INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On TCP Performance in an Heterogeneous Network: A Survey

Chadi Barakat — Eitan Altman — Walid Dabbous

N° 3737

July 1999

THÈME 1

Rapport de recherche

ISSN 0249-6399

On TCP Performance in an Heterogeneous Network: A Survey

Chadi Barakat , Eitan Altman , Walid Dabbous

Thème 1 — Réseaux et systèmes
Projet Mistral , Rodeo

Rapport de recherche n° 3737 — July 1999 — 33 pages

Abstract: The new transmission media used to transport Internet traffic present different characteristics from the network TCP is tuned to. This results in a degradation in the performance of the protocol in terms of resource utlization and transfer delay. Many works have studied the performance of the protocol over these new transmission media. The majority of these works were interested in a particular environment such as a satellite network or a mobile network. A large number of solutions have been proposed to improve TCP performance. In this paper, we summarize the main problems of TCP and the proposed solutions. The originality of this work is that we conduct the study independently of the network type. Instead of talking about particular transmission supports, we consider the different possible characteristics of the connection path. We study then the effect of these characteristics on TCP mechanisms. This analysis permits us to present an understanding of TCP problems, the limitations of the actual solutions and the required modifications to let TCP cope with an heterogeneous Internet on an end-to-end basis.

Key-words: TCP, Flow and Congestion Control, Error control, Performance, Problems, Solutions.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Un Survey sur les performances de TCP dans un réseau hétérogène

Résumé : Les nouveaux supports de transmission utilisés pour transporter le trafic Internet présentent des caractéristiques différentes du réseau pour lequel TCP a été conçu. Ceci a résulté en une dégradation de performance du protocole en terme d'utilisation des ressources et du temps de transfert. Plusieurs travaux ont étudié les performances de TCP en présence de ces nouveaux supports de transmission. La plupart de ces travaux se sont intéressés à un support particulier comme un lien satellite ou bien un lien terrestre sans fil. Un grand nombre de solutions ont été proposées pour améliorer la performance du protocole. Dans ce rapport, on résume les principaux problèmes de TCP et les solutions proposées. L'originalité de ce travail est qu'il a été mené indépendamment du type du support de transmission. Au lieu de parler de certains supports, on considère les différentes caractéristiques possibles du chemin traversé par la connexion TCP. On étudie ensuite l'effet de ces caractéristiques sur les mécanismes de TCP. L'analyse nous a permis de comprendre les vrais problèmes de TCP, les limitations des solutions proposées et les modifications nécessaires pour rendre TCP capable de s'adapter à un réseau hétérogène tout en respectant le principe de bout en bout.

Mots-clés : TCP, Contrôle de flux et de congestion, Contrôle d'erreurs, Performance, Problèmes, Solutions.

1 Introduction

The *Transmission Control Protocol* (TCP) [38] plays a crucial role in the operation of the Internet. It provides a reliable connection-oriented in-order transport service to many of today's applications (Web, Email, Ftp, Telnet, News, etc.). Also, it provides a control of their flows so as to avoid congestion in the network and overload at the receiver. With the simple best effort service provided by the Internet layer, TCP must be able to cope with the different characteristics of the transmission media forming the Internet, to provide a good service to applications and to use efficiently the available resources in the network. At the beginning, TCP has succeeded to accomplish these tasks. However, the increase in the heterogeneity of the Internet has caused a severe degradation in its performance. Internet traffic is now carried by new transmission media with completely different characteristics from the network TCP is tuned to. High speed links (Fiber Optic), long and variable delay paths (GEO and LEO satellite links), lossy links (Wireless networks), asymmetric paths (Hybrid satellite networks) and others, are becoming widely embedded in the Internet. Many researchers have studied by experimentations [4, 6, 8, 10, 11, 13, 23], analytical modeling [5, 17, 29, 30, 35] and simulations [1, 16, 18, 25, 24, 33, 41], the performance of TCP in this new environment. Some of the works have focused on the performance of TCP on a general path of a certain characteristic. An example of such works are those aiming to improve the loss recovery capacity of TCP [19, 33, 34]. Others have analyzed the protocol in a particular environment such as Satellite networks [3, 23], Mobile networks [8, 11], etc. These works have shown some problems in the operation of the protocol. Long propagation delay and losses on a satellite link, handover and fading in a wireless network, bandwidth asymmetry in some media, and other phenomena, have been shown to affect seriously the throughput of a TCP connection. Many solutions have been proposed to improve the performance of TCP. Some of these solutions propose modifications to TCP algorithms to help the protocol adapt itself to these new paths [1, 2, 4, 6, 9, 19, 24, 25, 28, 33]. They respect the main principle of TCP, an end-to-end protocol able to use efficiently the resources of any kind of networks. The other solutions keep the protocol unchanged and try to improve the performance by hiding the problem from TCP [3, 8, 10, 11, 23, 32]. Usually, modifications inside the network are proposed and these modifications vary according to the problem to hide.

In this paper, we summarize the different works on TCP performance. Our aim is to conduct the study independently of the type of the network causing the problem. Instead of talking about particular environments, we consider the different possible characteristics of a path crossed by a TCP connection. We characterize the path of a connection by four quantities: the Bandwidth-Delay Product (BDP), the propagation delay, the rate of losses not caused by congestion and the degree of bandwidth asymmetry between the forward and the reverse direction. By Bandwidth-Delay Product (BDP) of a path, we mean the product of the *bottleneck bandwidth* on this path [1] times the two-way propagation delay. With these four quantities, we are able to study the effect of the different new transmission media on TCP performance. For every quantity, we study its impact on TCP mechanisms and how much it impairs the performance of the protocol. Solutions proposed to reduce the impact of this quantity, are presented next to the problem.

In the next section, we present an overview of TCP mechanisms and algorithms. The following four sections explain the impact of every one of the four path characteristics we have defined and the associated solutions. Section 7 concludes our paper. In this section, we try to summarize the main problems of TCP and the required modifications.

2 Overview of TCP

TCP provides a control of application flows so as to avoid congestion in the network and overload at the receiver. It also provides a control of errors by retransmitting any lost packet. *Flow control* in TCP is based on a window that limits the maximum number of bytes the source can send before receiving any acknowledgment from the receiver. By varying the size of this window, the throughput of a TCP connection can be set to the appropriate value. At any time, this throughput is equal approximately to the window size divided by the Round Trip Time (RTT) of the connection. *Error control* in TCP is based on the use of sequence numbers, retransmission timers and cumulative positive acknowledgments (ACK). The destination acknowledges data packets. An ACK carries (implicitly) the sequence number of the last in-order byte received. It is therefore a cumulative information robust against ACK loss. This information is also used by the source to infer any transmission error. With these ACKs, the source tries to estimate the RTT of the connection and sets accordingly a timer when packets are sent. The expiration of this timer, called a *Timeout*, before the receipt of an ACK is considered by the sender as a loss signal. The first packet sent and not yet acknowledged is directly retransmitted. Another mechanism has been added to TCP to detect losses using ACKs without the need to wait for a long Timeout. It is called *Fast Retransmit* [40]. According to this mechanism, the receipt

[1] Paxon in [37] defines the bottleneck bandwidth as the upper bound on how fast any connection can transmit along the path due to the data rate of the slowest forwarding element along the path.

of four consecutive ACKs carrying the same sequence number (the original plus three Duplicate ACKs) indicates the loss of the packet following the last packet acknowledged. The number four is chosen to minimize the probability that a reordering of packets causes a wrong error detection.

