
- Bandwidth is not free
- No resource reservation and QOS routing supported
- Graceful degradation
- High fidelity
- Efficiency (optimal use of the available resource)
- Lessons
  - share performance
  - use a connection less packet switching network interface
  - a communication subsystem integrated within the application

Application Level Framing

A new communication architecture

- We need to reduce asynchrony
- more involve the application in transmission
- Analyze protocol functions
  - data manipulation functions
    - (copy, checksum, buffer allocation, data alignment, marshalling, byte swap, compression, encryption)
  - control functions
    - (sequence numbers, ACKs, window, etc…)
- Example of the transport:
  - Control: demultiplex, seq. numbers, update W, 10s of instructions
  - Data manipulation: copy and checksum
    - better performance if these two operations were *combined*

What is the bottleneck

- Heavy manipulation functions are the bottleneck
  - Presentation encoding/decoding, encryption
- How to best use the slowest part in the chain
  - remove transport level resequencing
  - let the application decide
  - need for autonomous « Application Data Units »

Application Level Framing

- An ADU is :
- Unit of error and flow control = Unit of transmission
  - avoid transmission inefficiencies in case of error
- Unit of processing = Unit of transmission
  - avoid idle waits
- Unit of processing = Unit of error and flow control
  - simplify adaptive multimedia applications design
- Can be processed as soon as it is received
  - augment the « message » structure
  - adequate size (ADU size discovery)

Conclusion

- ALF results in more complex application design
  - No code re-use
  - specific protocol mechanisms integrated within the application
- But, hopefully scalable
  - if carefully designed
- Research oriented audio and video conference applications are based on this architecture

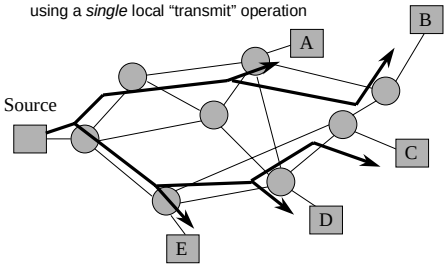
Group Communication

Bloc 6, INF 586

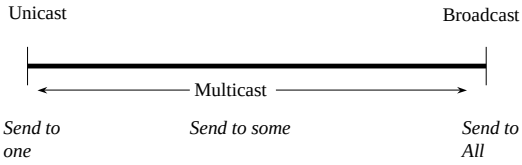
Walid Dabbous
INRIA Sophia Antipolis

Definition

- **Multicast**: is the act of sending a message to multiple receivers using a *single* local "transmit" operation



A Spectrum of Paradigms

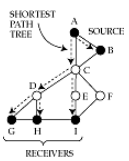


Send to one — *Send to some* — *Send to All*

Multicast flavors

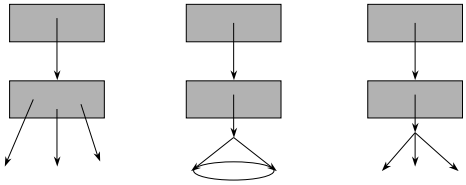
- Unicast: point to point
- Multicast:
  - point to multipoint
  - multipoint to multipoint
- Can simulate point to multipoint by a set of point to point unicasts
- Can simulate multipoint to multipoint by a set of point to multipoint multicasts
- The difference is efficiency

Multicast efficiency



- Suppose A wants to talk to B, G, H, I, B to A, G, H, I
- With unicast, 4 messages sent from each source
  - links AC, BC carry a packet in triplicate
- With point to multipoint multicast, 1 message sent from each source
  - but requires establishment of two separate multicast "groups"
- With multipoint to multipoint multicast, 1 message sent from each source,
  - single multicast "group"

The Layering of Multicast

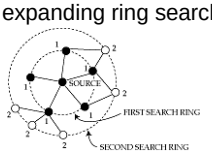


Multicast by Unicast — Multicast by Broadcast — Multicast by Multicast

The Many Uses of Multicasting

- Teleconferencing (1-to-many) and symmetric (all to all)
- Distributed simulation (war gaming, multi-player Doom)
- Resource discovery (where's the next time server?)
- Software/File Distribution
- Video Distribution
- Network news (Usenet)
- Replicated Database Updates

Example use: expanding ring search



- A way to use multicast groups for resource discovery
- Routers decrement TTL when forwarding
- Sender sets TTL and multicasts
  - reaches all receivers <= TTL hops away
- Discovers local resources first
- Since heavily loaded servers can keep quiet, automatically distributes load

Outline

- Wide area Multicast routing
  - or the dream of « universal » communication
- Reliable multicast transport
- Multicast congestion control
- *Not covered*: enforcing reception semantics across receivers (ordering, atomicity) -- better see in "distributed computing" litterature

Multicast Routing

Group Communication

- What: communicate with a group of endpoints, rather than just one
  - a "natural" generalization of unicast
- basis for real-world organizations
  - groups of cooperating entities
- fantasy: group communication as the basis for "organizations of computers"
  - the "true" foundation for building distributed systems
  - it just needs to scale
- Is it easy to implement?

Distributed Systems Structure

Three primary layers

Application Processing
Distributed State/Storage Management
Communication Facilities

What functionality to put at each level?

Hard? multicast versus unicast

- To implement a unicast application, I can use TCP, RPC mechanisms, RMI (Remote Method Invocation), etc.
- With multicast, just basic sockets and UDP
- Where is the multicast TCP?
- Where is the multicast middleware?
- Why so limited Internet multicast deployment?

Let's look at some history ...

In the 1970's, ...

- Multi-access networks appeared
  - Ethernet, rings
- Broadcast: an accident of the technology
- We discovered uses for this "accident":
  - discovery: e.g. broadcast to locate a printer server
  - multi-point delivery: e.g. Mazewar games
- But not scalable: a single broadcast address
  - e.g. 3 Mbps experimental Ethernet broadcast address
  - Not every node involved in Mazewars, a print server

Even LAN group communication needs to be scalable

In the 1980's, ...

