Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet[0]

V.P. Kumar, T.V. Lakshman, D. Stiliadis
Bell Laboratories
Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733
USA

Abstract

With the transformation of the Internet to a commercial infrastructure, the ability to provide differentiated services to users with widely varying requirements is rapidly becoming as important as meeting the massive increases in bandwidth demand. Hence, while deploying routers, switches, and transmission systems of ever increasing capacity, Internet Service Providers would also like to provide customer specific differentiated services using the same shared network infrastructure. In this paper, we describe router architectures that can support the two trends of rising bandwidth demand and rising demand for differentiated services. We focus on router mechanisms that can support differentiated services at a level not being contemplated in proposals currently under consideration due to concern regarding their implementability at high-speeds. We consider the types of differentiated services that service providers may want to offer and then discuss the mechanisms needed in routers to support them. We describe plausible implementations of these mechanisms (the scalability and performance of which have been demonstrated by implementation in a prototype system) and argue that it is technologically possible to raise considerably the level of differentiated services that service providers can offer their customers, and that it is not necessary to restrict differentiated services to rudimentary offerings even in very high-speed networks.

I. Introduction

The transformation of the Internet into an important and ubiquitous commercial infrastructure has not only created rapidly rising bandwidth demand but also significantly changed consumer expectations in terms of performance, security, and services. Consequently, service providers need to not only evolve their networks to higher and higher speeds but also need to plan for the introduction of services of increasing sophistication, so as to address the varied requirements of different customers. At the the same time, Internet Service Providers (ISPs) would like to maximize the sharing of the costly backbone infrastructure in a manner that enables them to control usage of network resources in accordance with service pricing and revenue potential. The two trends of rapidly rising bandwidth demand and rising need for differentiation has resulted in intense efforts to build fast packet forwarding engines and in efforts to define mechanisms for service differentiation.

The need to build fast forwarding engines is being addressed in a variety of ways. We discuss these in various sections of the paper. The focus of this paper, however, is on gigabit routers capable not only of fast forwarding but also of providing customer-specific service differentiation. We consider a general and flexible model of service differentiation and do not restrict ourselves to any of the specific proposals for service differentiation that are being discussed in various industry and research forums. With a general model in mind, we discuss in detail the various router mechanisms necessary for providing differentiated services. In particular, we discuss architectural constraints imposed by the need to provide differentiated services instead of just fast forwarding, and also the three important components of packet filtering, buffer management and scheduling necessary to support differentiated services at the network element level. For each of the mechanisms, we discuss implementation considerations drawing upon experience gained from a prototype implementation. The theme is that at the network element level, it is technologically viable to incorporate mechanisms that can provide customer-specific differentiated services even at very high speeds. Consequently, from the point-of-view of implementability there is no need to restrict service offerings to simple schemes encoded in the Type-of-Servce (TOS) bits.

II. Differentiated Services

Consider what a company with multiple sites might expect, ideally, from an ISP providing connectivity amongst these sites. One possible scenario is that it would expect to receive, at the very least, a service similar to that obtained with a leased line network. This means that any two corporate sites communicating over the shared interconnecting backbone should perceive a performance at least equal to that of a leased line of some minimum pre-specified capacity. Traffic between sites should not be affected by the traffic of other users on the shared network. Queuing delays and congestion should occur only because of traffic generated by the individual company, and not because of the overall load generated by others. Isolating traffic from different customers and providing minimum bandwidth guarantees in a customer specific manner, allows customers of ISP services to determine the bandwidth they require to satisfy their needs based on their own traffic requirements, just as they would with a leased line. They may want the additional flexibility of being able to specify the manner in which their internal traffic, from different sources, is allowed access to the available bandwidth. Furthermore, it may be desirable to define different levels of service for different types of traffic in a customer-dependent manner. For example, some customers may consider FTP or Web transfers to be low priority, and so for them the service provider could aggregate multiple flows of these types. However, other customers may define Voice-over-IP or database queries to be high priority, and therefore for them the service provider must ensure good performance by giving these traffic types priority or guaranteed minimum bandwidth from within that customer's available bandwidth.

In another scenario, some customers may require extremely reliable and predictable performance for a small set of applications. They may indicate this requirement to the network as dynamic reservations of exact bandwidth along with specification of delay bounds. This indication may be by explicit signaling using a resource reservation protocol like RSVP, or may be done implicitly by some other means. The service provider's infrastructure must be capable of allowing end users to choose such a stringent, although possibly expensive, service if they require it. Considering scenarios such as the above, we can generate some guidelines regarding user and service provider requirements which can then be used in the design of router mechanisms for supporting differentiated services.

A. Customer Requirements

Customers would like the network to provide a leased-line like behavior while being able to use any additional available bandwidth. Since customers typically consolidate all existing network connections out of a site into a single connection to an ISP, they would like to have the added flexibility of dividing bandwidth available to them in a particular manner amongst their internal users and also by traffic type. This enables them to effectively support applications such as telephony, LAN interconnect, audio-conferencing and video-conferencing, Web hosting, Internet access, etc., while at the same time being able to allocate available bandwidth flexibly amongst internal users.

B. Service Provider Requirements

While meeting the customer requirements, service providers want to maximize the sharing and multiplexing of their infrastructure so as to maximize their revenues. They must be able to take advantage of the possibility for statistical multiplexing.

Service providers should be able to identify traffic belonging to different customers so that customer-specific differentiation can be done. In conjunction with the need to provide differentiated services is the need to manage the associated revenue through customer-specific traffic accounting and billing. This is necessary so that different service level assurances can be charged appropriately.

Furthermore, service providers should have the flexibility to distinguish themselves from other service providers by being able to tailor their service offerings in whatever competitive manner they choose to do.

All of these imply that routers must allow controlled resource sharing that permits service providers to maximize the utilization of their network while meeting diverse customer requirements.

C. Current Internet Architecture

The prevalent Internet service model is the best-effort model (also known as the so-called *send and pray* model). This model does not permit users to obtain better service, no matter how critical their requirements are, and no matter how much they may be willing to pay for better service. Clearly, with the increased use of the Internet for commercial purposes, this model of no differentiation is not satisfactory since the ISPs do not have the means to meet an existing market demand.

The Internet architecture has been successful up to this point, and the best-effort service model has been adequate. The current Internet architecture evolved from a network which was providing reliable connectivity, despite link failures, to thousands of users. Now, the Internet is an international network with millions of users. Yet, some of the control mechanisms that were originally implemented to prevent congestion collapse and which depend on cooperative users are still in use. The network infrastructure did not incorporate any means for differentiation other than by severely under-utilizing specific portions of the network. Over-provisioning network bandwidth and keeping the network lightly loaded in order to support differentiated services for select customers is not a cost-effective solution, and cannot be achieved at all times. Furthermore, for a commercialized network, relying on cooperation of many disparate users for congestion control is not desirable. It cannot be assumed that users will cooperatively slow down their transmission rates when significant congestion is detected. Assuming cooperative behavior leads to several problems, especially when the network has no means for isolating traffic from different users or of enforcing cooperation. Among the problems are the following:

1. End users ignore the expected and correct usage of TCP transmissions by disabling their flow-control mechanisms (either purposely or due to incorrect implementation). This degrades performance for complying users.
2. Malicious users can easily degrade network performance by routing traffic through specific paths that increase local congestion. (an example is some of the recently reported denial-of-service attacks such as *smurfing*).
3. An increasing number of real-time applications use UDP without congestion control. For some applications, it is not optimal to adapt to congestion in a TCP-friendly manner. Applications may even increase their transmission rates during lossy periods by introducing more forward error correction. Such non-cooperative behavior can quickly lead to poor performance to many users unless the network has sufficient mechanisms for isolating and penalizing such users.

D. Improving on Best Effort

It may be argued that best effort may be good enough with appropriate provisioning. Nevertheless, some form of differentiation is still desirable. When there is a focused overload to parts of the network (such as when a popular Web site is heavily accessed, or some event not accounted for in traffic engineering happens) routers must have mechanisms to treat different customers differently. When such events happen, there are just not enough resources available to give reasonable service to everybody.

Providing any form of differentiation usually requires the network to keep some state information. This requirement translates to increased memory requirements and processing power. The majority of the installed routers use architectures that will experience a dramatic decrease in performance if mechanisms are added to provide sophisticated differentiating features. It was safe to assume that with existing architectures, implementing any sophisticated functionality in the network was either impossible at the speeds required or would be prohibitively expensive. Therefore, the commonly held belief is that all of the sophistication should be in end systems and that the core network should be as simple as possible,

However, there has been a shift toward incorporating more intelligence in the network. The current discussions about differentiated services within the IETF, for example, assume that the edge routers of a core network are smart enough to classify the packets of different users.