Congestion control has been added to TCP by Jacobson [26]. An additive-increase multiplicative-decrease strategy has been adopted to change the window size, and therefore the throughput, as a function of network conditions. The window used for flow control is always taken as the minimum of the congestion window and the one advertised by the receiver. Starting from one segment [2], or a larger value as we will see later, the congestion window is increased exponentially by one segment for every new ACK until the source estimation of the network capacity is reached. By capacity of the network, we mean the maximum number of packets (or of bytes) that can be fit on the path between the source and the destination. This is called sometimes the *pipe size*. It is equal to the Bandwidth-Delay Product plus the available buffer size. This fast increase of the window is called *Slow Start* and the capacity estimation is called the slow start threshold, denoted W_{th} in the sequel. The name slow start comes because, even though it is a fast increase, this increase is still slow compared to a direct transmission at a window of size W_{th}. Given that TCP transmits the packets allowed by its window in a burst, starting directly at the estimation of the network capacity would result in a long burst and a heavy overload on network buffers. Slow start aims to alleviate this burstiness while trying to fill as quickly as possible the network capacity. Moreover, the capacity of the network may be overestimated. A loss then occurs during slow start before reaching W_{th}. Here, the source sets W_{th} to half the current window size. Slow start serves then as a means to get a closer estimation of the available capacity in the network. This estimation role of slow start is more pronounced at the beginning of the connection where no idea on the network is available. Most TCP implementations suppose that the receiver is the bottleneck and set W_{th} to the window it advertises.

Once W_{th} is reached, the source switches to a slower increase of the congestion window by one segment for every window's worth of acknowledgments. This is the *Congestion Avoidance* algorithm. In contrast to slow start which aims to fill as quick as possible the network capacity, congestion avoidance aims to probe slowly the network for any extra bandwidth. The probe continues until a loss occurs. Here, the source supposes that the network is getting into congestion and it sets its estimation of the capacity to half the current size of the window. The first version of TCP that implements congestion control, called Tahoe [26], sets the congestion window

[2] A segment is the size of data in a TCP packet.

at this point to one and uses again slow start to reach the new estimation of the capacity. Slow starting after every loss deteriorates the performance given the low bandwidth utlization during the slow start phase. But, slow start is still required when the loss is detected via Timeout. A Timeout means that the arriving of ACKs has stopped (otherwise the loss is detected with Fast Retransmit) and thus a slow start is required to fill again smoothly the available capacity. In the Fast Retransmit case, the other versions of TCP (Reno [40], New Reno [19], SACK [16], FACK [33], etc.) have proposed the use of the ACK stream that keeps flowing after the detection of the loss in order to recover from losses without slow starting. When three duplicate ACKs are received, these versions call a *Fast Recovery* algorithm. This algorithm tries to retransmit the losses in the current window while maintaining a number of packets in the network equal to the new estimation of the capacity. Once losses are recovered, this algorithm ends and a normal congestion avoidance is called without the need to call slow start. If Fast Recovery fails, the ACK stream stops, a Timeout occurs and the source resorts to a slow start phase as in the Tahoe version.

3 Effect of a large BDP

The increase in link speed (ex. Optic Fibers at Gbps) has led to paths of high BDP. On these paths, TCP window must be able to reach large values in order to fill the network capacity and therefore to use efficiently the available bandwidth. Large windows are becoming possible with the definition of the window scale TCP option [28]. This new option carries a scale factor and the real window size in Bytes is equal to the value included in the old window field times the scale factor. This permits large windows up to 2^{30} Bytes. However, at large windows, a congestion period may lead to the loss of many packets. Further, operating at large windows increases the probability that a loss is detected via Fast Retransmit. Resorting to slow start as in Tahoe after every loss detection results in a poor performance given the long time taken by slow start to fill a path of a large capacity. An efficient Fast Recovery phase is then required in order to correct many losses in the same window and to avoid as much as possible the slow start phase. At large BDP, network buffers have also an important impact on the performance. These buffers must be well dimensioned and must scale with the BDP. In the next two subsections, we show how at large BDP, the Fast Recovery phase and network buffers may affect TCP performance. Note that the BDP may be large due to a long propagation delay as on satellite links. A long delay has other impacts on TCP. In this section, we are only focusing on the effect of the increase in the network capacity. The effect of the increase in the propagation delay is left for the next section.

3.1 Fast Recovery

In order to use efficiently the resources on a large BDP path, the Fast Recovery phase must be able to recover, as quick as possible, from many losses in the same window. Thus, slow start is most of the time avoided and the bandwidth utilization is kept at a high value. Fast Recovery is called when a loss is detected with Fast Retransmit. Some ACKs (Duplicate ACKs) still arrive at the source indicating that some packets are still reaching the destination. Thus, the network is not very congested and a slow start phase may be avoided. Fast Recovery tries to use the information carried by ACKs to estimate the number of packets in flight while recovering from losses. New packets are sent if this number falls below the network capacity estimate. The network capacity is estimated after the loss detection to half the window size when Fast Recovery is called. This continues until all the losses in the same window of the first loss are recovered. Losses in the same window are supposed to be caused by the same congestion event and therefore the window is reduced once by half. Note that if a loss is detected with Timeout, the ACK stream, called also ACK clock because it triggers the transmission of packets, stops and the pipe drains. A slow start is therefore required to refill smoothly the pipe.

The difference between the different versions of TCP which implement Fast Recovery is in the estimation of the number of packets in flight during the recovery phase. The Reno version of TCP [40] considers that every Duplicate ACK is a signal that a packet has left the network and it then injects a new packet if the window allows. However, Reno supposes that one packet is lost in a window and leaves Fast Recovery when an ACK for the first loss is received. Leaving Fast Recovery early may prohibit the source from detecting the other losses in the same window with Fast Retransmit. The number of new packets sent during recovery may be too small so that three Duplicate ACKs are not received in order to trigger again Fast Retransmit. A Timeout is then required to detect the other losses. Given the coarse granularity of TCP retransmission timer (measured in ticks of 500 ms), a Timeout results in a long period of inactivity and hence in a poor performance. Moreover, a timeout is followed by a slow start which is the behavior Reno tries to avoid. Note that this slow start is called with a smaller threshold due to the reduction in W_{th} at every loss detection. In the case of multiple losses per window, it has been shown that Reno performs worse than the Tahoe version that resorts always to slow start [16].

New-Reno [19, 25] has been proposed to overcome the problems of Reno when many packets are lost in the same window. The idea is to stay in Fast Recovery until all the losses in the same window are recovered. This avoids the Timeout and the slow start but cannot result in a recovery faster than one loss per RTT. This is because cumulative ACKs cannot inform the source of more than one loss per RTT. The source needs to wait for the ACK of the retransmission of a loss to discover the next loss in the same window. Moreover, relying on ACKs to estimate the number of packets in flight leads to a problem when ACKs are lost on the return path. The loss of ACKs results in an underestimation of the number of packets that have left the network thus in a low utlization of the bandwidth during recovery. A smaller number of packets than we intend is kept in flight. This results also in a large burst of packets when Fast Recovery ends.