- 10 Mbps Ethernet:
  - 47 bits of multicast addresses
  - Lots of addresses for group communication applications

L2 group communication has become feasible

V Distributed System: early 80s

- Extended RPC-like IPC to support group operations
  - send message to "group" object; 0, 1 or more return messages
- Example uses:
    - name server group for distributed lookup
    - scheduler group to select host for remote execution
    - process manager group to act on one or more processes
    - transaction groups for distributed transactions
    - various multi-player games, distributed applications
- Experience: Average performance, fault-tolerance and ease of programming

The group model was born!

History of IP Multicast

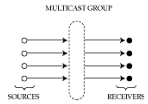
- 1983: how to scale multicast from an Ethernet to the Internet?
- IP multicast memo - april 1984
- Deering's thesis - 1991
  - host group model and IGMP
  - DVMRP - local broadcast-and-prune routing
- Focus on host group model and "local" routing
- Required underlying service
  - datagram + multicast delivery in LAN
  - (if broadcast or unicast LAN -> emulate multicast)

The host group model

Deering, 1991

- senders need not be members
- groups may be of any size
- no topological restrictions on membership
- membership dynamic and autonomous
- host groups may be transient or permanent

Multicast group



- Associates a set of senders and receivers with each other
  - but independent of them
  - created either when a sender starts sending from a group
  - or a receiver expresses interest in receiving
  - even if no one else is there!
- Sender does not need to know receivers' identities
  - *rendezvous point*

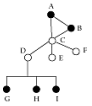
Addressing

- Multicast group in the Internet has its own Class D address
  - looks like a host address, but isn't
- Senders send to the address
- Receivers anywhere in the world request packets from that address
- "Magic" is in associating the two: *dynamic directory service*
- Four problems
  - which groups are currently active - *sdr*
  - how to express interest in joining a group - *IGMP*
  - discovering the set of receivers in a group - *Flood and prune*
  - delivering data to members of a group - *Reverse path forwarding*

Issues in wide-area multicast

Difficult because

- sources may join and leave dynamically
  - need to dynamically update shortest-path tree
- leaves of tree are often members of broadcast LAN
  - would like to exploit LAN broadcast capability

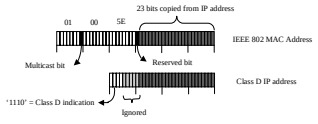


- would like a receiver to join or leave without explicitly notifying sender
  - otherwise it will not scale

Multicast in a broadcast LAN

- Wide area multicast can exploit a LAN's broadcast capability
- E.g. Ethernet will multicast all packets with multicast bit set on destination address
- Two problems:
  - what multicast MAC address corresponds to a given Class D IP address?
  - does the LAN have contain any members for a given group (why do we need to know this?)

Class D to MAC translation



- Multiple Class D addresses map to the same MAC address
  - a host may receive MAC-layer mcast for groups to which it does not belong -> dropped by IP
- Well-known translation algorithm => no need for a translation table

Internet Group Management Protocol

- Detects if a LAN has any members for a particular group
  - If no members, then we can *prune* the shortest path tree for that group by telling parent
- Router periodically broadcasts a *query* message
- Hosts reply with the list of groups they are interested in
- To suppress traffic
  - reply after random timeout
  - broadcast reply
  - if someone else has expressed interest in a group, drop out
- To receive multicast packets:
  - translate from class D to MAC and configure adapter

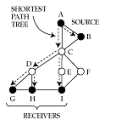
Wide area multicast

- Assume
  - each endpoint is a router
  - a router can use IGMP to discover all the members in its LAN that want to subscribe to each multicast group
- Goal
  - distribute packets coming from any sender directed to a given group to all routers on the path to a group member

Simplest solution

- Flood packets from a source to entire network
- If a router has not seen a packet before, forward it to all interfaces except the incoming one
- Pros
  - simple
  - always works!
- Cons
  - routers receive duplicate packets
  - detecting that a packet is a duplicate requires storage, which can be expensive for long multicast sessions

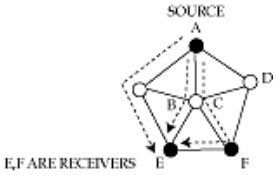
Shortest path tree



- Ideally, want to send exactly one multicast packet per link to reach all interested destinations
  - forms a *multicast tree* rooted at sender
- Optimal multicast tree provides *shortest* path from sender to every receiver
  - *shortest-path* tree rooted at sender

A clever solution

- *Reverse path forwarding*
- Rule
  - forward packet from S to all interfaces if and only if packet arrives on the interface that corresponds to the shortest path *to* S
  - no need to remember past packets
  - C need not forward packet received from D

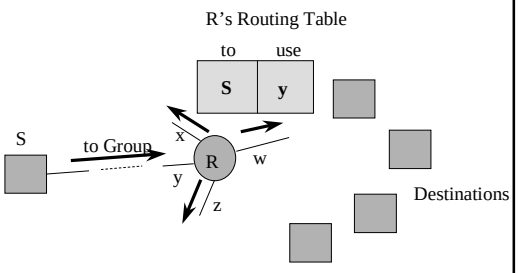


Reverse Path Forwarding

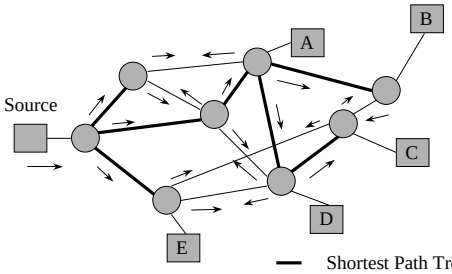
Detailed description

- Routers forward based on *source* of multicast packet
- Flood on all outgoing interfaces if packet arrives from Source on link that router would use to send packets to source
- Otherwise Discard
- Rule avoids flooding loops
- Uses Shortest Path Tree from destinations to source (reverse tree)

Reverse Path Forwarding: router action

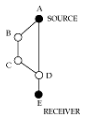


Reverse Path Forwarding



Cleverer

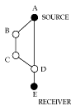
- Don't send a packet downstream if you are not on the shortest path from the downstream router to the source
- C need not forward packet from A to E



- Potential confusion if downstream router has a choice of shortest paths to source (nodes B and C in figure on slide 28)

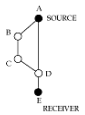
Pruning

- RPF does not completely eliminate unnecessary transmissions



- B and C get packets even though they do not need it
- Pruning => router tells parent in tree to stop forwarding
- Can be associated either with a multicast group or with a source *and* group
  - trades selectivity for router memory

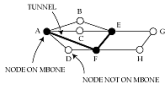
Rejoining



- What if host on C's LAN wants to receive messages from A after a previous prune by C?
  - IGMP lets C know of host's interest
  - C can send a *join(group, A)* message to B, which propagates it to A
  - or, periodically flood a message; C refrains from pruning

A problem

- Reverse path forwarding requires a router to know shortest path to a source
  - known from routing table
- seems straightforward!
- But not all routers do support multicast
  - *virtual links* between multicast-capable routers
  - shortest path to A from E is not C, but F

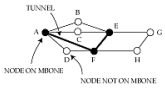


A problem (contd.)