E. Requirements for Sophisticated Differentiated Services

Although traditional routers are limited in terms of their quality-of-service and differentiation features, recent research and advances in hardware capabilities have provided mechanisms to overcome these limitations. From the router perspective, the tools for providing differentiated services are based on the following operations that can now be done at high speeds:

1. Packet classification that can distinguish packets and group them according to their different requirements.
2. Buffer management that determines both of the following: how much buffer space should be given to certain kinds of network traffic and which packets should be discarded in case of congestion.
3. Packet scheduling that decides the packet servicing order so as to meet the bandwidth and delay requirements of different types of traffic.

High-speed implementation is made possible by the following:

1. New developments in the areas of fair-queuing and per-flow scheduling have resulted in simple and efficient algorithms for managing the complexity of scheduling decisions. These algorithms have been implemented in ATM switches showing that they can be implemented at very high speeds even for packets as small as an ATM cell.
2. State information can be managed using both aggregation and protocol specific information. In the context of TCP, the argument has been that flows are too numerous in the backbone for per-flow information to be maintained. However, many of these TCP flows are not continually backlogged in the router queues. By providing support for tens of thousands of queues, which can be done without being very expensive, the performance requirements of a much higher number of TCP flows can be met by taking advantage of statistical multiplexing.
3. The argument that route look-up and packet classification are expensive operations is becoming less valid as new research in these areas provides fast and efficient algorithms. New schemes can examine the complete information in the IP packet header and use it as a method of classification. When combined with fast switching fabrics, they can remove the processing bottlenecks of traditional routers.

III. Processing Models for Forwarding Engines

In this section, we first describe architectural features that are necessary in IP forwarding engines designed to provide differentiated services. Then, after showing why traditional router architectures cannot offer differentiated services, we present an example routing architecture that was designed specifically with the new requirements in mind.

Forwarding engines must, at the very least, be able to identify the context of packets, determine the next hop IP and link-layer addresses, and assign for each packet the appropriate buffering and output link bandwidth. The forwarding engine is the prime point of control in determining the manner in which packets are handled within the router.

A. The Requirement for Real-Time Operation

The first requirement for differentiated services is that the maximum delay for header processing must be no larger than the delay that a packet from the service class with the least delay can experience. This is because it is the header processing which determines the service level to be accorded to a packet and hence violation of service assurances prior to header processing cannot be allowed. Since, the tolerable delay for some packets might be almost zero if we allow constant-bit-rate circuit emulation type flows, we get the following architectural constraint:

> *There should be no queueing before header-processing.*

This implies that packet header processing must be done at wire-speeds and must not be traffic dependent. Traditionally, routers have been using cache-based methods for forwarding packets. The justification is that packet arrivals are temporally correlated and can be grouped into connections or flows [13], [29], so that if a packet belonging to a new connection arrives then more packets belonging to that same connection can be expected to arrive in the near future. More precisely the definition of a connection is:

> A sequence of packets between the same source and destination with interarrival times of less than 60 seconds.

The reasoning in favor of cache-based architectures is that packets in the Internet are usually initiated by applications and form connections [28]. With this expected behavior, the first packet of an application can be processed through a slow path that analyzes the complete header. The header of the packet is then inserted into a cache or hash table together with the action that must be applied to this first packet as well as to all other packets of the same application. When subsequent packets of that flow arrive, the corresponding action can be determined directly from the cache or hash table.

There are three main problems associated with this architecture:

1. In current backbone routers, the number of connections that are active at a given interface is extremely high. Recent studies have shown that an OC-3 interface might have an average of 256K connections active concurrently. [52]. For this many connections, use of hardware caches is extremely difficult, especially if we consider the fact that a fully-associative hardware cache is required. So caches of such size are most likely to be implemented as hash tables since only hash tables can be scaled to these sizes. However, the $O(1)$ look-up time of a hash table is an average case result, and the worst-case performance of a hash table can be poor since multiple headers might hash into the same location. The number of bits in the packet headers that must be parsed is typically between 100 and 200 bits, and even hash tables are limited to only a couple of million entries. So, any hash function that is used must be able to randomly distribute 100 to 200 bit keys of the header to no more than 20-24 bits of hash index. Since there is no knowledge about the distribution of the header values of the packet that arrive to the router, the design of a good hash function is not trivial.
2. Due to the large number of connections that are simultaneously active in a router and due to the fact that hash tables generally cannot guarantee good hashing under all arrival patterns, the performance of cache based schemes is heavily traffic dependent. If a large number of new connections arrive at the same time, the slow path of the system that implements the complete header processing can be temporarily overloaded. This will result in queueing of packets before they are processed. When this happens, no intelligent mechanism can be applied for buffering and scheduling of these packets because without header processing there is no information available about either the destination of the packets or about any other fields relevant to differentiation. So it is possible that congestion, packet dropping, and service quality violation happen due to processing overload and not due to output link congestion.
To better illustrate this consider the model in Figure 1. Packets arrive at the interfaces and are placed in a queue for processing. After the packet classification and next-hop lookup operations are performed, they are forwarded to the outgoing interfaces where they are queued for transmission. With this architecture, some interfaces may be idle even when there are packets destined to those interfaces waiting in the input queues. For example, all packets destined for output 1 can be blocked in the slow path processing module behind packets that are destined to other outputs. Output 1 remains idle, although there are packets in the buffers available for transmission. Obviously such a system will suffer from a type of Head-of-Line blocking that will limit the router throughput substantially. Note that the traditional Head-of-Line blocking problem in input buffered switches can be diminished, if there is knowledge about the destination of more than one packet in the queue [36]. However, the fundamental problem with the system of Figure 1 is that nothing is known about the context of packets which are queued for header processing. Thus it is impossible to apply any intelligent queueing mechanisms in the input queues and this form of Head-of-Line blocking cannot be eliminated*
3. A commercial Internet infrastructure should be robust and should provide predictable performance at all times. Variations in the throughput of forwarding engines based on traffic patterns are undesirable. In addition, the network should not be vulnerable to attacks from malicious users. A mali-

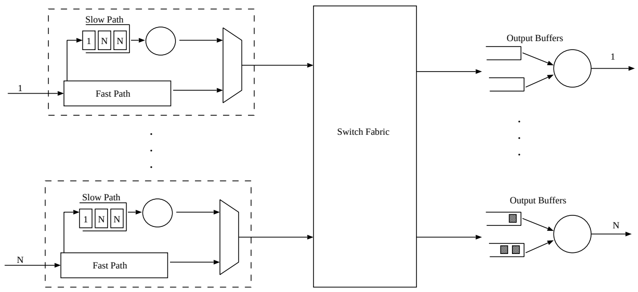

Fig. 1. Queueing model of a system that uses a cache-based architecture for packet classification.

*It is to be emphasized that the head-of-line blocking we refer to is not the traditional head-of-line blocking at the switch fabric. What we mean is that output interfaces may be idle even though there are packets destined to those interfaces waiting in the input *header processing buffers* .

cious user or group of users discovering the limitations of the hash algorithms or caching techniques, can generate traffic patterns that force the router to slow down and drop a large portion of the packets arriving at a particular interface.

Summarizing, we claim that packet queueing delays are acceptable only after header processing, when provisioning of differentiated services and robustness are important. This *no-queueing before processing* principle applies because it is the header processing operation that enables the router to determine the service level to be accorded to a particular packet. Hence large queues formed while waiting for the header processing operation can violate service assurances for some connections, even before the router determines the service level to be accorded to the connection. The implication that this has on the design of forwarding engines is that it is the **worst-case performance** of the forwarding engine that determines the true maximum packet processing rate and not the average case performance (the averaging being done on traffic arrival patterns). If average case performance is used to determine supported packet processing speeds, then buffering is needed before processing. Unless the variance in processing times of the header processing functions is very small, the delay in this pre-processing buffer can cause service assurances to be violated.

B. Criteria for real-time packet processing and system constraints

We now list, based on the prior discussion, the criteria that a differentiated-services capable fast router must meet:

1. The router must be fast enough to support Gigabit links. Internet service providers envisage to build networks with link capacities of 2.4 Gigabits/sec and more.
2. The forwarding engines of the routers must be able to process every packet arriving to the physical interfaces at wire-speed. Recent traffic studies have shown that 75% of the packets are smaller than the typical TCP packets size of 552 bytes. In addition, nearly half the packets are 40 to 44 bytes in length, comprising of TCP acknowledgments and TCP control segments [52]. Since the forwarding engines cannot use buffering to absorb the variation in processing times, they must operate at wire-speed when all packets are as small as 40 bytes. This means that the forwarding engines must have provably small worst-case processing times which are independent of traffic patterns.
3. The processing engines of the backbone must be able to perform classification and packet filtering operations as well as next-hop look-up operations. In order to make the routers and the networks manageable both the packet filtering and route look-up operations must be based on aggregated rules such as CIDR aggregations [21], [58]. Memory accesses are expensive and are the dominant factor in determining the worst-case execution time.
4. For operation at very high speeds processing algorithms must be amenable to hardware implementation.