More information is needed at the source to recover faster than one loss per RTT and to estimate more precisely the number of packets in the pipe. This information is provided by *SACK* (Selective ACK) [34]. SACK is a TCP option added to the header of an ACK packet and containing the three blocks of contiguous data most recently received at the destination. With this information, the source is able to detect more than one loss in a single RTT. Also, it can calculate more precisely the number of packets that have left the network without relying on the ACK stream. Many algorithms have been proposed to use this information during Fast Recovery. We find TCP-SACK [16] that uses ACKs for the estimation of the number of packets in the pipe and SACKs to retransmit more than one loss per RTT. This leads to an important improvement in the performance when bursts of losses appear in the same window. But the recovery is always sensitive to the loss of ACKs. As a solution we find FACK (Forward ACK) [33] that relies on the SACK information in the estimation of the number of packets in the pipe. Thus, FACK resolves the sensitivity of TCP-SACK to the loss of ACKs on the reverse path. The number and the identity of packets to transmit during the recovery phase is then decoupled from the ACK clock. The ACK clock is still used however to smooth the transmission of TCP packets. Later, we will see another proposition to decouple another mechanism of TCP from the ACK clock. The aim of these works is to make TCP operation insensitive to any disturbance on the feedback channel causing a loss of ACKs. The ACK clock is required as a means to smooth the transmission of packets, but in the presence of such disturbance, it doesn't reflect reliably what is happening on the forward direction. This will affect some TCP mechanisms that use ACKs to infer what is happening to data packets.

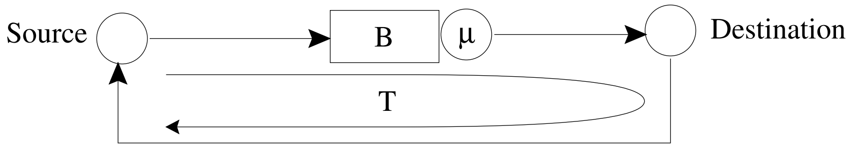

Figure 1: Single node network model

3.2 Impact of buffer size on the performance

The size of buffers in network nodes must scale with the BDP of the connection's path. If they are not appropriately sized, these buffers may affect seriously the performance of the protocol. The effect is different between slow start and congestion avoidance.

The exponential growth of the congestion window during slow start results in bursts of packets sent at a rate exceeding the bottleneck bandwidth on the path. On average, the transmission rate during slow start can reach twice the bottleneck bandwidth if the destination acknowledges every packet and if ACKs are not lost on the return path. In this case, ACKs return to the source at the rate of the bottleneck and every ACK triggers the transmission of two packets. Bursts during slow start are unavoidable since the current implementations of TCP don't provide any kind of spacing between the packets of a burst. Sometimes the size of a burst is limited but in general when the source decides to transmit many packets in response to an ACK, it sends these packets in a burst at the maximum rate allowed by its output interface whatever is the bottleneck bandwidth. Arriving at the input of the bottleneck link, packets sent in bursts must wait till their predecessors get served. A queue builds up in network buffers during slow start even if the window size is still smaller than the network capacity. The length of this queue increases with the increase in the slow start threshold hence with the increase in the BDP of the connection's path. If the buffers are small, an overflow may occur early during slow start before reaching the network capacity. Here the TCP source will consider that the network is congested, thus it reduces its estimation of the capacity and starts the congestion avoidance phase at a smaller window which results in a poorer performance.

This problem of losses during slow start has been studied in [5, 29]. The authors have found a condition on buffer size so as to avoid an overflow during slow start in the steady state of a TCP-Tahoe connection. They modeled the network with a single bottleneck node of bandwidth μ Packets/s and of buffer size B Packets (figure 1). T is taken as the two-way propagation delay. They calculated the maximum window size of a Tahoe connection as being $W_{max} = B + \mu T$. Slow start ends then at $W_{th} = W_{max}/2$. During slow start, half the window size is supposed to be queued in buffer B (the source is supposed to send bursts at twice μ). Then, B must be taken so large so that $B > W_{th}/2$. This gives a buffer size larger than $\mu T/3$. This analysis is not applicable to a Reno connection where a slow start phase is avoided most of the time. Moreover, with Reno and the other versions of TCP, if Fast Recovery fails, slow start will be called but with W_{th} smaller than $W_{max}/2$. Thus, the condition on B becomes less stringent.

For the slow start at the beginning of the connection, TCP assumes that a loss during this phase means that the window has reached the maximum capacity of the network. It uses these losses as a means to estimate the network capacity. However, if buffers are of small size compared to the BDP, a loss may occur during this phase even before filling the pipe. This will result in a wrong estimation of W_{th}, therefore in a lower performance. If we assume that the receiver acknowledges every packet as in the previous analysis, the buffer B must be larger than half BDP if we want TCP window to reach the BDP during the first slow start. In [25], an estimation of the slow start threshold has been proposed at the beginning of the connection in order to avoid losses. A threshold equals to BDP has been proposed. Again, this estimation may not avoid losses if buffers are small compared to BDP. With small buffers, losses during slow start are not a signal of network congestion but rather a signal of a transient congestion due to the bursty type of slow start traffic.

In congestion avoidance, packets are transmitted at approximately the bottleneck bandwidth. A queue starts to build up in the network when the window exceeds the BDP (μT in figure 1). Congestion occurs when the bottleneck buffer overflows. The window size at this point is normally equal to the pipe size. When the source detects this loss, it reduces its window by a factor of two and starts a new cycle, of course if the Fast Recovery algorithm works perfectly. To get always a throughput approximately equal to the bottleneck bandwidth, the new window after the reduction must be larger than the BDP. This requires a buffer size equal to the BDP. But a large buffer increases the RTT of the connection which may harm interactive applications. A tradeoff exists then between increasing buffers size to increase utlization and decreasing it to deliver quickly TCP packets. Note here that we are talking about Drop Tail buffers. Active buffer management routers as RED (Random Early Detection) [20] drop (or mark) packets when the average queue length exceeds a certain threshold. These routers are proposed to solve the bias of Drop Tail ones against bursty traffic. However, they don't let the window grow in congestion avoidance to the double of the BDP and hence the utilization drops to below the bottleneck bandwidth when congestion occurs. Here, the minimum threshold should be equal to the BDP to get good utlization. This contrasts one of the aims of RED, that of limiting the size of queues in network nodes in order to reduce the end-to-end delay. The Vegas version of TCP [9] solves this dependency of the utilization and the delay on the buffer size by not relying the congestion control only on losses. The increase in the RTT is used to maintain the window at a size equal to the BDP plus some packets (between 1 and 3) in the bottleneck buffer. The window is usually reduced by one packet rather than by half. Thus, the utilization is most of the time close to the bottleneck bandwidth and the end-to-end delay is kept close to the propagation delay.

4 Effect of the propagation delay

As any closed-loop congestion control protocol, the efficiency of TCP algorithms depends on the feedback delay. Long RTTs are now becoming prevalent in the Internet with the introduction of paths crossing satellite links. The one-way propagation delay across a GEO satellite is in the order of 250 ms. TCP reaction to congestion takes effect one RTT later. At long RTT, this reaction may become unnecessary or insufficient which may result in an inefficient utlization of network resources. However, other problems exist that are proper to TCP. As we said, most of the mechanisms of the protocol are based on the ACK clock. One important mechanism is the rate at which the congestion window increases. The source increases its window as a function of the number of ACKs received and doesn't take care of the connection RTT. This results in a control policy, thus in a performance, function of the propagation delay. Increasing the delay should not affect the performance of the connection during congestion avoidance, since it is supposed to run during this phase in steady state. This may affect the performance if we switch to congestion avoidance at a low slow start threshold. As we will see later, this may also affect the performance if the connection contends for the bottleneck bandwidth with other connections of shorter RTT. However, the increase in the delay impairs seriously the performance of slow start which is a transitory phase designed to fill quickly but smoothly the network capacity. Slow start requires then more time to achieve. Given that the network is underutilized during slow start due to the small size of the window, the performance of TCP degrades. This degradation in the performance is more critical for short transfers such as Web transactions. These transfers complete in general in the slow start phase and suffer therefore from a long delay and a low throughput.