- Two problems
  - how to build virtual links
  - how to construct routing table for a network with virtual links

Tunnels

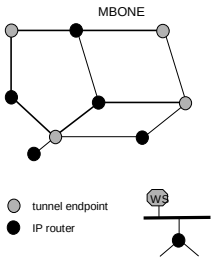
- Why do we need them?



- Consider packet sent from A to F via multicast-incapable D
- If packet's destination is Class D, D drops it
- If destination is F's address, F doesn't know multicast address!
- So, put packet destination as F, but carry multicast address internally
- Encapsulate IP in IP => set protocol type to IP-in-IP

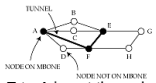
Mbone

- Mbone = multicast backbone
- virtual network overlaying Internet
- needed until mcast capable routers deployed
- IP in IP encapsulation
- limited capacity, resilience



Multicast routing protocol

- Interface on "shortest path" to source depends on whether path is real or virtual



- Shortest path from E to A is not through C, but F
  - so packets from F will be flooded, but not from C
- Need to discover shortest paths only taking multicast-capable routers into account
  - DVMRP

DVMRP

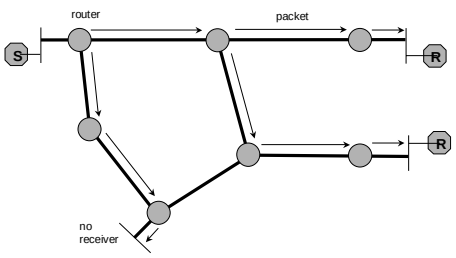
- Distance-vector Multicast routing protocol
- Very similar to RIP
  - distance vector
  - hop count metric
- Used in conjunction with
  - flood-and-prune (to determine memberships)
    - prunes store per-source and per-group information
  - reverse-path forwarding (to decide where to forward a packet)
  - explicit join/graft messages to reduce join latency (but no source info, so still need flooding)

Multicast Forwarding in DVMRP

1. check incoming interface: discard if not on shortest path to source
2. forward to all outgoing interfaces
3. don't forward if interface has been *pruned*
4. prunes time out every two minutes to remove state information in routers

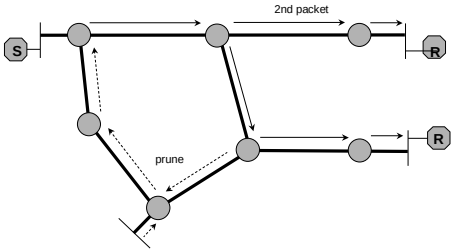
DVMRP Forwarding (cont.)

Basic idea is to flood and prune



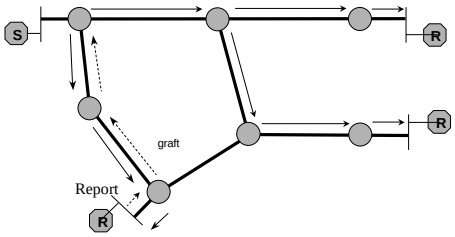
DVMRP Forwarding (cont.)

Prune branches where no members and branches not on shortest paths



DVMRP Forwarding (cont.)

Add new user via grafting; departure via pruning



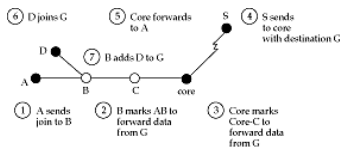
MOSPF

- Multicast extension to OSPF
- Routers flood group membership information with LSPs
- Each router independently computes shortest-path tree that only includes multicast-capable routers
  - no need to flood and prune
- Complex
  - interactions with external and summary records
  - need storage per group per link
  - need to compute shortest path tree per source and group

Core-based trees

- Problems with DVMRP-oriented approach
  - need to periodically flood and prune to determine group members
  - need to store per-source and per-group prune records at each router
- Key idea with core-based tree
  - coordinate multicast with a *core* router
  - host sends a join request to core router
  - routers along path mark incoming interface for forwarding

Example



- Pros
  - routers not part of a group are not involved in pruning
  - explicit join/leave makes membership changes faster
  - router needs to store only one record per group
- Cons
  - all multicast traffic traverses core, which is a bottleneck
  - traffic travels on non-optimal paths

Protocol independent multicast (PIM)

- Tries to bring together best aspects of CBT and DVMRP
- Choose different strategies depending on whether multicast tree is *dense* or *sparse*
  - flood and prune good for dense groups
    - only need a few prunes
    - CBT needs explicit join per source/group
  - CBT good for sparse groups
- Dense mode PIM == DVMRP
- Sparse mode PIM is similar to CBT
  - but receivers can switch from CBT to a shortest-path tree

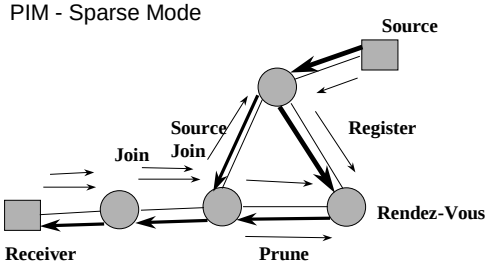
PIM- Dense Mode

- Independent from underlying unicast routing
- Slight efficiency cost
- Contains protocol mechanisms to:
  - detect leaf routers
  - avoid packet duplicates

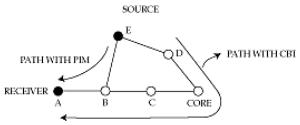
PIM - Sparse Mode

- Rendezvous Point (Core): Receivers Meet Sources
- Reception through RP connection = Shared Tree
- Establish Path to Source = Source-Based Tree

PIM - Sparse Mode



PIM (contd.)