C. Overview of Router Architectures

We give a brief overview of some of the most popular router architectures. Then, we show how they fail to meet most of the requirements that are imposed by the dual needs of high-capacity and differentiated services.

C.1 Single Processor - Shared Bus

The first generation of routers were based on a single general-purpose CPU and an intelligent real-time operating systems. The choice of this architecture was premised on the fact that protocols at that time were constantly changing and multi-protocol operation meant that routers could not be optimized for a specific protocol. Establishing and maintaining connectivity was far more important than high forwarding capacity.

These routers consisted of a general-purpose computer and multiple interface cards interconnected through a shared bus. Packets arriving at the interfaces were forwarded to the central processing engine which determined the next hop address and sent them back to the appropriate outgoing interfaces. Therefore each packet is transferred twice over the shared bus: once from the interface cards to the processor, and once from the processor to the outgoing interface card. Routing and other control protocols were also implemented in the same forwarding engine. Obviously, the performance of such a router depends heavily on the throughput of the shared bus and on the forwarding speed of the main processor. Since a single processor must perform multiple real-time operations, the selection of the operating system is of critical importance and a lot of emphasis was placed on the sophisticated design of the operating system. This architecture could not scale to meet the increasing throughput requirements of the interface cards.

C.2 Multiple Processor - Shared Bus

The straightforward improvement introduced in the second generation of routers was to distribute the forwarding computation. Packets are therefore transmitted only once over the shared bus. Interface cards became more intelligent with the addition of fast processors and caches. This allowed them to process packets locally some of the time. The first packet of a connection is forwarded to the main forwarding engine. A cache entry is added to the interface card cache and this allows subsequent packets of the same connection to be forwarded directly between the interfaces. The cache entry is on a per-connection basis or on a per-route basis [43]. The argument for the latter was that even if the number of connections is very large, the number of routes that are needed is probably not large. However, in the network core where the highest forwarding speeds are required, it is unlikely that this will be true since packets are likely to be destined to many parts of the network and not be localized to specific subnets. Indeed, it is more likely that packets are distributed to every other router in the network. This architecture clearly has a traffic dependent throughput and the shared bus is still a bottleneck.

As memory sizes increased and cost dropped, the distributed interface cards were enhanced with larger memories and complete forwarding tables were copied to each interface card. This approach further enhanced the performance of these routers. However, the shared bus and the general purpose CPUs can neither scale to high capacity links nor provide traffic pattern independent throughput.

C.3 Multiple Processors - Space Switching

The third generation of routers were designed to alleviate the two obvious bottlenecks of the second generation of routers. The congested shared bus was replaced by a switch fabric providing sufficient bandwidth for transmitting packets between interface cards and allowing throughput to be increased by several orders of magnitude. With the forwarding path between interface cards being not the bottleneck anymore, the new bottleneck is packet processing. Most third generation router architectures used general purpose CPU designs that could not handle large link capacities.

C.4 Shared Parallel Processors - Space Switching

A nice concept for parallelizing packet processing so as to be able to scale processing speeds considerably was introduced in [2], [3]. The basic observation is that it is highly unlikely that all interfaces will be bottlenecked at the same time. Hence, sharing of the forwarding engines can increase the port density of the router. This shared processor architecture is presented in Figure 2. The forwarding engines are only responsible for resolving next-hop addresses. The results of this next-hop address resolution are sent back to the interface cards that subsequently forward the packets to the

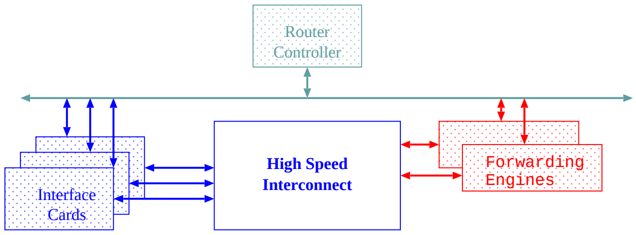

Fig. 2. Router architecture with multiple parallel forwarding engines that are separated from the interface cards. A load balancing mechanism attempts to equalize the processing load of all forwarding engines.

outgoing interfaces. Note that only packet headers are forwarded to the forwarding engines. This eliminates an unnecessary packet payload transfer over the switch fabric. Packet payloads are always directly transferred between the interface cards and they never go to either the forwarding engines or the controller unless they are specifically destined to them.

It is possible that if enough forwarding engines are added to such a system, it can reach the necessary forwarding capacities for high-speed backbones. However, the assumption on which this architecture is premised is that a single processor is unlikely to be able to handle the peak throughput of a particular interface and therefore a processing pool shared by multiple interfaces is necessary. The time required to process each packet depends on the actual load of each forwarding engine. Any load balancing mechanism must be done on a per-connection basis; out-of-order transmissions can trigger TCP fast retransmits and degrade performance. Therefore, the time-scale at which load balancing is done (the time-scale of connection interarrival times) can be an order of magnitude or more different from the time-scale of packet arrivals. If after a connection has been assigned to a processor, the load of the processor increases (because it is hard to predict the traffic generated by each connection when load balancing decisions are done) then packets could be queued for long periods before processing and cause violation of service assurances for that connection. It can be claimed that if the number of forwarding engines is sufficiently large then with high-probability there will be a free forwarding engine. Although this is true, it is not clear how cost-effective it is. Note that a copy of the forwarding databases must be kept at each forwarding engine and they must all be kept consistent. Amongst the architectures discussed so far, this architecture is the most promising one†.

†This architecture has also been studied for use in processing signaling protocols. Useful load balancing schemes and a detailed perfomance analysis is presented in [26]

Next, we present a router architecture that can scale to very large forwarding speeds while providing worst-case traffic independent bounds on processing times. This is made possible by having each processing element perform a specific task which is particularly easy for IP-only routers.

D. Optimizing Packet Processing

It is evident that the increasing link capacities and the need for differentiated services stretch processor based architectures to the limit. The multiprocessor architecture, described in the previous section, attempts to address these issues by using several forwarding engines each of which can perform packet processing for packets arriving on particular sets of links. In this section, we develop a more economical and efficient solution that is based on a functional partition of packet processing with each processing element optimized to a specific task and operating at wire-speed in a traffic independent manner.

The first step necessary for this approach is to identify the main functions that comprise the forwarding process. Some of the key functions are enumerated below (each of these are discussed in detail in subsequent sections):

1. Buffer and forward packets through a switchg fabric.
2. Apply filtering and packet classification.
3. Determine the next hop of the packet. Use the forwarding databases that are created by the routing protocols to determine which outgoing interface the packet is to be sent to and what the specific link layer addresses are.
4. Queue the packet in an appropriate queue based on both the classification decisions and the route table look-up.
5. Schedule packet transmission on outgoing links so as to meet service requirements.

These processing functions use as inputs the packet header and some data structures which are created by routing protocols, network management, RSVP, etc. When the same processing element executes all of the above functions, it must have access to the packet header and to all these data structures. Thus, when multiple processors are used for forwarding packets and every packet is completely processed by one of them, all of these data structures must be up-to-date in every processor. Obviously, the synchronization problems for achieving this are non-trivial (Again, see [26] for a performance analysis of this problem in the context of processing signaling protocols). In addition, since each packet requires accesses to different data bases, there is no locality of reference in the data structures required

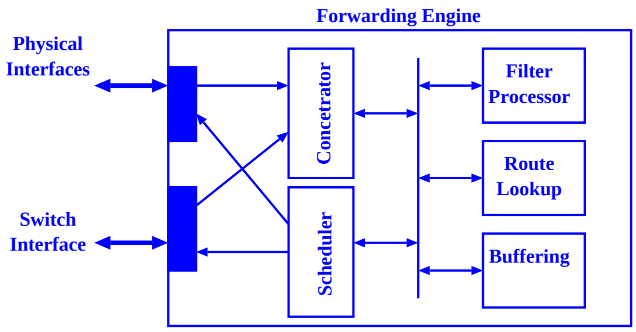

Fig. 3. Processing architecture where parallelism is achieved by partitioning forwarding functions amongst different processors.

by the processors. Since all processors have to keep all the forwarding information and data structures, the memories required are large and thus generally slow.