Another problem caused by this dependency of the window increase on RTT is the *unfairness* in the allocation of the bottleneck bandwidth between connections of different RTTs. It is known that an additive-increase multiplicative-decrease congestion control strategy as the congestion avoidance algorithm of TCP leads to fairness if all the connections increase their throughput at the same rate [12]. The fairness here means a fair division of the bottleneck bandwidth among the different connections crossing it. This behavior is not seen with TCP. Many works have shown the bias of TCP against connections with long delay paths [24, 29]. Short delay connections increase their rates more quickly and grab most of the available resources. The average throughput of a connection has been shown [29] to vary as the inverse of T^α where T is the two-way propagation delay and α a factor between 1 and 2.

4.1 Proposed solutions to accelerate the window increase

Many propositions have been suggested to increase faster the window at the beginning of the connection. By increasing the burstiness of the slow start phase, these propositions try to reduce the number of RTTs required to reach the network capacity estimate. Again in the absence of any kind of bandwidth estimation and packets spacing, this problem seems to be unresolvable on an end-to-end basis. The tradeoff between burstiness and time during this phase is very clear. We must reach a certain window (W_{th}) but we are allowed to transmit packets only in bursts. Bursts overload the network and may cause losses which impairs the performance. Note here that this problem of slow window increase on long delay paths is exacerbated in the presence of ACK loss. This is because the increase is based on the number of ACKs received rather than on the number of packets that had reached the destination. For the same reason, the increase slows down if ACKs are delayed by the receiver. In the literature, the problem of slow start on long delay paths has been studied in the satellite context. However, it can exist on any path presenting a large delay. The solutions proposed can be divided into three categories. Some that change the window increase algorithm of TCP. Others that solve the problem at the application level and the others solve it at the network level.

4.1.1 TCP level solutions

The first proposition was to use a larger window than one segment at the beginning of slow start [2]. This large window has been proposed for the first slow start not those following a Timeout. After a Timeout, the network is supposed to be severely congested so that a transmission at a small initial window is required. An initial window of maximum 4 segments has been proposed. Another proposition, called Byte Counting, was to account for the number of Bytes covered by an ACK while increasing the window [1]. The congestion window is increased as a function of the number of data packets acknowledged rather than the number of ACKs received at the source. This decouples the window increase algorithm from the ACK clock. However, this may result in burstiness when many ACKs are lost. For this reason, a limit has been proposed on the number of packets that can be sent in a single burst (Limited Byte Counting).

The problem of slow start on a long delay path is then a result of the general problem of burstiness in TCP. The natural solution to such a problem is to use bandwidth estimation and packet spacing to control the rate of TCP packets. ACKs (or may be another type of acknowledgments) are used only for error control and for obtaining information on network conditions. But, rate control is very difficult to implement and it is not appropriate for short data transfers. One of the forces of TCP is its simplicity and its ability to cope with all kinds of applications, and this simplicity is due mainly to the ACK clock. With this clock, the ACKs return spaced to the source which causes a natural spacing between transmitted data packets without any overload at the source. This self-clocking of TCP must be preserved as a means to smooth the transmission of data packets. However, a rate control based solution can be envisaged to accelerate the window increase during slow start without overloading the network. By estimating the bottleneck bandwidth and spacing the outgoing packets, the source can transmit directly at a large window equal to BDP. Once the BDP is achieved, the source switches to congestion avoidance and the ACK clock takes the hand. This lets the source avoid a great part of slow start. An identical solution has been studied in [6] and a gain in performance has been noticed.

4.1.2 Application level solutions

These solutions try to solve the problem at the application level without changing TCP. The application manages the data to transfer and the establishment of TCP connections in such a way that increases the overall performance. The first solution, called XFTP [4], consists in establishing many simultaneous TCP connections for the transfer of a single file. This increases the growth rate of the resultant window which improves the utlization of the network resources during slow start. Also, the distribution of losses on many connections instead of one makes the recovery algorithm more efficient. However, this solution increases the aggressiveness of the transfer and hence the losses in the network. An adaptive mechanism has been proposed [4] to change the number of connections as a function of network congestion.

Another solution has been proposed to accelerate the transfer of WEB pages. Instead of using an independent TCP connection to fetch every object in a page, the client establishes a Persistent connection and asks the server to send all the objects on it (HTTP 1.1 [15]). WEB objects are most of the time of small size and using a connection per object results in a poor performance because most of, if not all, the transfer happens in the slow start phase. Also, the CPU is overloaded and a large memory is required to store the control blocks of the different connections. With a Persistent connection however, only the first objects suffer from the long slow start phase but once slow start is terminated and the connection passes to the steady state, the remaining objects are transfered at a high rate. The low throughput during slow start is hence compensated by the long time we stay in the congestion avoidance phase. A Persistent connection saves also the CPU time and the memory.

As a further solution, we find the caching. The TCP connection initiated by the client is established most of the time to a nearby cache which shortens the path, accelerates slow start and improves the performance of most of the transfers. Note here that caching is better supported by satellites due to their broadcast nature. Once a cache searches a file, all the caches in the satellite scope receive it.

4.1.3 Network level solutions

This approach consists in solving the problem inside the network rather than in hosts. It is worthy when a long delay link is located on the connection's path. In order to decrease the RTT of the connection, the long delay link is eliminated from the feedback loop by acknowledging packets by the router at the input of this link (A in figure 2). The packets are then transmitted on the long delay link (A-B) using an optimized transport protocol (ex. the STP protocol proposed in [23]). Another version of TCP can also be used [32]. Given that the bandwidth available between the two ends of the long delay link (between A and B) is known, the transport protocol used across it can be tuned to increase quickly its transmission rate without the need for a long slow start phase. Once arriving at the output router (B), another TCP connection is used to transmit the packets to the destination. In a satellite environment, the long delay link may lead directly to the destination so another TCP connection is not required. Because packets have already been acknowledged (by router A), any loss between the input of the link (A) and the destination must be locally retransmitted on behalf the source. Also, acknowledgments from the receiver must be silently discarded (on B) so as not to confuse the source. Given that the source is spoofed by the input router of the long delay link, this approach is called *TCP-Spoofing*.

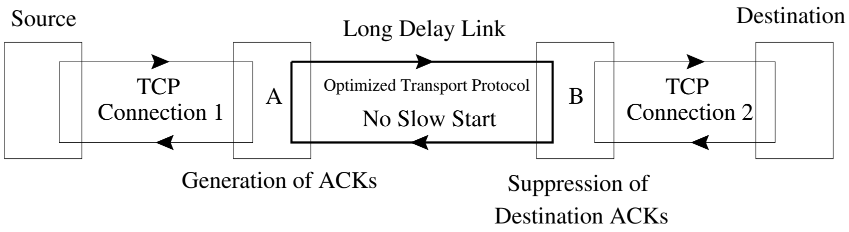

Figure 2: Spoofing : Elimination of the long delay link from the feedback

In fact, Spoofing splits the end-to-end connection into small ones. The main gain in performance comes from not using slow start (or from using a faster version) on the long delay link. Thus, the congestion window increases quickly (on the two sides of the long delay link) which improves the performance of the transfer. Another advantage of spoofing which is common to all the closed-loop flow control protocols, is a better reaction to network congestion. The feedback loop is rendered shorter and the throughput of the new small connections is adapted more quickly to network conditions. This results in a more efficient utlization of the available resources. An identical solution has been proposed in ATM networks for the improvement of ABR rate control over long delay paths [7]. An end-to-end ABR connection is divided into small ones. The switch at the input of a small connection behaves as a Virtual Source (node B in figure 2 as an example) and the one at its output behaves as a Virtual Destination. A virtual destination returns a feedback (RM cells) to the virtual source located before it which in turn controls the rate of ATM cells as function of the available bandwidth between them. As we will see later, splitting the connection into small ones results further in a better resilience against losses not caused by congestion. At small RTT, the window requires less time to return to its initial value before the loss. Moreover, one small connection reduces its window and the others are not affected.