- In CBT, E must send to core
- In PIM, B discovers shorter path to E (by looking at unicast routing table)
  - sends join message directly to E
  - sends prune message towards core
- Core no longer bottleneck
- Survives failure of core

More on core

- Renamed a *rendezvous point*
  - because it no longer carries all the traffic like a CBT core
- Rendezvous points periodically send “I am alive” messages downstream
- Leaf routers set timer on receipt
- If timer goes off, send a join request to alternative rendezvous point
- Problems
  - how to decide whether to use dense or sparse mode?
  - how to determine “best” rendezvous point?

Inter-domain multicast

- Need complex protocols for
  - Intra-domain address allocation
  - Inter-domain address allocation
  - Intra-domain multicast routing (DVMRP/MOSPF/PIM)
  - Build inter-domain multicast tree

The Multi-source Multicast Problem

- Far harder than we thought!
- The rendezvous problem: How does sender in Afghanistan find receivers in Argentina?
  - A highly dynamic global directory at the IP-level?
    - If you can solve this ...
- How to deny this sender if unwanted?
  - still uses network resources unless widely denied
- A global address space of 28-bits, with dynamic allocation by applications
  - not enough bits to allocate

Mission impossible, and for what applications?

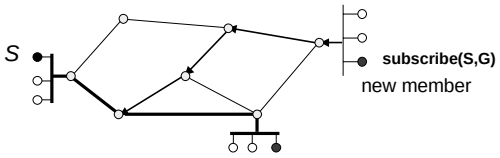
Scalable Multicast Applications?

- Small-scale multicast can just use unicast
  - not pretty but it works, except for discovery
- Large-scale discovery, expanding ring search?
  - 100 million hosts, each occasionally multicasts to the 100 million - how occasional? How about never!
- Most large-scale applications are single-source
  - e.g. Multicast file transfer, Internet TV/radio, web cache update/invalidation

Multi-source multicast is hard and not needed!

Solution: Single-source multicast

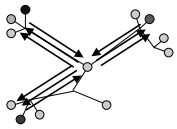
- Key (embarrassingly simple) idea:
  - identify multicast distribution by (S,G)



- Subscription to (S,G) follows unicast path to S
  - relies only on unicast routing information

Scalable multicast routing becomes trivial

Video Conferencing



- 1 channel per participant, if small
  - same cost as for PIM-SM if all are active
- Large video conference - too many channels
- But, floor control and access control is needed
  - "rendevous through an application-level moderator
- Same (or less) cost as PIM-SM at the network layer

SSM is fine with video conferences

Multi-source Group Applications?

- Small groups: unicast or channel per member
- Large groups without structure are mobs
  - Application/middleware-level relays or reflectors needed to provide structure
- Same as a PIM rendezvous point but:
  - can do floor control, access control
  - application can select RP
  - application can select redundancy, fail-over strategy
  - dramatically simplifies the network layer
- Network "middle" boxes can provide relay service

SSM + relays is superior for multi-source apps

Benefits

- Solves the scalable multicast routing problem:
  - join protocol, building on proven unicast routing
- Solves the access control problem:
  - only source can send (duh!)
- Solves multicast address allocation problem
  - thousands of multicast addresses per-source
- Simplifies the network layer

15 years to realize we were putting too much at the network level

The Internet deployment delayed as a result

Some Lessons

- We need to ask hard questions about real compelling applications
  - The service model should be a slave to these applications
- We had the wrong service model:
  - SSM channel model versus the group model
- End-to-end (again): do not put functionality at the lower level unless there is a real win,
  - because it can be a real lose!
- Group communication is hard

Dimensionality of Group Comm.

- Size - how many
- Reliability - of message delivery
- Timing - synchronized with receivers
- Proximity - near by, far away
- Similarity - homo. vs. hetero. nodes
- Network connectivity - fast/slow, errors, etc.

Group comm. Is far more diverse than unicast; no, far, far far more diverse

Size of groups

- With unicast, its one - the other end
- With group communication, it could be:
  - 10
  - 100
  - 1000
  - 10,000
  - 100,000
  - 1,000,000

So, how many solutions to group problems are there? Many, many?

Reliability

- With unicast, the other end needs to receive the packet
- With group communication, it could be we need:
  - one of the group to receives it
  - a few of the group to receive it
  - Most ...
  - all?

Again, not just one single group problem,

but many!

Timing

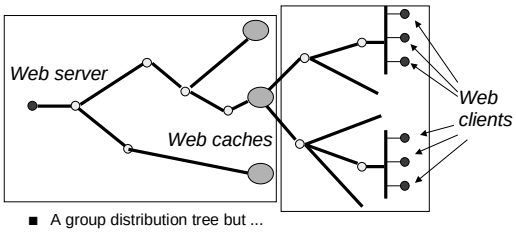
- With unicast, the other end is ready to receive
- With group communication, it could be:
  - one of the group is ready for the message
  - a few of the groups are ready
  - several ...
  - Most
  - all?

Not just one group problem, but many?

Group Comm. is not Scalable!

- Any middleware has to deal with the cross-product of (some of) these dimensions
  - seems hopeless
- Within one application:
  - the larger the group, the more the time skew
  - handling time skew implies longer-term storage
  - long-term storage (disk) not part of the real-time store-and-forward communication layer.
- Alternative: file-and-forward networking

Example: Web Multi-point Delivery



- A group distribution tree but ...
- Web cache as a “file-and-forward” node
- Limited scale groups from caches to clients
- Limited scale groups from server to caches

A combination of comm. and storage nodes

Summary

- Multicast Routing is well researched problem.
- However, challenge now is
  - deployment
  - inter-operability
  - management

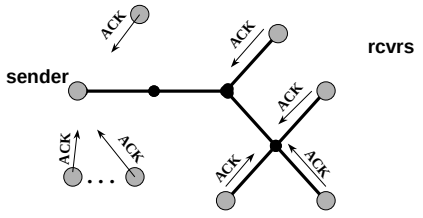
Reliable Multicast

Problem

How to transfer data reliably from source to *R* receivers

- scalability: 10s -- 100s -- 1000s -- 10000s -- 100000s of receivers
- heterogeneity
- feedback implosion problem