There is an easier way, however, to exploit the special requirements of packet processing so as to increase performance and reduce cost. This architecture is shown in Figure 3. Packets arrive at the interfaces and their processing is functionally split among multiple processors so that each processor handles one of the packet processing functions necessary for forwarding. For the forwarding functions described above, five processors are required. Since each processor performs only one type of function, each processor needs only the data structures that are associated with a particular function. It is not necessary to keep all information related to forwarding in every processor. Thus, the memory required for each processor is much smaller and hence can be much faster and cheaper. Specializing the task that each processor performs makes it possible to get a high locality of reference for program memory accesses.

Additional advantages of such an architecture are as below:

1. There is no connection-level assignment of processing units to connections. For the previously discussed multiprocessor architecture, such an assignment is needed for load balancing and this has several disadvantages that were pointed out. With this architecture, flow identification is only required for storing packets in the output link queues.
2. The importance of the operating system is diminished. Each processor is a single processing element that is dedicated to one function.
3. When higher throughput is desirable, Application Specific Integrated Circuits (ASICs) can replace processing elements easily. The one-function only nature of each processor enables efficient hardware implementation.

Note here that a combination of the above architecture with the multiple shared-processor architecture is possible as well, and may be very cost-efficient. One minor problem might be, that communication between interfaces and processors might add some additional constant latency to the packet forwarding. As we will see in the subsequent sections, each of these individual functions can be implemented at wire-speeds in a traffic independent manner even at backbone link rates. Hence, this architecture supports the twin requirements of both high speeds and differentiated services.

A variant of the above architecture was developed and implemented as part of the Bell Labs Router project. A multi-gigabit router was designed where each processing engine can process up to 1 million packets per second. The processing elements were implemented using Field-Programmable Gate Arrays (FPGAs) and the system speed was only 33MHz. Obtaining such high throughput with differentiated services features using a low clock speed is evidence of the scalability of this architecture.

IV. Route Table Look-up

For a long time, route table look-up was considered one of the most challenging operations performed during the forwarding process. The problem is defined as that of searching through a data base of destination prefixes and locating the longest prefix that matches the destination address of a given packet. Longest prefix matching was introduced as a consequence of the requirement for increasing the number of networks addressed through Classless Inter-Domain Routing (CIDR) [21].

The first approaches for longest prefix matching used radix trees or modified Patricia trees [45], [19] combined with hash tables. These look-up algorithms have complexity proportional to the number of address bits which, for IPv4 is only 32. Hence, even with these algorithms, a route look-up can complete in less than a microsecond using 30ns memories resulting in a forwarding rate of 1 million packets per second. If a radix greater than 2 is used, the time required is even lower though this is at the expense of some wasted memory space.

Early implementations of routers, however, could not afford such expensive computations. So, most routers relied either on connection or route caches [22], [43]. Cache based architectures, however, result in unpredictable and traffic dependent performance. Furthermore, with the rapid growth of the Internet, it is not clear anymore how effective caches are in network backbones.

Recent research has resulted in even faster solutions. The stratified trie algorithms proposed by Van Emde Boas in [41], [6] were utilized by De Berg, Van Kreveld and Snoeyink to propose algorithms for one dimensional point location, when the problem is restricted to discrete valued ranges, with $O(\log N)$ complexity, where N is the number of bits required to represent the range end-points [5]. The same technique was recently used in the context of route table look-ups in [56]. The main idea is to first create a perfect hash table of prefixes for each prefix length. A binary search among all prefix lengths, using the hash tables for searches amongst prefixes of a particular length, can find the longest prefix match in $O(\log N)$ time. Although the algorithm has very fast execution times, calculating perfect hashes can be slow and can slow down updates.

Another elegant approach was proposed in [17]. The basic principle is to create a small and compressed data structure, exploiting the sparseness of actual entries in the space of all possible routing table entries, that represents large look-up tables using a small amount of memory. This results in a lower number of memory accesses to a fast memory and hence faster look-ups. In addition, the algorithm does not have to calculate expensive perfect hash functions, although updates to the route table are still not easily done. A more efficient approach for implementing the compression and minimizing the complexity of updates was presented in [54].

V. Packet Filtering and Classification

One of the most important requirements for forwarding engines that support differentiated services is the ability to identify the context of packets and apply the actions necessary to satisfy service requirements. Mechanisms to support the above functions are termed *packet filtering* or *packet classification*.

The traditional application of packet filters has been for providing firewall and security functions. However, another important application, in support of differentiated services, is the identification and classification of packets originated by specific sites, customers, or applications so as to provide the resources necessary for meeting customer-specific differentiated services requirements. Large scale packet filtering functionality enables both edge routers, as in [14], and core routers to support differentiated services that are more flexible and customer-specific than those restricted to service offering based on interpretation of the TOS bits alone [38].

Packet filters should parse a large portion of the packet header, including information concerning transport protocols, before forwarding decisions are made‡. The parsing is based on a set of rules that are defined

‡In the industry forwarding decisions based on transport protocol information is referred as layer-4 forwarding.

either by network management software or by real-time reservation protocols such as RSVP. The actions that packet filters may apply can include the traditional security functions, such as dropping of unauthorized packets, redirection of packets to proxy servers, etc., as well as actions related to queueing, scheduling, and routing decisions using not only destination addresses but also source addresses, source port numbers, destination port numbers, etc.

Two desirable attributes for packet filtering are:

1. Filter rules must apply to ranges of addresses, port numbers, or protocols, and should not be restricted to exact matches. This allows rules to apply to aggregates and keeps the number of rules to be speficied manageable. If filter algorithms can only handle exact matches, then preprocessing must translate ranges in filter rules to exact values. This is infeasible since the size of the ranges grows exponentially with the length of the packet field on which the ranges are defined.
2. Rules must be assigned explicit priorities in order to resolve conflicts if rules overlap.

A. General Packet Classification Problem

The general packet classification problem can be viewed as a point location problem in multidimensional space, where given a set of n d-dimensional objects (that represent the filter rules) and a point in a d-dimensional space (that represents the arriving packet) the problem is to find the object that contains the point. This is a classic problem in computational geometry and numerous results have been reported in the literature [9], [10], [16], [40]. When considering the general case of $d > 3$ dimensions, as is the problem of packet classification, the best algorithms considering time or space have either an $O(\log^{d-1} n)$ complexity with $O(n)$ space, or an $O(logn)$ time-complexity with $O(n^d)$ space. [40]. Though algorithms with these complexity bounds are useful in many applications, they are not directly useful for packet filtering. To illustrate this, let us assume that we would like the router to be able to process 1K rules of 5 dimensions within $1\mu s$ (to sustain a 1 million packets per second throughput). An algorithm with $log^4 n$ execution time and $O(n)$ space requires 10K memory accesses per packet. This is impractical with any current technology. If we use an $O(\log n)$ time and $O(n^4)$ space algorithm, then the space requirement becomes prohibitively large since it is in the range of 1000 Gigabytes. Furthermore, none of the above algorithms addresses the problem of arbitrary overlaps.

However, the above results, do not mean that packet filtering cannot be done at high speeds given the constraints of the problem when applied to routers. We now sketch ideas that were used for a prototype implementation. Interested readers are referred to [33] for details of the implemented algorithm.

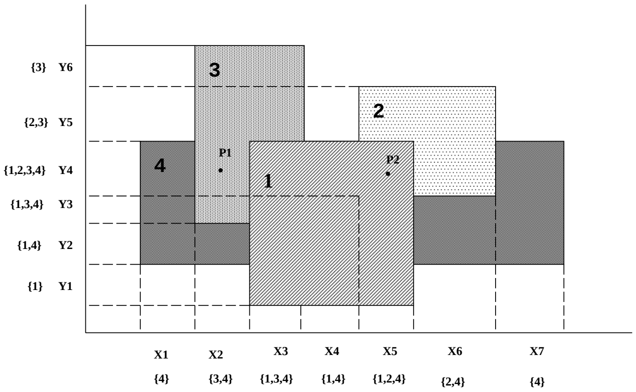

Fig. 4. General packet classification using set intersection

A simple approach to the problem of multidimensional search, as used for packet filtering, is to use decomposable search. Here the idea is to state the original query as the intersection of a few numbers of simpler queries. The challenge in obtaining a poly-logarithmic solution, is to decompose the problem such that the intersection step does not take more time than the required bound. However, as was pointed out before, even a $log^4 n$ solution for 5 dimensional packet filtering is not practical for our application where n can be in the thousands. Therefore, we need to employ parallelism of some sort. Moreover, we require simple elemental operations to make the algorithm amenable to hardware implementation. Instead of algorithms with the best asymptotic bounds, we are interested in decomposing the queries such that sub-query intersection can be done fast (with memory-access as cost metric) for n in the thousands and memory word-lengths that are feasible with current technology.