Despite his advantages, spoofing still has a lot of drawbacks. First, it breaks the end-to-end semantics of TCP. A packet is acknowledged (by A) before reaching its destination. Also, it doesn't work when encryption is accomplished at the IP layers and it introduces a heavy overload on intermediate routers. Further, the transfer is vulnerable to path changes and symmetric paths are required to be able to discard the destination ACKs before they reach the source. IP Tunneling can be used to solve the later problem. Spoofing can be seen as a particular solution to some long delay links in the Internet. It is interesting when the long delay link is the only path leading to the destination. Long paths with no particular long delay link will not benefit from it. This solution is often proposed to accelerate the data transfer in networks using GEO satellite links to access the Internet.

4.2 Proposed solutions to improve the fairness

Two trends exist to improve the fairness of TCP. The first tries to solve the problem at the TCP level by accelerating the window growth for long delay connections, while the second keeps TCP unchanged and tries instead to solve the problem at the network level. Some intelligence is proposed to be added in network nodes in order to allocate the bottleneck bandwidth fairly among different connections.

As TCP-level solutions, we find first the proposition made in [17] and called *Constant Rate* algorithm. The author proposed to increase the window (W) in the congestion avoidance phase by a factor proportional to the inverse of the square of the connection RTT

$$W = W + c/(RTT^2).$$

The result is a constant increase rate (equal to c) of the connection throughput regardless of the length of the path it crosses. This improves the fairness in the allocation of the bottleneck bandwidth. The first problem in this proposition is the choice of the throughput increase rate (c). Also, accelerating the window increase while preserving the ACK clock results in large bursts for long delay connections. In [24], the authors show that with Drop Tail buffers, this increase in burstiness may degrade the fairness instead of improving it. However, RED routers are able to absorb this increase in burstiness which has been proven to increase vastly the fairness of TCP. To solve the problem, they propose instead an *increase-by-K* algorithm. The window is increased by K segments every RTT and this is only for long delay connections. The performance of long delay connections improves without affecting that of the others when K is given small values.

Now, in order to improve the fairness at the network level without changing TCP, flows of different connections must be isolated in network nodes. Given that congestion control in TCP is based on losses, isolation means that a congested node must distribute losses between the different TCP connections in such a way that they get the same throughput. The best isolation is achieved by accomplishing a per-connection queuing with a Round Robin service discipline (Fair Queuing). Such solution ensures fair share of bandwidth at short time scales but it is very hard to implement given the large number of TCP connections that cross an Internet router. An easier solution for implementation is to use the same buffer for all connections but to manage its space intelligently. Packets of different connections are then served with a FIFO strategy. Many buffer management policies have been proposed. Some of these policies, as ERD (Early Random Drop) [22] and RED, choose to drop incoming packets with a certain probability when the queue length or its average exceed a certain threshold. This is in order to distribute losses on the different connections proportionally to their throughput. These policies improve the fairness while not requiring any per-connection state. However, it has been shown in [31] that dropping packets proportionally to the bandwidth share of the corresponding flow doesn't lead always to fairness in bandwidth especially if the bottleneck is crossed by a non-adaptive traffic. A better fairness requires a control of the buffer occupancy of every connection. Aggressive connections must be prevented from monopolizing buffer space. Further, a flow using less than its share of the bandwidth must be isolated from those sending at high rates. We find here the other set of policies [21] (Flow RED, Fair Buffer Allocation, Virtual Queueing, etc.) that try to achieve fairness by sharing the buffer space fairly between active connections. These policies ensure that each connection has at least a minimum number of packets in the queue which isolates completely connections sending at small rates from aggressive ones. A long delay connection is now sure to find some places in the buffer which permits it to proceed without being shut down by short delay connections. The result is an improvement in fairness but at the same time an increase in buffer management overhead.

Solving the fairness problem at TCP level has the advantage of keeping the nodes simple. But, such solution requires first that the end-systems act correctly. Second, it results in burstiness and therefore a possible performance deterioration. Long delay connections are obliged to send large bursts in order to follow the fast window increase of short ones. With the prevalence of non-TCP-friendly traffics, an end-to-end solution is no longer sufficient to ensure fairness. Some mechanisms in network nodes are required to absorb bursts of long TCP connections and to protect conservative TCP flows from aggressive ones.

Network mechanisms are also required to ensure fairness at a level above TCP, say at the user or at the application level. A user (ex. running XFTP [4]) may cheat and establish many TCP connections in order to increase its share of the bandwidth. If the network manages to divide fairly its resources between active TCP connections, this user will be favored. The total flow of packets generated by these connections must be considered by the network as a single flow. This requires an aggregation in flows of TCP packets crossing a a network node. The buffer space is then managed to divide the available resources fairly between the active flows. The level of aggregation determines the level of fairness we want. The source and destination IP-addresses, the transport protocol used, the source and destination port numbers, are the most useful information carried by IP-packets and helping the network in defining the flows.

5 Effect of non-congestion losses

TCP assumes that losses in the network are only caused by congestion. The window is reduced by one half whatever is the source of the loss. This reaction to losses results in severe throughput deterioration if packets are lost on the path for other reasons than congestion. Non-congestion losses, called also *random losses* in some literature [29, 30], involve all kinds of losses that occur independently of TCP congestion control algorithms. Their negative effect on TCP performance increases with the Bandwidth-Delay Product of the connection's path [29]. A random loss that occurs while TCP is transmitting at a large window reduces considerably the performance. After this loss, the source requires a long time to return to its initial rate.

Random losses are mostly caused by transmission errors. A packet may be corrupted while crossing a bad quality link. A corrupted packet is then discarded at the link level, IP level or TCP level without being handed to the application. Bad quality links involve mainly wireless links where many phenomena such as interference, absorption, obstacles and others [14], may reduce the Signal to Noise ratio at the receiver and cause therefore a misinterpretation of the arriving signal. Wireless links are now widely seen in the Internet. We find them in satellite and in terrestrial mobile networks. Mobility of the ends of a wireless link, as in a LEO-satellite constellations, induces other phenomena that come to increase the loss probability of a packet. Fading, shadowing, handover, etc. [36], may put the link in a bad state for a non negligible time which results in bursts of losses.

In [29], losses due to transient congestion in network nodes are also considered as random losses. A transient congestion is defined as being any congestion that lasts for a small time compared to the RTT of the connection. A TCP reaction to this short congestion has no meaning. It can be caused by a bursty source (ex. a non controlled real time traffic) sharing the bottleneck with a TCP connection. Losses inside an ATM backbone providing an ABR service to carry TCP traffic can be considered also as a transient congestion [21].

The solutions proposed to this problem can be divided into two main categories. The first consists in hiding the lossy parts of the Internet so that only congestion losses are detected by the source. This solution requires some work in the network but has the advantage of not requiring any change to TCP. The second type of solutions consists in enhancing TCP with some mechanisms to help it to distinguish between different types of losses. It can be considered as an attempt to decouple congestion control of TCP from error control.