Feedback Implosion Problem



Issues

- level of reliability
  - full reliability
  - semi-reliability
- ordering
  - no ordering
  - ordering per sender
  - full ordering (distributed computing)

Applications

- application requirements
  - file transfer, finite duration
  - streaming applications (billing, etc.), infinite duration
  - low latency (DIS, teleconferencing)
- application characteristics
  - one-many: one sender, all other participants receivers (streaming appl. teleconferencing)
  - many-many: all participants send and receive (DIS)

Approaches

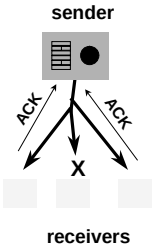
- shift responsibilities to receivers
- feedback suppression
- server-based recovery
- local recovery
- forward error correction (FEC)

Sender Oriented Reliable Mcast

Sender: mcasts all (re)transmissions
selective repeat
use of timeouts for loss detection
ACK table

Rcvr: ACKs received pkts

Note: group membership important



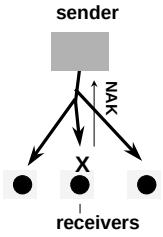
Vanilla Rcvr Oriented Reliable Mcast

Sender: mcasts (re)transmissions
selective repeat
responds to NAKs

Rcvr: upon detecting pkt loss
sends pt-pt NAK
timers to detect lost retransmission

Note: easy to allow joins/leaves

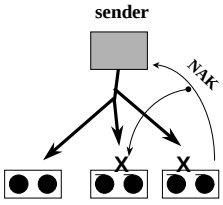
Significant performance improvement shifting burden to receivers for 1-many; not as great for many-many



Feedback Suppression

- randomly delay NAKs
- multicast to all receivers

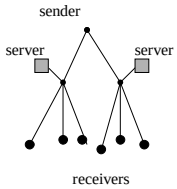
+ reduce bandwidth
- additional complexity at receivers (timers, etc)



Server-based Reliable Multicast

- first transmisions mcast to all receivers and servers
- each receiver assigned to server
- servers perform loss recovery
- can have more than 2 levels

LBRM (Cheriton)



Local Recovery

Lost packets recovered from nearby receivers

- deterministic methods
  - impose tree structure on rcvrs with sender as root
  - rcvr goes to upstream node on tree

  RMTP (Lucent)
- self-organizing methods
  - rcvrs elect nearby rcvr to act as retransmitter using scoped multicast and random delays (SRM)
- hybrid methods

RMTP (Lucent)

Reliable Multicast Transport Protocol

- imposes a tree structure on rcvrs corresponding to multicast routing tree
- nodes inside tree
  - aggregate ACKs/NAKs
  - provide repairs to downstream nodes
- late-joins supported thru 2-level cache
- rate- and window-based flow control

SRM (LBL)

Scalable Reliable Multicast

- rcvr-oriented using NAK suppression and self-organizing local recovery
- supports late-joins and leaves
- as built in wb, uses rate-based flow control
- has been used with 100s of participants over the Internet

SRM detailed operation

- NACKs and retransmissions are multicast
  - suppress duplicates, random delay before replying
- Each host estimates the « delay » with all other hosts
  - schedule an action after a randomization delay
  - chose a smaller delay for NACKs/retransmissions if closer to source/requesting host
- If other request/repair received
  - cancel action, double interval
- If timer expired
  - do action, double interval

Forward Error Correction (FEC)

Add redundancy in order to reduce need to recover from losses (e.g., Reed Solomon codes)

(*k,n*) code

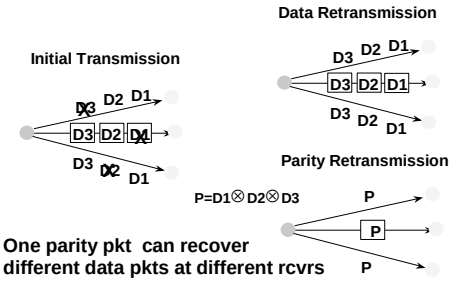
- for every *k* data pkts, construct *n-k* parity pkts
- can recover all data pkts if no more than *n-k* losses

\+ reduce loss probability

\- greater overheads at end-hosts

Q: can FEC reduce network resource utilization?

Potential Benefits of FEC



One parity pkt can recover different data pkts at different rcvrs

Summary

- reliable mcast is a hot research topic
- unresolved issues
  - proper integration of different ideas wrt different applications
  - integration with flow/congestion control
  - interaction with group membership
  - notion of semi-reliability

Multicast congestion control

Problem

- Match transmission rates to
  - Network capacity
  - Receiver “consumption” rates

Multicast Flow Control Challenges

- Accommodating *heterogeneity* among receivers and paths leading to them
- Preserving *fairness* among
  - receivers of same flow
  - distinct flows
- *Scalability* of feedback

Multicast Flow Control Solutions

- Loss-Tolerant Applications (e.g., Video)
  - Information content per unit time can be preserved at lower data rates
- Applications demanding data integrity
  - lower data rates => lower information content per unit time
- *Goal*: Co-Existence with TCP?

Single rate congestion control

- Scalable Feedback Control
  - Receivers measure loss rates
  - Randomly generated feedback
  - Source estimates receivers’ state and adjusts video rate by changing compression parameters
- Source adapts to “slowest” receiver
  - or another “single” rate
- Problem: fairness among receivers

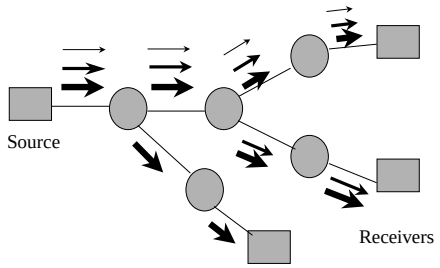
Simulcast

- Improving *fairness* using
- Send replicated video streams at different rates
  - Receivers can control rate of each stream within limits
  - Receivers can move among streams
- Fairness at the expense of increased bandwidth consumption

Receiver-driver Layered Multicast

- Single video stream subdivided into layers
- Receivers add and drop layers depending on congestion
- Challenge: Distributed Consensus, Layer Synchronization