We will illustrate the algorithm with an example in 2-dimensions. (Figure 4). Rules are represented by 2-dimensional rectangles that can arbitrarily overlap. The preprocessing step of the algorithm projects the edges of the rectangles to the corresponding axis. In the example shown, the four rectangles create seven intervals in each axis. In the worst case, the projection results in a maximum of $2n + 1$ intervals on each dimension. We next associate a set of rules with each dimension. A rule belongs to the set if and only if the corresponding rectangle overlaps with the interval that the set corresponds to. Note that because of the method by which the intervals were created, it is not possible for a rectangle to overlap, say, only with half an interval. Assume that the packet represented by point $P1$ arrives to the system. During the first on-line step, we locate the intervals in both axes that contain this point. In the example, these are intervals X2 and Y2 for the X and Y axis respectively. In the second step, we use the sets to locate the highest priority rectangle that covers this point by a simple intersection operation.

The algorithm has been implemented in 5 dimensions in the Bell Labs Router prototype using a single FPGA device and Synchronous SRAMs chips supporting up to 512 rules and processing 1 million packets per second in the worst case. This is achievable despite the device being run at a very low speed of 33 MHz. Since we used the same memory chips as those used in the L2-caches of personal computers, the cost of the memories is low. The device can be used as a co-processor, next to a general purpose processor that handles all the other parts of IP forwarding or firewall functions. An improved algorithm presented in [33] can be used to process $8K$ filter rules at 1 million packets per second using a 66Mhz clock.

B. Classification in Two Dimensions

The 2-dimensional classification problem is of increasing importance in the evolving Internet architecture. Although RSVP, or similar reservation protocols, can offer end-to-end service guarantees for specific flows, it is not clear whether such a reservation based model can be scaled for operation over high-speed backbones where a very large number of flows can be concurrently active. An alternative approach that has been proposed is route aggregated flows along specific traffic engineered paths. This directed routing is based not only on the destination address of packets, but on the source address as well [34]. RSVP or similar reservation protocols can be used for setting the aggregation and routing rules [34], [7]. The key mechanism needed to support such a scheme in a high-speed core router is a 2-dimensional classification or lookup scheme that determines the next hop, and the associated resource allocations, for each packet as a function of both the source and destination addresses. Two-dimensional classification is useful not only for setting-up traffic engineered source-destination paths, but also for multicast forwarding which requires lookups based on both the source address and multicast group [20], [55].

The two dimensional look-up problem is a restricted case of the general classification problem where the rules in at least one of the two dimensions (source or destination address) are defined in terms of CIDR prefixes. Rules can still have arbitrary overlaps and priority amongst rules is used to resolve conflicts. For IP routers, we are interested in solutions that can scale to hundreds of thousands of entries. Also, as for all other operations, we are interested in only worst-case performance of the algorithms since we want to avoid queueing for header processing in order to provide service assurances.

A fast algorithm for this problem is presented in [33]. It is shown that for a number of possible prefix lengths of 32 and $n = 2^{18} = 256K$ filter rules, the number of memory accesses is about 50 which makes it possible to process at a million packets per second with a less than a 100 Mhz clock speed. Combining fast filtering in many dimensions with source-destination based routing widens the range of options feasible for evolving the current best-effort Internet to the Internet of the future capable of providing customized differentiated services. Specifically, there may be no need to restrict filtering to the edges or to very simple operations such as using only the TOS bits in the IP packet header.

VI. Resource Management

The previous sections discussed how processing power, one of the scarce router resources, should be managed to meet the new differentiated services requirements of the Internet. An important principle was the *"no queueing before processing"* principle and consequent architectural choices for worst-case engineering. The other critical resource is output link bandwidth. Coupled with management of output link bandwidth is buffer allocation. This section is focused on bandwidth and buffer management.

Buffer and bandwidth allocation must be jointly addressed. A scheduler that does dynamic bandwidth allocation merely gives opportunities to flows for link access. If a flow does not have any waiting packets to be transmitted, this opportunity is wasted (though some schedulers retain these lost opportunities as credits for future use). For adaptive flows, it is especially important that provision of transmission opportunities (i.e., scheduling) be coupled with proper buffer allocation that ensures that these flows use the link access opportunities and so achieve a desired level of link sharing.

A desired level of link sharing is set for classes of packets using administrative means or by dynamic reservations using protocols such as RSVP. Packet filtering mechanisms map each arriving packet into one of these classes. Schedulers determine the sequence of packet transmissions on the link taking into account the desired link sharing, the packets from different classes currently in the system, and a history of the recent service sequence (which in a fair queueing scheduler is summarized by a system potenital [27], [47]).

We distinguish between two types of flows that impact scheduling:

1. *Non-Adaptive* flows whose packet arrival rates are independent of past scheduling decisions. Examples are open-loop UDP traffic such as perhaps video or audio streams. The bandwidth requirements for these flows are set by provisioning mechanisms, or established by signaling. Provisioning is usually done at an aggregate level, i.e., the requirement might be that a given amount of bandwidth must be assured for flows of a certain type between two sites in a network. Bandwidth establishment by signaling can be done using RSVP which allows setting up aggregate and flow-level end-to-end reservations. Irrespective of the mechanism, resource control mechanisms are required in every node to enforce the reservations.
2. *Adaptive flows* that adapt their transmission rates based on scheduling decisions made in the recent past. TCP flows are adaptive flows that constantly probe for available bandwidth by increasing their transmission rates in response to available network bandwidth. Adaptive flows, in particular TCP flows which are loss sensitive in their adaptation, require careful buffer management in conjuction with scheduling. The need for TCP-aware buffer management has been pointed out in [8], [50], [51], [35]. We discuss TCP-aware buffer management in detail later in the paper.

A further requirement that is imposed by the commercialization of the Internet is that user-specific differentiated service models must be supported since the same infrastructure is to be used for serving different user requirements. The state-of-the-art candidate scheme for bandwidth allocation to meet differentiated services requirements is by hierarchical link sharing using heirarchical weighted fair queueing. This is discussed next.

A. Hierarchical Link Sharing

Hierarchical link sharing is a flexible resource management scheme that allows partitioning of resources at several levels of aggregation. It was initially proposed in [15], [25] and later expanded upon in [4]. Hierarchical link sharing partitions bandwidth into a hierarchy of classes, where each class is defined to be a set of flows with some specific properties, such as packets that belong to the same protocol, or packets traveling between specific subnets etc. The levels of the hierarchy can be made to include, at the lowest level, packets from an individual application.

The hierarchical link sharing model, as proposed in [25], is illustrated in Figure 5. Here, the main idea is that link capacity is partitioned amongst different administrative domains, and within each domain it can be further partitioned based on application types. Going a step further, the bandwidth of each application type can be further partitioned amongst individual flows of that type. A different view of this model is that the link capacity is partitioned between classes of traffic, and the bandwidth of each traffic class is further partitioned between administrative domains or individual flows.

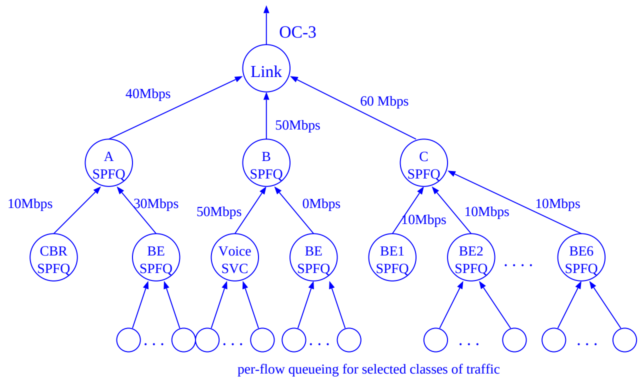

Fig. 5. Application of hierarchical link sharing. Each node can be either a Shaped Virtual Clock (SVC) or a Start-Potential-based Worst-case Fair Queueing (SPFQ) scheduler.

Our prototype implements Hierarchical Weighted Fair Queueing (HWFQ) which has the extra advantage that it uses the same schemes across all hierarchy levels and thus makes the implementation simpler. The model presented in [4] was extended, using the theory of Shaped Rate-Proportional Servers [48], so that each node in the hierarchy can be any Shaped Rate-Proportional Server. Then, a node in the hierarchy may share excess bandwidth with the other nodes at the same level, or it may shape traffic to a particular bandwidth allocation. In our implementation, each node in the hierarchy is allowed to use a Shaped Virtual Clock that provides ideal shaping of the traffic, or use Starting-Potential based Fair Queueing that allows fair distribution of the bandwidth among competing classes. Both algorithms are Shaped Rate-Proportional Servers and they use exactly the same data structures [48].