5.1 Hiding of random losses

This set of solutions consists in recovering non-congestion losses locally without the intervention of the source. The source sees then a clean path and it continues to increase its throughput until a real congestion occurs. Local recovery requires the identification of the lossy part of the Internet. Only random losses must be hided. Losses due to congestion must still be detected by the source otherwise the network will be overloaded.

A local recovery can be accomplished at the link level or at TCP level. From IP point of view, a link is any transmission medium located between two adjacent routers. Improving the quality of links in the Internet results in only congestion losses reaching the source. This leads us to the typical network TCP is tuned to: A network formed of the interconnection of good quality transmission media. If a link-level solution is not possible, a recovery at the TCP level can be used instead. An intermediate node on the path monitors TCP packets, detects losses and retransmits them on behalf the source. This retransmission must cover only the lossy part of the path where no congestion can appear. Another connection must be established to control the congestion between this lossy part and the destination, of course if the destination is not directly connected to the lossy part.

5.1.1 Link-level solutions

Two well known mechanisms exist for the improvement of a link quality: Retransmissions (ARQ) and Forward Error Correction (FEC). ARQ is efficient when losses are not frequent and when the propagation delay is not important. An extra bandwidth is consumed only when a packet is retransmitted. However, link-level retransmission may interfere with TCP mechanisms [11]. If the link layer doesn't provide an in-sequence delivery of packets to the IP layer, TCP packets following the loss keep to arrive at the destination and trigger the transmission of Duplicate ACKs. These Duplicate ACKs may reach the source while the link layer is trying to retransmit the packet. This causes an unnecessary window reduction which we try to avoid. The proposed solution to this problem was to use a link-level protocol aware of TCP mechanisms (figure 3). The link layer suppresses the Duplicate ACKs so that they don't reach the source (at router R). If the link layer fails to retransmit the packet,

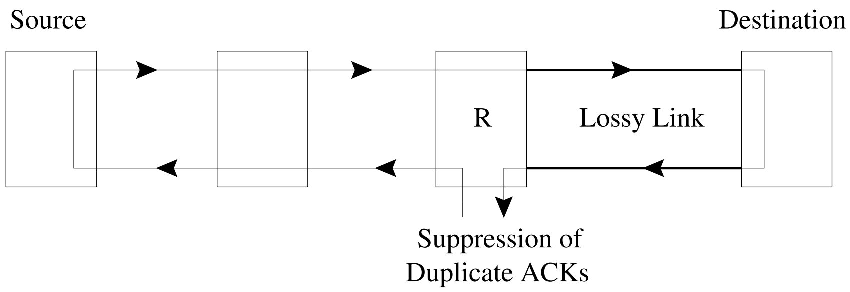

Figure 3: A TCP-aware Link Layer

the source will then timeout and retransmit the packet itself. Note here that this later solution (i.e. a TCP-aware link layer) is applicable only when the lossy link is the last part of the path (i.e. the link used by the destination to access the Internet). The suppression of Duplicate ACKs by the link layer means implicitly that the loss is caused by a transmission error not by congestion. If the lossy link is followed by other routers where congestion may occur, congestion losses will be then hided which must be avoided.

FEC at the link level consists in sending, in addition to the useful data, some redundant information in order to rebuild the corrupted part of the packet. The drawback of this technique is that the redundant information is not used when the link is in the good state. Thus, it presents a certain waste of the available bandwidth. Also, the computation of this redundant information requires extra CPU processing time, memory and blocking delay. However, the advantages of FEC are in general worth the cost. First, corrupted packets are corrected directly without retransmission which is important for long delay lossy links as the satellites ones. Also, FEC doesn't interfere with TCP mechanisms. For these reasons, this technique has been recommended in a satellite environment [3]. Convolutional coding, Viterbi decoding together with interleaving techniques and Reed-Solomon encoding, are widely used to render satellite links as clean as terrestrial ones.

5.1.2 TCP-level solutions

These solutions try to improve the link quality by retransmitting packets at the TCP level instead of the link level. A link layer aware of TCP breaks the principle of layering. According to these solutions, a TCP agent in the router at the input of the lossy link keeps a copy of every data packet. It discards this copy when it sees the ACK of the packet and it retransmits the packet on behalf the source when it detects a loss. This retransmission is accompanied by a discard of Duplicate ACKs. This technique has been proposed [8, 11] for terrestrial wireless networks where the delay is not very important to require the use of FEC. The TCP agent is placed in the base station at the entry of the wireless network. Two possible implementations of this agent have been proposed.

The first implementation (ex. Indirect TCP [8]) consists in terminating (or splitting) the originating TCP connection at the entry of the lossy link. The agent then acknowledges the packets and takes care of handing them to the destination. Another TCP connection well tuned to a lossy environment (ex. TCP-SACK) can be established across the lossy part of the network. A different transport protocol can also be used across this lossy part. This solution is similar to Spoofing described in section 3.1.3. Spoofing aims to eliminate the long delay link from the path of the connection in order to solve the problem of the slow start phase. This solution however aims to eliminate the lossy link from the loop. As with Spoofing, it breaks the end-to-end semantics of the Internet. Also, it causes difficulties during handover since a large state must be transfered between base stations.

The second implementation, called *Snoop protocol* [11], respects the end-to-end semantics. The intermediate agent doesn't terminate the TCP connection. It just keeps copies of data packets and it doesn't generate any artificial ACK. New ACKs sent by the destination are forwarded to the source, however Duplicate ACKs are stopped. A packet is retransmitted when three Duplicate ACKs are received or when a local Timeout expires. This local Timeout is set of course to a value less than that of the source. The drawback is that an interference may happen between the source and the agent mechanisms as in the link-level case. In fact, the snoop protocol is no other than a link-level recovery implemented at the TCP level. Again, because it hides all losses, it requires that no congestion occurs between the snoop agent and the destination.

5.2 End-to-end solutions

Link-level solutions are clearly beneficial in improving the performance of TCP. Hiding random losses at the TCP level is also beneficial, but to avoid the problem of not reacting to congestion losses that occur after the lossy link, this lossy link must be the last part of the path as in the case of a terrestrial mobile network. Of course, this is true if we want to preserve the end-to-end semantics of TCP otherwise the connection can be split at the output of the lossy link and random losses can then be hided. But, random losses can appear anywhere in the network not only on the last link. This requires in addition the definition of some end-to-end solutions.

Two kinds of solutions have been proposed in the literature. The first one consists in informing the source explicitly whether a packet is lost randomly or due to congestion. An *Explicit Loss Notification* (ELN) signal has been proposed [39] to infer the source explicitly of the occurrence of a non-congestion loss. The source reacts then by retransmitting the packet without reducing its window. An identical signal has been proposed to halt the congestion control at the source when a disconnection appears due to handover in cellular networks. The difficulty with such solution is that a packet corrupted at the link level is discarded before reaching TCP and then it is difficult to get this information.

The second type of solutions has been proposed however to improve the control of congestion provided by TCP rather than the recovery from random losses. We mention it here because it consists a step towards a solution to the problem of random losses on an end-to-end-basis. These proposed solutions aim to decouple congestion detection from losses. With some additional mechanisms in the network or at the source, the congestion is detected and the throughput is reduced before the overflow of network buffers. First, this reduces the size of queues in the nodes and thus the end-to-end delay. Further, the avoidance of losses avoids retransmissions and then gives better service for interactive applications. As solutions, we find the Vegas version of TCP [9] and the Explicit Congestion Notification (ECN) proposed in [18]. In Vegas, the RTT of the connection and the window size are used to compute the number of packets in network buffers. The window is decreased when this number exceeds a certain threshold. With ECN, an explicit signal is used by the routers to indicate congestion to TCP sources rather than dropping packets.