Receiver-driven Layered Multicast



Receiver-Driven Layered Multicast

- Drop Layer:
  - indicated by loss
- Add Layer:
  - No such indication
- Use join experiments with shared learning
  - Reluctance to join layers that failed
  - Inform others via multicast of failed experiments

Flow Control for Reliable Multicast

- Less Understood/Mature Area
- Some Possibilities:
  - Window flow control (a la TCP)
    - Not Scalable, Not Fair (across receivers)
  - Multiple Multicast Groups

Multiple Multicast Groups

- Simulcasting or Destination Set Splitting
- Cumulative Layering

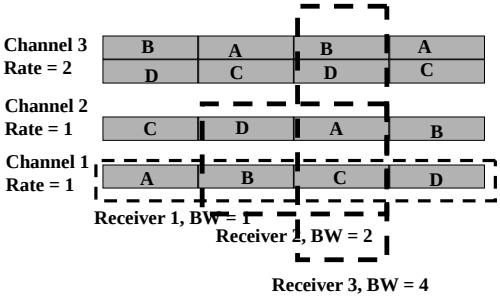
Simulcasting

- Similar to the Video case
- Send multiple (*uncoordinated* streams) at different rates
- Each stream carries all data
- Receivers join appropriate stream - *one at a time*

Cumulative Layering

- Multiple data streams at different rates
- Each stream contains entire data
- Receivers join asynchronously -- Streams transmit continuously
- Schedule to minimize reception time
- (Scheme allows for FEC encoding)

Cumulative layering



Cumulative Layering

- Can achieve minimum reception time with asynchronous receivers
- Schedulability requires some parameter relationships
- Synchronization among channels needed

Summary

- Flow control for multicast communication is a **hard** problem:
  - Scalability
  - Heterogeneity
  - Added dynamic dimension (receivers and their join behavior)
  - Plenty of room for innovation!

Scheduling

Bloc 7, INF 586

Walid Dabbous
INRIA Sophia Antipolis

Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

Scheduling

- Sharing resources always results in contention
  - file systems
  - long distance trunks
  - web sites
- A *scheduling discipline* resolves contention:
  - who's next?
- Key to *fairly sharing resources* and *providing performance guarantees*

Components of a scheduling discipline

- A scheduling discipline does two things:
  - decides service order
  - manages queue of service requests
- Example:
  - consider queries awaiting web server
  - scheduling discipline decides service order
  - and also if some query should be ignored
    - storage is limited
- Allocates different service qualities to different users by its
  - choice of service order (allocate different delays)
  - choice of which request to drop (allocate different loss rate)

Where to schedule?

- Anywhere where contention may occur
- When statistical fluctuations result in queuing
  - not in circuit switched networks
- At every layer of protocol stack
  - e.g. the web server application
- We will focus on network layer:
  - bandwidth on a specific link
  - *output* queue buffers at routers
    - assumed sufficiently fast switch fabrics

Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

Why do we need one?

- *Because future applications need it*
- We expect two types of future applications
  - "elastic" or best-effort (adaptive, non-real time)
    - e.g. email, some types of file transfer
  - guaranteed service (non-adaptive, real time)
    - e.g. packet voice, interactive video

What can scheduling disciplines do?

- Give different users different qualities of service
- Scheduling disciplines can allocate
  - bandwidth
  - delay
  - loss
- Required to provide "performance guarantees"
- They also determine how *fair* the network is
  - even if best effort applications do not require performance bounds

The Conservation Law

- FCFS is the simplest possible scheduling discipline but
  - provides no differentiation among connections
- More sophisticated scheduling discipline provides this
  - but sum of mean delays (weighted by load share) is independent from the scheduling discipline

N connections at a scheduler, λ_i mean rate for connection i, x_i mean service time for a packet from connection i. ρ_i mean utilization of a link due to connection i, q_i mean waiting time for a packet from connection i at (work-conserving) scheduler:

$$\sum_{i=1}^{N} \rho_i q_i = Cst$$

Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

Requirements

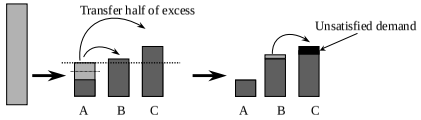
- An ideal scheduling discipline
  - is easy to implement
  - is fair (for best effort connections)
  - provides performance bounds (for GS connections)
  - allows easy *admission control* decisions (for GS)
    - to decide whether a new flow can be allowed

Requirements: 1. Ease of implementation

- Scheduling discipline has to make a decision once every few microseconds!
- Should be implementable in a few instructions in hardware
  - for hardware: critical constraint is VLSI *space* required to maintain scheduling state and *time* to access this state
    - single shared buffer is easy
    - per-connection queuing not feasible
- Work per packet should scale less than linearly with number of active connections
  - O(N) does not scale (N=100.000 simultaneous connections in wide-area routers)

Requirements: 2. Fairness

- Scheduling discipline *allocates* a *resource* (bw, buffers)
- An allocation is fair if it satisfies *max-min fairness*
  - maximizes the minimum share of a source whose demand is not fully satisfied
  - resources allocated in order of increasing demand
  - No source gets more than its demand
  - Sources with unsatisfied demand get an *equal* share of the resource
- Intuitively
  - each connection gets no more than what it wants
  - the excess, if any, is equally shared



Fairness (contd.)