The advantage of this general mechanism is best explained with an example. Assume that for one administrative domain we want to support two types of traffic: Constant Bit Rate and best-effort. For a second administrative domain, we want to give voice traffic full priority over best-effort traffic. Finally, for a third adminstrative domain, we would like different levels of best-effort service. The assignment of schedulers and weights is shown in Figure 5. We assign a Virtual Clock scheduler to the CBR traffic, and a Fair Queueing Scheduler to the best effort traffic of the first domain (A). This mechanism allows the best-effort traffic to get any of the excess bandwidth available from other classes. For domain B, we assign all the bandwidth to the high priority voice traffic. As a result, best effort traffic will only be serviced when voice traffic is not present. For the third domain, we assign different weights to the different levels of best effort service.

The advantage of the above scheme is that we can use the same data-structures and algorithms for providing a variety of services to different administrative domains or applications.

The next question is whether we should do any aggregation at all at the flow level. If it is possible to implement per-flow queueing for a large number of flows we can derive many advantages. Clearly, flow isolation is a benefit of per-flow queueing. Enhanced TCP performance by combining per-flow queueing and TCP-aware buffer management is another benefit. This is discussed in detail later.

B. Scalability of per-flow queueing

The prevailing assumption is that per-flow queueing is prohibitively expensive because routers must keep state for all flows and there can be hundreds of thousands of active flows in network backbones. The notion of active flow used is that a flow is active as long as the interarrival time of any two packets between the same source and destination is less than 60 seconds [37]. This notion is useful when the purpose is to study mechanisms that ensure that that only a small number of packets are forwarded to the slow path and that most of them should be handled through a cached fast path. The problem with using this as the basis for maintaining flow state for scheduling purposes in per-flow queueing schemes is that this is overly conservative. Maintaining state for each such active flow is not necessary to provide the guarantees of fair queueing except in some extreme situations.

There are mainly two types of state that a per-flow queueing mechanism must maintain: i) the weights of individual flows and ii) information about the service offered to flows. When providing differentiated services, in most cases, weights are established for a set of flows with given characteristics and not for individual flows. An example might be differentiating between TCP flows which are ftp-like and TCP flows which are telnet-like. The packet filtering function performs the differentiation and it will typically use filter rules that classify many flows into a class rather than have a separate class per flow. Once the weight for a class is established, each flow must use a different queue to enable the per-flow queueing mechanism to isolate flows and to guarantee the desired differentiation. As discussed below, the state information necessary to provide this isolation is less than that required to maintain state for every active flow.

Ideal Weighted Fair Queueing

Weighted Fair Queueing schedulers [42], [18], emulate a fluid-flow system within the constraints of a packet system. In an ideal fluid flow system, packets are infinitely divisable and all flows that are backlogged are simultaneously served with a rate that is proportional to the assigned weights. In such a scheduler, the instantaneous service offered to an individual flow is only determined by the weight assigned to this flow and by the weights assigned to all other flows that are currently backlogged. Thus, there is no need for the scheduler to keep state for any flow that is not currently backlogged. Since the number of backlogged flows is limited by the available buffering, the number of flows for which state has to be maintained is also limited to numbers determined by buffer availability.

Non-ideal Weighted Fair Queueing

For packet fair-queueing schedulers to emulate an ideal fluid-flow system, they have to maintain all the state information of the fluid-flow system as well as information related to the state of the packet system. This is because a packet scheduler cannot emulate a fluid-flow system exactly since packets from different flows cannot be served at the same time. In the worst-case, the set of flows that are backlogged in the reference fluid system and the set of flows that are actually backlogged in the packet system may be completely disjoint [49]. In this case, the maximum state information that a packet fair queueing system must maintain is approximately twice that of the fluid-flow system. This can still be much lower than the number of active flows.

In [49], a class of schedulers has been defined that use simple mechanisms to extract the state of the flow-fluid system from the actual state of the packet system. This information is used to schedule packets in the packet system [27], [49]. These schedulers associate a function with each flow or connection, called the connection potential, that tracks the normalized service offered to the connection while the connection is active.[§] A system potential [27], [49] is used to keep track of the normalized service offered to all flows during the current system busy period. When a flow becomes newly backlogged, its flow potential is calculated as the maximum of its previous value and the value of the system potential. For some types of schedulers, the difference between the connection potential of any non-backlogged flow and the system potential can be bounded. In these schedulers, in addition to the backlogged flows, state

[§]Normalized service is defined as the ratio of offered service to allocated bandwidth.

must be maintained for a small subset of flows that were backlogged during the immediate past. Furthermore, in the case of Self-Clocked Fair Queueing (SCFQ), the system potential is always larger than the connection potential of any non-backlogged flow [27]. Thus, for SCFQ, state must be maintained only for the backlogged flows (refer to [51] for a more detailed discussion).

Summarizing, for an ideal weighted fair queueing scheduler or an SCFQ scheduler, in the worst case, the number of flows is equal to the number of packets in the system. To illustrate this with an example, assume that a router supports an OC-12 link (622Mbits/sec bandwidth). Assume that the average packet size is 256 bytes. Then assuming a buffer size equivalent to 200ms, it is easy to verify that the maximum number of flows that can be backlogged in the router is no more than 64K, which is a very reasonable number as per current technology limitations.

B.1 TCP-aware Buffer Management

Since TCP controlled traffic constitutes a significant component of Internet traffic, it is natural to incorporate mechanisms in routers that enhance TCP performance. The following general criteria acts as a guideline. For long TCP flows (transfers much larger than the obtained bandwidth-delay product) we would like to keep the link share as close to the desired one as possible¶. For short transfers, we would like small queueing delays for some TCP flows such as *telnet* flows. We would like to achieve good TCP performance even in the presence of cross-traffic which is not TCP controlled, or is uncontrolled, or controlled in a manner that is unfair to TCP sources. Also, we would like to keep the link utilization as high as possible.

TCP Observations

Traffic burstiness: TCP creates bursty network traffic. This is because data transmission in TCP is ack clocked and acks bunch together in FIFO queues to create burstiness [44], [59]. Also, TCP slow-start is very bursty due to the doubling of the window sizes every round-trip time. Losses in the slow-start phase can lead to very poor throughput and buffering at least equal to approximately one-third the bandwidth delay product is needed to avoid losses in the slow-start phase [30]. However, to keep throughput high, especially for long transfers, a generally accepted thumb-rule is to have buffering equal to at least one bandwidth delay product.

With such large buffers, packets from quasi-delay-sensitive applications, like telnet, may queue up behind a large slow-start burst from a file transfer and experience large queueing delays despite the use of mechanisms like RED [24]. Hence it would be good to separate telnet-like sources from others.

Unfairness: It is known that TCP is inherently unfair to connections with long-round trip times [23] and the unfairness can sometimes be as bad as the inverse square of round-trip times [30]. This implies the use of active TCP-aware buffer management in routers to alleviate unfairness as suggested in [8].

Synchronization: Another reason for the use of TCP-aware buffer management is that with drop-tail queueing, TCP windows can synchronize leading to poor and oscillatory link utilization [57]. It is necessary to reduce synchronization by appropriate buffer management.

Random Loss Sensitivity: Since TCP assumes that every loss is a congestion loss and reduces its transmission rate drastically, TCP throughput is very vulnerable to loss. For a fixed loss-rate, the throughput goes down as the inverse square of the bandwidth-delay product [30], [39]. This loss sensitivity is further increased if there is a slow reverse path or if the ack path is congested. The sensitivity increase is proportional to the ratio of the transmission time of ack packets in the reverse path to that of data packets in the forward path [31]. Random losses can be caused by transient fast (faster than round-trip times) fluctuations in open-loop traffic such as uncontrolled UDP traffic. Buffer management must protect TCP sources from losses caused by TCP-unfriendly sources, i.e., from sources which do not react to losses by reducing transmissions as fast as TCP.

Is Per-Flow Fair Queueing Sufficient for Achieving TCP Performance Goals?

Per-flow fair queueing provides fair opportunities-to-transmit. However, if a flow does not have a backlog (due to it having small windows for instance) often enough then fair opportunities-to-transmit do not translate to fair link sharing [50]. Proper buffer management is needed in conjunction with fair queueing to ensure that TCP flows share the available bandwidth in a fair manner (or in a controlled unfair manner if that is desired).

The link-sharing goals for TCP are to achieve i) high throughput, ii) fairness in bandwidth sharing among competing TCP connections, iii) protection of conforming connections from malicious users, iv) reduce the well-known ack-compression phenomenon for TCP [44] that causes network traffic to be bursty even for smooth sources, and v) provide low latency to telnet like applications that are delay-sensitive and use TCP.