If all the sources, receivers and routers are compliant (according to Vegas or ECN), the reduction of the throughput before the overflow of network buffers will reduce drastically the congestion losses. The remaining losses can be then considered as mostly due to random losses. Moreover, if some averaging on the queue length is accomplished while detecting the congestion, transient overflow of buffers will not be detected by the source as a congestion event and therefore the losses that result can be also considered as random. Given that random losses requires only retransmission without window reduction, the disappearance of congestion losses may lead to the definition at the source of a new congestion control algorithm that reacts less seriously to losses. The congestion window may be reduced slightly or even not reduced upon the occurrence of a loss. However, it must still be reduced by a factor of two in response to explicit congestion signals. This may resolve the problem of random losses without affecting TCP performance.

This ideal behavior doesn't exist in todays networks. In the absence of any feedback from the network as with Vegas, the congestion detection mechanism at the source may fail and here the congestion losses are unavoidable. If the source bases its congestion control on an explicit information from the network as with ECN, some non-compliant routers will not provide the source with the required information and drop packets upon congestion instead. A reduction of the window upon loss detection is necessary in this case. For these reasons, these approaches still consider losses as a congestion signal and reduce their window accordingly. However, with some refinements to avoid failures in congestion detection, they may consist a solution to the problem. Such solution requires further study to understand its impact on the network and the other users.

6 Effect of Bandwidth Asymmetry

One of the basic assumptions of TCP is that ACKs traverse the reverse channel without any disturbance. They are supposed to reach the source as they left the destination. If this is the case, the source can rely on this clock to guess what is happening on the forward channel. This should result in an efficient recovery phase. Also, this should yield a fast increase in the window and a smooth transmission of packets. However, in todays networks, many paths are presenting some bandwidth asymmetry between the two directions. In order to provide a high speed Internet access, the capacity of the forward direction is increased using satellite down-links or cables. However, a low speed channel, such as a dial-up modem line, is still used to carry ACKs back to the source(figure 4). As an example of such networks we see the Direct Broadcast Satellite networks and Asymmetric Digital Subscriber Loop (ADSL) networks. Even if ACKs are of small size with respect to data packets, the reverse channel might be unable to carry the high rate of ACKs if the forward link has to be fully utilized. This results in a congestion and losses on the ACK channel (at point A). If the reverse channel is used also for data, the problem of asymmetry will be exacerbated. Data packets of large size contend for bandwidth with small ACK packets which result in more problems such as ACK compression. Also, the contention between many connections for the small bandwidth available on the reverse path may result in unfairness. We can consider these problems first as a result of the reliance of TCP algorithms on the ACK clock. Second, as a result of

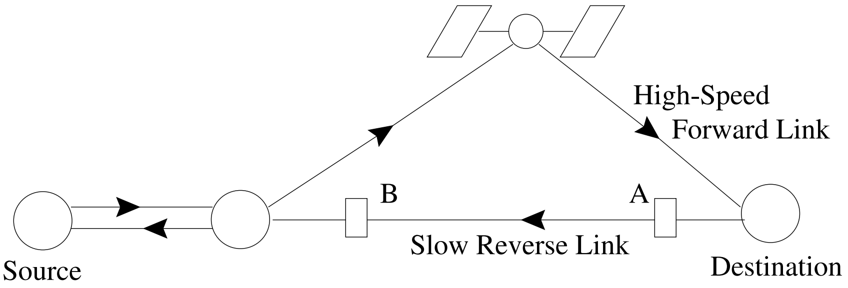

Figure 4: An asymmetric path

the high rate at which ACKs are generated (an ACK for every two packets if ACKs are delayed).

Before studying these points and the proposed solutions, let us define the asymmetry factor K. In [30], K is defined as the ratio of the bottleneck rate on the forward path in terms of data packets per unit of time to the bottleneck rate on the reverse path in terms of ACK packets per the same unit of time. This factor represents the overload on the reverse channel. Due to the small size of ACKs (usually 40 Bytes) with respect to data packets, the reverse channel needs not to be of the same raw rate as the forward one. A path is presenting some bandwidth asymmetry if

$$K = \frac{\text{Forward bandwidth}}{\text{Reverse bandwidth}} \times \frac{\text{ACK packet size}}{\text{Data packet size}} > 1.$$

First, we study the case of a single connection using a slow reverse channel for its ACKs. The impact on performance of ACK congestion and losses is mentioned. Second, we study the problems that appear when multiple connections contend for the reverse channel. The interaction between forward connections as well as between forward and reverse connections is highlighted.

6.1 Case of a single connection

The high rate of ACKs causes a queue building in the buffer of the reverse bottleneck (A in figure 4). This increases the RTT of the connection and causes losses of ACKs. The increase in the RTT results in a throughput deterioration given that the source is authorized to transmit only one window of packets per RTT. Also, it slows the growth of the throughput which impairs further the performance when operating on a long delay path or in a lossy environment.

The loss of ACKs disturbs one of the main functionalities of the ACK clock, that of smoothing the transmission rate. The window slides quickly upon the receipt of an ACK covering multiple lost ACKs and a burst of packets is sent. Transmitting in bursts may overwhelm the buffers of the forward direction which results in a performance degradation [10, 30]. If the receiver acknowledges every data packet, one ACK over K is correctly received. This results in the transmission of bursts of size K at the source. The connection is supposed to be in the steady state. Thus, the forward buffers must be at least of the size of the asymmetry factor [30]. Also, the loss of ACKs slows down the growth of the congestion window which results in a poor performance in case of long delay paths and lossy links. It has been shown that, in case of random losses on the forward direction, the average throughput of a TCP connection varies as the inverse of root square of K [30]. Note here that these problems in the operation of TCP can be observed also if ACKs are lost for other reasons than bandwidth asymmetry as in the case of a bad quality ACK packets channel.

The proposed solutions to this problem can be divided into two types according to which side of the reverse path they are applied. Receiver side solutions try to solve the problem by reducing the congestion on the return path. The first solution, called SLIP Header Compression [27], compresses the headers of TCP packets on the slow channel (A-B) to increase its capacity in terms of ACK packets per unit of time, thus to decrease the factor K. It profits from the fact that most of the information in a TCP header don't change along the connection lifetime.

The other solutions propose to reduce the rate of ACKs [10]. Less ACKs are sent to the source so as to avoid congestion. This alleviates the problem of the increase in RTT. Also, as we will show later, it solves the problem of fairness when many connections contend for the reverse channel. However, the loss of ACKs still exist and some solutions at the sender are required.

The first proposition to alleviate the congestion is to delay ACKs at the destination [10]. An ACK is sent every d packets and an adaptive mechanism has been proposed to change d as a function of the congestion on the ACK path. Another proposition [10] keeps the destination unchanged and solves the problem in the buffer of the reverse bottleneck (in A). It profits from the fact that the information carried by ACKs is cumulative. When an ACK arrives at the entry of the slow channel (A), the buffer is scanned to see if another ACK of the same connection is buffered. The old ACK is then substituted by the new one. The queue in this buffer is then maintained at a small length which reduces the latency. Again, the problem of ACK loss is not resolved. We can understand this solution as a filtering of the ACK stream to match its rate to the rate of the reverse channel. ACKs will be naturally filtered by a Drop Tail buffer but sometime later when it gets full. The gain in performance when the ACK rate is reduced comes from the fact that the natural filtering is anticipated.