- Fairness is *intuitively* a good idea for best effort connection
  - GS connections should pay the network
  - for network operators fairness is not a concern
- Fairness is a *global* objective, but scheduling is local
- Each endpoint must restrict its flow to the *smallest* fair allocation
- Dynamics + delay => global fairness may never be achieved
- But it also provides *protection*
  - traffic hogs cannot overrun others
  - automatically builds *"firewalls"* around heavy users
- NB: policing at network entrance provides protection, but not fairness

Requirements: 3. Performance bounds

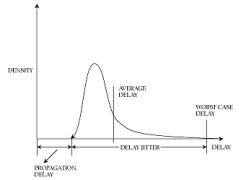
- What is it?
  - A way to obtain a desired level of service
  - restricted by conservation law
    - cannot give *all* connections delay lower than FCFS
- Contract between user and network
  - user somehow communicates perf req to operator
  - hard to guarantee end to end performance bounds
- Performance bounds can be *deterministic* (holds for every packet) or *statistical* (probabilistic bound)
- Common parameters are
  - bandwidth
  - delay
  - delay-jitter
  - loss (will consider zero loss)

Bandwidth

- Specified as minimum bandwidth measured over a prespecified interval
- E.g. > 5Mbps over intervals of > 1 sec
- Meaningless without an interval!
- Can be a bound on average (sustained) rate or peak rate
- Peak is measured over a 'small' interval
- Average is asymptote as intervals increase without bound
- Bw bound required for all GS connections

Delay and delay-jitter

- Bound on some parameter of the delay distribution curve



- GS networks are expected to specify and guarantee only the deterministic or statistical *worst-case* delay (every other connection behaves in the worst possible manner)

Req'ments: 4. Ease of admission control

- Admission control needed to provide QoS
- Decide given the currents connections whether to accept a new one without jeopardizing the performance of existing connections
- Overloaded resource cannot guarantee performance
  - but performance guarantees should not lead to network underutilization
- Choice of scheduling discipline affects ease of admission control algorithm

Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

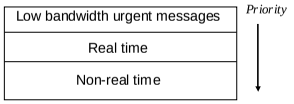
Fundamental choices

- Degrees of freedom in designing a scheduling discipline

1. Number of priority levels
2. Work-conserving vs. non-work-conserving
3. Degree of aggregation within a level
4. Service order within a level

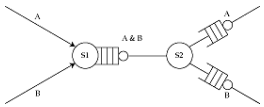
Choices: 1. Priority scheduling

- Packet is served from a given priority level only if no packets exist at higher levels (*multilevel priority with exhaustive service*)
- Highest level gets lowest delay
- Watch out for starvation! (admission control for all but lowest priority whose 'server' is on 'vacation' when server higher priority)
- Usually map priority levels to delay classes



Choices: 2. Work conserving vs. non-work-conserving

- Work conserving discipline is never idle when packets await service
- Why bother with non-work conserving?
- Avoid burst 'accumulation' that
  - requires larger buffers
  - results in higher jitter



Non-work-conserving disciplines

- Key conceptual idea: delay packet till *eligible*
- Reduces delay-jitter => fewer buffers in network
- How to choose eligibility time?
  - rate-jitter regulator
    - bounds maximum outgoing rate
    - E(1) = A(1); E(k+1) = max(E(k)+Xmin, A(k+1))
    - where Xmin is inverse of peak rate
  - delay-jitter regulator
    - compensates for variable delay at previous hop
    - E(0,k) = A(0,k); E(i+1,k) = E(i,k) + D + L
    - D is max delay at previous switch, L max delay on transmission link between switch i and i+1

Do we need non-work-conservation?

- Can remove delay-jitter at an endpoint instead
  - but also reduces size of switch buffers…
- Increases mean delay
  - not a problem for *playback* applications
- Wastes bandwidth
  - can serve best-effort packets instead
- Always punishes a misbehaving source
  - even if bandwidth is available
- Bottom line: not too bad, implementation cost may be the biggest problem (calendar queue)

Choices: 3. Degree of aggregation

- More aggregation
  - less state
  - cheaper
    - smaller VLSI
    - less to advertise
  - BUT: less individualization
- Solution
  - aggregate to a *class*, members of class have same performance requirement
  - no protection within class
    - 'share' burst effect

Choices: 4. Service within a priority level and an aggregation class

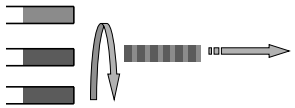
- In order of arrival (FCFS) or in order of a service tag
- Service tags => can arbitrarily reorder queue
  - Need to sort queue, which can be expensive
- FCFS
  - bandwidth hogs win (no protection)
    - greediness is rewarded
  - no guarantee on delays
- Service tags
  - with appropriate choice, both protection and delay bounds possible

Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

Scheduling best-effort connections

- Main requirement is (max-min) *fairness*
- Achievable using *Generalized processor sharing (GPS)*
  - Visit each non-empty (virtual) queue in turn
  - Serve infinitesimal from each
    - in any finite time interval it can visit every logical queue at least one
  - achieves max-min fairness by definition
  - may serve data in proportion to given *weight*



More on GPS

- GPS is unimplementable!
  - we cannot serve infinitesimals, only packets
- No packet discipline can be as fair as GPS
  - while a packet is being served, we are unfair to others
- Degree of unfairness can be bounded
- **Define**: *work(i,a,b)* = # bits transmitted for connection i in time [a,b]
- *Absolute* fairness bound (AFB) for discipline S
  - Max (work_GPS(i,a,b) - work_S(i, a,b))
- *Relative* fairness bound (RFB) for discipline S
  - Max (work_S(i,a,b) - work_S(j,a,b))

What next?

- We can't implement GPS
- So, lets see how to emulate it
- We want to be as fair as possible
- But also have an efficient implementation

Weighted round robin

- RR: Serve a packet from each non-empty queue in turn
- Unfair if packets are of different length or weights are not equal
- Different weights, fixed packet size
  - serve more than one packet per visit, after normalizing to obtain integer weights
- Different weights, variable size packets
  - normalize weights by mean packet size
    - e.g. weights {0.5, 0.75, 1.0}, mean packet sizes {50, 500, 1500}
    - normalize weights: {0.5/50, 0.75/500, 1.0/1500} = { 0.01, 0.0015, 0.000666}, normalize again {60, 9, 4}

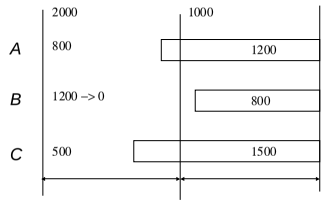
Problems with Weighted Round Robin

- With variable size packets and different weights, need to know mean packet size in advance
  - what about compressed video?
- Fair on time scales longer than a round time
  - Can be unfair for long periods of time
    - if a connection has a small weight
    - or number of connections is large
- E.g.
  - T3 trunk with 500 connections, each connection has mean packet length 500 bytes, 250 with weight 1, 250 with weight 10
  - Each packet takes 500 * 8/45 Mbps = 88.8 microseconds
  - Round time =2750 * 88.8 = 244.2 ms