Is Buffer Management Alone Sufficient?

In the absence of per-flow queueing, control of link-

¶For ease of exposition, we only explain the case where link sharing within a class is equal, i.e., fair link sharing.

sharing is achieved by buffer management and packet discard schemes like Random Early Detection (RED) and drop-from-front. Though some of the above goals can be achieved, the lack of per-flow state hampers the degree to which the goals are achieved. However, the above goals can be achieved much better with a combination of per-flow queueing and appropriate buffer management [50].

TCP-aware Buffer Management for use with fair queueing

Merely dividing available buffer space and using RED on each queue is not appealing since buffer space is wasted drastically. For good buffer usage, we would like to share buffer space between different flows. Using RED on this shared buffer space breaks the natural flow isolation that per-flow queueing provides (see [50] for an example) and allows TCP connections to be affected by TCP unfriendly sources. Also, fair-queueing with uncontrolled buffer usage by each flow leads to unfairness.

A solution is to give each flow a nominal reserved buffer space. These nominal allocations can be set in proportion to weights (or ratio of bandwidth to round-trip times if they are known). Each flow's buffer usage can exceed reservation if unused buffer space is available. However, if the total buffer occupancy exceeds a global threshold and a new packet arrives then some packet already in the buffer must be pushed out. The candidate queues eligible for pushout are those queues whose occupancy is above their nominal allocation. From this candidate set, we pick the queue that has the most excess occupancy (i.e., longest deviation from nominal alocation). The reason we pick this longest queue is that TCP flows with short round-trip times and high bandwidth usage are likely to have the longest queues above nominal since their windows grow faster than flows with longer round-trip times. We alleviate unfairness to long round-trip time connections by dropping packets from this longest queue. Also, TCP-unfriendly flows are likely to have long queues. Within the longest queue, we use drop-from-front [32].

Drop-from-front triggers TCP's fast retransmit feature faster reducing the length of congestion episodes and improving link utilization. It improves fairness even with FIFO queueing, and reduces chances of windows synchronizing. It is also a useful mechanism for dropping acks [31] (in case of ack congestion) since TCP acks are cumulative. Also, when used with fair-queueing it almost eliminates the chance of retransmitted packets being lost [50]. Loss of retransmitted packets leads to expensive TCP time-outs and consequent loss of throughput.

This buffer management scheme protects TCP

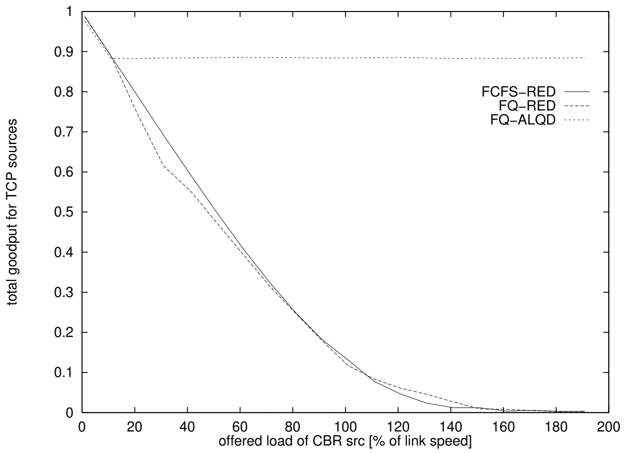

Fig. 6. Aggregate throughput for 10 TCP sources (2 to 160ms RTT) sharing a link with a loss-insensitive constant bit-rate source.

sources from TCP unfriendly flows. Hence, with this buffer management, there is no need for other applications (such as reliable multicast or RTP applications) to have TCP-friendly congestion control. Those applications can adapt their traffic in whatever manner is preferable to them without having to comply with TCP's congestion control mechansim. TCP-aware buffer management eliminates the need for other applications to be TCP-aware. Another useful aspect is that this buffer management allows TCP to have larger initial windows (hence speeding up completion of short transfers) without affecting other TCP flows.

Protection from TCP-unfriendly sources The protection that the above buffer management scheme gives to TCP sources is illustrated in Figure 6.

Figure 6 shows the fraction of the link bandwidth used by 10 persistent TCP sources sharing a link with a loss-insensitive TCP-unfriendly source. (Details are in [50].) The sending rate of this unfriendly source is varied from zero to twice the link bandwidth. With a consistent overload produced by the TCP-unfriendly source, the buffer management and dropping strategy have a stronger impact on bandwidth sharing than the scheduling policy itself.

When the TCP-unfriendly source increases its rate, for both first-come-first-serve (FCFS) and fair queueing with global RED there is very similar TCP-source performance degradation. However, with longest queue drop policies(marked ALQD in figure to denote an approximated longest queue implementation), a fair queueing scheduler is able to almost perfectly maintain the guaranteed rate of $\frac{1}{11}$ per flow, irrespective of whether the flow is TCP-like or is totally loss insensitive.

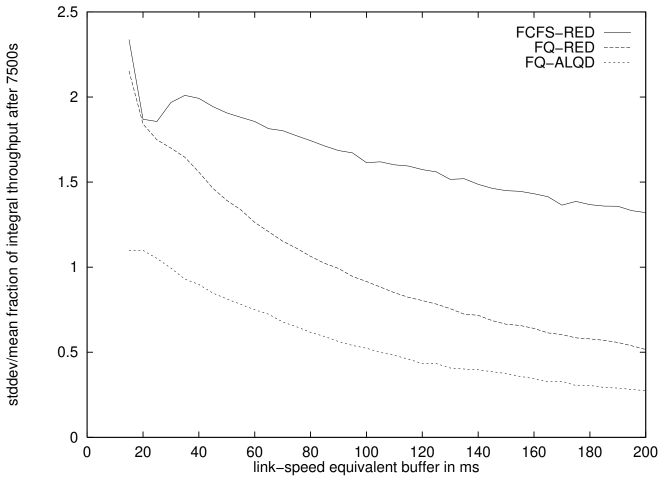

Fig. 7. Coefficient of Variation versus Buffer Size for many TCP connnection with differing round-trip times. FCFS-RED is First-Come First-Served scheduling with Random Early Detect; FQ-RED is Fair Queueing with Random Early Detec; FQ-ALQD is Fair Queueing with Longest Queue First drop and drop-from-front.

Figure 7 shows the coefficient of variation of the throughput of different TCP flows when they share a common link with fair-queueing and with RED or the buffer management scheme proposed in the previous section. Clearly, the proposed buffer management scheme has better fairness than RED.

B.2 Implementation Issues

It has been a long standing assumption that implementation of large scale per-flow queueing and hierarchical link sharing is not possible at reasonable cost with current technologies. Implementation of per-flow queueing and hierarchical link sharing in the Bell Labs Router has shown that it is indeed possible to implement these mechanisms at the necessary scale at high speeds. A primary limitation in applying intelligent buffer management and scheduling mechanisms was that processing elements were performing all functions for header processing which limited throughput as functionality increased. Distribution of processing on a functional basis and not on per-packet basis, as shown in Figure 3, allows throughput to be increased considerably while increasing functionality as well. Furthermore, the state information needed for per-flow queueing is not prohibitive as has been assumed. In the Bell Labs Router prototype, we have demonstrated that per-flow queueing and hierarchical link sharing can be implemented, on a scale necessary for backbone routers, using a single FPGA device running at 33MHz and supporting output link capacities up to 1Mpps.

VII. Switch Fabric

Switch fabric design is a very well studied area, especially in the context of ATM networks [53], [12], [11], [36], [1], [46], and a detailed discussion of switch fabric design issues is omitted for brevity. The key point to keep in mind is that transport through the fabric must be done in a service preserving manner.

VIII. Discussion and Conclusions

With the Internet's on-going evolution from a best effort network to a network with service differentiation, an important topic of interest is the type of differentiated services that service providers may want to provide. One possibility is to encode in the TOS bits a set of canonic differentiated service types and incorporate only these mechanisms into the network infrastructure. However, it is not clear whether this simple and evolutionary approach can satisfy differentiated services demands.

The point of view taken in this paper is that it is possible to incorporate in routers mechanisms that, in addition to enabling simple differentiated services, which have been proposed recently, can also enable more sophisticated services. The required mechanisms are:

1. All header processing is done at wire speed: "No queueing before processing"
2. Packet classification with range matching on many fields for a large number of rules, extend route lookups to source and destination based lookups.
3. Flexible and general scheduling and buffer management techniques.
4. Mechanism for flow isolation.
5. Switch fabric capable of preserving service guarantees.