Deficit round-robin

- Modifies WRR to handle variable packet sizes
  - without knowing mean packet size of each connection in advance
- Initialize 'deficit counter' to zero
- visit each queue
  - if (DC + quantum) ≥ size of packet at head of queue -> serve
    - and decrement deficit counter
- Easy to implement
- But fair on large time scale

Example



Weighted Fair Queueing (WFQ)

- Deals better with variable size packets and weights
  - like DRR
- GPS is fairest discipline
- Find the *finish time* of a packet, *had we been doing GPS*
- Then serve packets in order of their finish times

WFQ

- Suppose, in each *round,* the server served one bit from each active connection
- *Round number* is the number of rounds already completed
  - can be fractional
- If a packet of length *p* arrives to an empty queue when the round number is *R*, it will complete service when the round number is *R + p => finish number* is *R + p*
  - independent of the number of other connections!
- If a packet arrives to a non-empty queue, and the previous packet has a finish number of *f*, then the packet's finish number is *f+p*
- Serve packets in order of finish numbers

Evaluation

- Pros
  - like GPS, it provides protection
  - can obtain worst-case end-to-end delay bound
  - gives users incentive to use intelligent congestion control (and also provides rate information implicitly)
- Cons
  - needs per-connection state
  - implementation complexity
  - explicit sorting of output queue

Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

Scheduling guaranteed-service connections

- With best-effort connections, goal is fairness
- With guaranteed-service connections
  - what performance guarantees are achievable?
  - how easy is admission control?
- We now study some scheduling disciplines that provide performance guarantees

WFQ

- Turns out that WFQ also provides performance guarantees
- Bandwidth bound
  - ratio of weights * link capacity
  - e.g. connections with weights 1, 2, 7; link capacity 10
  - connections get at least 1, 2, 7 units of b/w each
- End-to-end delay bound
  - assumes that the connection doesn't send 'too much' (otherwise its packets will be stuck in queues)
  - more precisely, connection should be *leaky-bucket* regulated
  - # bits sent in time $[t_1, t_2] <= \rho (t_2 - t_1) + \sigma$

Parekh-Gallager theorem

- Let a connection be allocated weights at each WFQ scheduler along its path, so that the least bandwidth it is allocated is *g*
- Let it be leaky-bucket regulated such that # bits sent in time $[t_1, t_2] <= \rho (t_2 - t_1) + \sigma$
- Let the connection pass through *K* schedulers, where the *k*th scheduler has a link rate *r(k)*
- Let the largest packet allowed in the network be *Pmax*
- The transmission delay is bounded by:

$$D^*(i) \le \sigma(i)/g(i) + \sum_{k=1}^{K-1} P\max(i)/g(i,k) + \sum_{k=1}^{K} P\max/r(k)$$

Significance

- Theorem shows that WFQ can provide worst-case end-to-end delay bounds
- So WFQ provides both fairness and performance guarantees
- Bound holds regardless of cross traffic behavior
- Can be generalized for networks where schedulers are variants of WFQ, and the link service rate changes over time
- But bounds are VERY large and useless

Problems

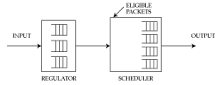
- To get a delay bound, need to pick *g*
  - the lower the delay bounds, the larger *g* needs to be
  - large *g* => exclusion of more competitors from link
  - *g* can be very large, in some cases 80 times the peak rate!
- Sources must be leaky-bucket regulated
  - but choosing leaky-bucket parameters is problematic
- WFQ couples delay and bandwidth allocations
  - low delay requires allocating more bandwidth
  - wastes bandwidth for low-bandwidth low-delay sources

Delay-Earliest Due Date

- Earliest-due-date: packet with earliest deadline selected
- Delay-EDD prescribes how to assign deadlines to packets
- A source is required to send slower than its *peak rate*
- Bandwidth at scheduler reserved at peak rate
- admission control ensures that delay bound will be met
- Deadline = 'expected' arrival time + delay bound (time at which it should be sent, had it been received according to the contract)
  - If a source sends faster than contract, delay bound will not apply
- Each packet gets a hard delay bound
- E2E Delay bound is *independent* of bandwidth requirement
  - but reservation is at a connection's peak rate
- Implementation requires per-connection state and a priority queue

Rate-controlled scheduling

- A *class* of disciplines
  - two components: regulator and scheduler
  - incoming packets are placed in regulator where they wait to become *eligible*
  - then they are put in the scheduler
- Regulator *shapes* the traffic, scheduler provides performance guarantees

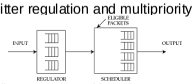


Analysis

- First regulator on path monitors and regulates traffic => bandwidth bound
- End-to-end delay bound
  - delay-jitter regulator
    - reconstructs traffic => end-to-end delay is fixed (= worst-case delay at each hop)
  - rate-jitter regulator
    - partially reconstructs traffic
    - can show that end-to-end delay bound is smaller than (sum of delay bound at each hop + delay at first hop)

Decoupling

- Can give a low-bandwidth connection a low delay without overbooking
- E.g consider connection A with rate 64 Kbps sent to a router with rate-jitter regulation and multipriority FCFS scheduling



- After sending a packet of length *P*, next packet is eligible at time (now + *P*/64 Kbps)
- If placed at highest-priority queue, all packets from A get low delay
- Can decouple delay and bandwidth bounds, unlike WFQ

Evaluation

- Pros
  - flexibility: ability to emulate other disciplines
  - can decouple bandwidth and delay assignments
  - end-to-end delay bounds are easily computed
  - do not require complicated schedulers to guarantee protection
  - can provide delay-jitter bounds
- Cons
  - require an additional regulator at each output port
  - delay-jitter bounds at the expense of increasing mean delay
  - delay-jitter regulation is expensive (clock synch, timestamps)

Summary

- Two sorts of applications: best effort and guaranteed service
- Best effort connections require fair service
  - provided by GPS, which is unimplementable
  - emulated by WFQ and its variants
- Guaranteed service connections require performance guarantees
  - provided by WFQ, but this is expensive
  - may be better to use rate-controlled schedulers