Using these mechanisms routers can aggregate customers in a very flexible manner, isolate individual customer's traffic, and allocate resources in a customizable manner. This permits each ISP to customize its service offerings to suit its customer demands by offering services such as Virtual Private Networks with customized service differentiaton, virtual leased lines, etc. Restricting router functionality to simple differentiated services possibilities due to technology constraints is not necessary. Current technology enables sophisticated differentiated services to be offered at a level heretofore thought not practical.

Acknowledgements

The authors would like to thank members of the Bell Labs Router team that was responsible for building a prototype router encompassing many of the ideas discussed in this paper. We would like to thank, in partic-

ular, R. Arunachalam, S. Rathnavelu, K. J. Singh, B. Suter, and H. Tzeng for their many contributions.

References

[1] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, 11(4):319–52, November 1993. Also as Digital Systems Research Center Technical Report, No. 99.

[2] A. Asthana, C. Delph, H. Jagdish, and P. Krzyzanowski. Design of a gigabit ip router. Technical Report 11251-911105-09TM, AT&T Bell Laboratories, November 1991.

[3] A. Asthana, C. Delph, H. Jagdish, and P. Krzyzanowski. Towards a gigabit ip router. *Journal of High-Speed Networks*, 1(4), 1992.

[4] J.C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of ACM SIGCOMM'96*, pages 143–156, Palo Alto, CA, August 1996.

[5] M. De Berg, M. van Kreveld, and J. Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18:256–277, 1995.

[6] P. Van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[7] J. Boyle. RSVP Extensions for CIDR Aggregated Data Flows. In *Internet Draft*, http://www.internic.net/internet-drafts/draft-ietf-rsvp-cidr-ext-01.txt, 1997.

[8] B. Braded, D. Clark, J. Crowcroft, B. Davie, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K.K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. Internet Draft, March 1997.

[9] B. Chazelle. How to search in history. *Information and Control*, 64:77–99, 1985.

[10] B. Chazelle and J. Friedman. Point location among hyperplanes and unidirectional ray shooting. *Computational Geometry: Theory and Applications*, 4:53–62, 1994.

[11] F.M. Chiussi. Design, performance and implementation of a three-stage banyan-based architecture with input and output buffers for large fast packet switches. Technical Report CSL-TR-93-573, CSL, Stanford, CA, June 1993.

[12] F.M. Chiussi, J.G. Kneuer, and V.P. Kumar. Low-cost scalable switching solutions for broadband networks. *IEEE Communications Magazine*, 35(12):36–43, December 1997.

[13] K.C. Claffy. *Internet Traffic Characterization*. PhD thesis, University of California, San Diego, 1994.

[14] D. Clark. Service Allocation Profiles. In *Internet Draft*, http://www.internic.net/internet-drafts/draft-clark-diff-svc-alloc-00.txt, 1997.

[15] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM '92*, pages 14–26, August 1992.

[16] K.L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987.

[17] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small forwarding tables for fast routing lookups. In *Proceedings of ACM SIGCOMM'97*, France, September 1997.

[18] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1(1):3–26, 1990.

[19] W. Doeringer, G. Karjoth, and M. Nassehi. Routing on longest-matching prefixes. *IEEE/ACM Transactions on Networking*, 4(1):86–97, February 1996.

[20] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast - sparse mode : Protocol specification. In *RFC 2117*, June 1997.

[21] V. Fuller et. al. Classless Inter-Domain Routing. In *RFC1519, ftp://ds.internic.net/rfc/rfc1519.txt*, June 1993.

[22] D.C. Feldmeier. Improving gateway performance with a routing-table cache. In *Proceedings of IEEE INFOCOM'88*, New Orleans, LI, March 1988.

[23] S. Floyd and V. Jacobson. On traffic phase effects in packet switched gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[24] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[25] S. Floyd and V. Jacobson. Link sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[26] D. Ghosal, T.V. Lakshman, and Y. Huang. Parallel architectures for processing high-speed network signaling protocols. *IEEE/ACM Transactions on Networking*, pages 716–728, December 1995.

[27] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM '94*, pages 636–646, April 1994.

[28] R. Jain and S. Routhier. Packet trains — measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4:986–995, 1986.

[29] K.Claffy, C. Polyzos, and H.W.Braun. Application of sampling methodologies to network traffic characterization. In *Proceedings of ACM SIGCOMM'93*, pages 194–203, September 1993.

[30] T.V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, June 1997.

[31] T.V. Lakshman, U. Madhow, and B. Suter. Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance. In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan 1997, April 1997.

[32] T.V. Lakshman, A. Neidhart, and T.J. Ott. The drop-from-front strategy in TCP over ATM and its interworking with other control features. In *Proceedings of IEEE INFOCOM'96*, pages 1242–1250, San Francisco, CA, 1996.

[33] T.V. Lakshman and D. Stiliadis. Packet classification algorithms for gigabit internet routers. Technical Report 113470-980202-02T, Lucent Technologies - Bell Laboratories, January 1998.

[34] T. Li and Y. Rekhter. Provider Architecture for Differentiated Services and Traffic Engineering (PASTE). In *Internet Draft*, http://www.internic.net/internet-drafts/draft-li-paste-00.txt, 1998.

[35] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM'97*, September 1997.

[36] N. McKeown, V Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of INFOCOM'96*, pages 296–302, March 1996.

[37] P. Newman, T. Tylon, and G. Minshall. Flow labelled ip: A connectionless approach to atm. In *Proceedings of IEEE INFOCOM'96*, pages 1251–1260, San Francisco, CA, April 1996.

[38] K. Nichols and S. Blake. Differentiated services operational model and definitions. Internet-Draft, February 1998. http://ds.internic.net/internet-drafts/draft-nichols-dsopdef-00.txt.

[39] T.J. Ott, M. Mathis, and J.H.B. Kemperman. The stationary behavior of ideal tcp congestion avoidance. ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps, 1996.

[40] M. H. Overmars and A.F. van der Stappen. Range searching and point location among fat objects. *Journal of Algorithms*, 21(3):629–656, 1996.

[41] P. Van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of 16th IEEE Conference on Foundations of Computer Science*, pages 75–84, 1975.

[42] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of INFOCOM '92*, volume 2, pages 915–924, May 1992.

[43] C. Partridge. Locality and route caches. In *NSF Workshop on Internet Statistics Measurement and Analysis*, San Diego, CA, February 1996.

[44] S. Shenker, L. Zhang, and D.D. Clark. Some observations on the dynamics of a congestion control algorithm. *Computer Communication Review*, pages 30–39, October 1990.

[45] K. Sklower. A tree-based routing table for Berkeley Unix. Technical report, University of California, Berkeley, 1993.

[46] D. Stiliadis and A. Varma. Providing bandwidth guarantees in an input-buffered crossbar switch. In *Proceedings of INFOCOM '95*, April 1995.

[47] D. Stiliadis and A. Varma. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. In *Proceedings of ACM SIGMETRICS '96*, pages 104–115, http://www.cse.ucsc.edu/research/hsnlab/publications/, May 1996.

[48] D. Stiliadis and A. Varma. A general methodology for designing efficient traffic scheduling and shaping algorithms. In *Proceedings of IEEE INFOCOM'97*, pages —, 1997.

[49] D. Stiliadis and A. Varma. Rate-proportional servers: A design methodology for fair queueing algorithms. *IEEE/ACM Transactions on Networking*, April 1998.

[50] B. Suter, T.V. Lakshman, D. Stiliadis, and A. Choudhury. Design considerations for supporting tcp with per-flow queueing. In *Proceedings of IEEE INFOCOM'98*, San Francisco, CA, 1998.

[51] B. Suter, T.V. Lakshman, D. Stiliadis, and A. Choudhury. Efficient active queue management for internet routers. In *Proceedings of Interop Engineers Conferece*, Las Vegas, NV, May 1998.

[52] K. Thomson, G.J. Miller, and R. Wilder. Wide-area traffic patterns and characteristics. *IEEE Network*, December 1997.

[53] F. A. Tobagi. Fast packet switch architectures for broadband integrated services digital networks. *Proceedings of the IEEE*, 78(1):133–167, November 1990.

[54] H.-Y. Tzeng. Longest prefix search uisng compressed trees. Technical Report TM, Bell Laboratories, under submission, 1997.

[55] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. In *RFC1075, ftp://ds.internic.net/rfc/rfc1075.txt*, June 1993.

[56] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed ip routing lookup. In *Proceedings of SIGCOMM'97*. ACM, September 1997.

[57] L. Zhang. *A new architecture for packet switching network protocols*. PhD thesis, M.I.T. Comput. Sci., Cambridge, MA, 1989.

[58] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7(5):8–18, September 1993.

[59] L. Zhang, S. Shenker, and D.D. Clark. Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic. In *Proceedings of ACM SIGCOMM'91*, pages 133–147, 1991.