A solution at the source to the loss of ACKs requires a decoupling between TCP mechanisms and the ACK clock. However, the problem of burstiness cannot be completely resolved in the absence of any packet spacing. It can be alleviated by imposing a limit on the size of bursts. But in this case, where ACKs are lost systematically, limiting the size of bursts will limit the throughput of the connection. Another solution has been proposed [10] to reconstruct the ACK clock at the output of the reverse slow channel (at B). When an ACK arrives at B, all the missing ACKs between the sequence number it carries and that of the last ACK received at B are generated. The generated ACKs are spaced by a time interval derived from the average rate at which ACKs get out of the reverse slow channel. This reconstruction may consist a solution to this particular problem. However, the general problem of TCP burstiness upon ACK loss still exists. Some measures should be taken at the source to avoid the overload of network buffers with long bursts.

6.2 Case of multiple connections

Consider first the case when multiple forward connections contend for the reverse channel (A-B in figure 4). In the absence of any ACK filtering, the firstly starting connection will overflow the reverse buffer (the buffer in A). When another connection arrives, it is very probable, due to the congestion in this buffer, that the ACKs of the first packets it transmits get lost. Here, the new connection will suffer from a slow increase in its throughput and a possible Timeout. This results in a blockage of any new connection until the dominant one reduces its throughput [30]. The main cause for this is that a Drop Tail buffer treat arriving packets with the same manner. ACKs of a new connection, or those of a connection transmitting at a small window, must be given certain priority to let their window grow to a high value. Once grown, the effect of ACK losses becomes less important. The reverse buffer must be then shared fairly between the ACKs of different connections. Active buffer management policies as FRED [31] can be used. Also, any mechanism that reduces the generation rate of ACKs as a function of the congestion in the reverse buffer improves the fairness.

Now, a forward connection can also contend for the reverse buffer with a connection sending data in the reverse direction. The transmission time of a data packet on the reverse link is much greater than that of ACKs. This will decrease considerably the bandwidth available to ACKs which increases the asymmetry factor K. ACKs are then delayed after data packets which increases the RTT and the ACK loss rate.

If the forward connection is now sending at high rate so that it overflows the reverse buffer with its ACKs, the data packets of the reverse connection will be frequently lost. The reverse connection reduces then its throughput whereas the forward one continues to increase it. Again, this will result in a blockage of the reverse connection until the forward one ends [30]. This problem requires an intelligent drop policy that prohibits a connection from occupying all the buffer. Data packets find then a place in the reverse buffer even if the ACKs are transmitted at high rate. Now, to prohibit data packets from blocking for long time the reverse path, an intelligent scheduling policy can be envisaged at the input of the slow link (at router A). The aim is to serve packets according to their size. Many ACKs can then be served between data packets.

Another phenomenon noticed when many connections exist in the two directions is the *compression of ACKs* [41]. In this case, the reverse channel is taken not too slow so that in the absence of reverse data packets, ACKs return to the source as fast as they are transmitted. If these ACKs arrive to a buffer after a data packet, they will be queued until the data packet is transmitted. The ACKs reach then the source in batches which results in burstiness and possibly in losses. The solution here is to give ACKs priority over data packets. The transmission of a data packet is then interrupted to let the ACKs pass. Given the difficulty to implement such solution, some measures must be taken at the source to alleviate the problem of burstiness.

7 Conclusions

In this work, we have summarized many problems that affect TCP performance in the Internet of today. The impact of the new characteristics of Internet paths has been presented. The study has been accomplished independently of the network type. Some of the solutions proposed in the literature have been mentioned. Many of these solutions were proposed in a particular context such as a satellite network or a wireless network. We showed the utility of these solutions on a general path having a certain characteristic. The analysis provides us with an understanding of the mechanisms TCP used for the control of congestion, of flow of packets and of transmission errors.

The reliance on the ACK clock for the transmission of packets is the main cause for the simplicity of flow control in TCP. Packets are sent in a burst when an ACK is received without any kind of rate control. No adaptive mechanisms are used to space the packets of a burst. Sometimes, a limit on the maximum size of bursts is placed. A limit has been proposed for the burst that may be sent at the end of

Fast Recovery [16]. The number of packets that we are allowed to send when an ACK is received and which are in excess of the limit placed on the burst size, are transmitted upon the receipt of the subsequent ACKs. But this supposes implicitly that the subsequent ACKs are not lost, otherwise the number of packets exceeding the limit will increase and a transmission at a full window will not be possible.

The smoothness of the flow of data packets is then strongly dependent on the disturbance of the ACK clock. Congestion control in TCP has also based its algorithms on this ACK clock. The first algorithm is that used to increase the window. The window is increased upon the receipt of an ACK. Further, the amount of the increase is a function of the number of ACKs received. The second algorithm is Fast Retransmit. A packet is considered lost if three Duplicate ACKs are received. Some versions of Fast Recovery depend on this clock by using the number of Duplicate ACKs to estimate the number of packets in the pipe. Also, in the absence of any explicit information from the network, TCP considers the loss of a packet as a congestion signal. Other algorithms of TCP depend on this clock as the estimation of the RTT of the connection. All these algorithms suppose that the ACK clock is not disturbed and represent perfectly what is happening on the forward direction. They are sensitive to loss of ACKs not to the change in the spacing between ACKs. This later change affects mainly the flow control.

Some propositions have been presented to decouple the congestion control algorithms of TCP from the loss of ACKs. FACK [33] with the SACK information decouples the Fast Recovery algorithm. Byte Counting [1] decouples the amount of the increase in the window. In the absence of rate control, any solution to decouple an algorithm from the loss of ACKs results in an increase in burstiness. Such solution supposes implicitly that all the ACKs are correctly received but in batches around the really received ones. Now, even if TCP algorithms manage to cope with ACK losses, other problems in TCP congestion control still exist. TCP increases its window when an ACK is received. The amount of the increase in the window is limited in order to limit the aggressiveness of the protocol. This leads to a poor performance of the protocol during the slow start phase especially on large Bandwidth-Delay Product paths. Also, this results in an unfairness of the protocol against long connections. Again, the solution to these problems requires an acceleration of the window growth which leads us to more burstiness. During slow start, a faster growth of the window to reach the network capacity is needed. Long delay connections have to be more aggressive in order to compensate their long RTT.

We think that the solution of TCP burstiness is one of the important issues in the improvement of the performance of the protocol. A typical solution at the

TCP level is the estimation of the bottleneck bandwidth and the spacing of packets. However, this solution is difficult to implement and it is not clear if its worth the effort especially with the introduction in the network of RED routers [20]. One of the goals of these routers is to keep some empty places in their buffers in order to absorb bursts of packets. This may be of interest for the congestion avoidance phase where bursts are not frequent. However, for slow start, the traffic is bursty for long time. Any proposition to reduce the time of this phase will exacerbate the problem of burstiness. A particular solution to this phase based on rate control may help TCP to fill quickly the pipe without overloading the network buffers. An estimation of the bottleneck bandwidth and the RTT lets the source pass directly to a window equal to the Bandwidth-Delay Product. Then, it switches to congestion avoidance where the rate control can be disabled to reduce the overhead of the implementation. We think that a bypassing of slow start, or at least a part of this phase, will improve the performance of TCP over long delay links. Further research is required to show the feasibility of such solution.

The other problem to solve is that of congestion detection and random losses control. We think that an improvement in TCP performance requires a decoupling between these two mechanisms. We gain in performance if random losses are recovered without the invocation of TCP congestion control algorithm. Link-level error recovery, as ARQ and FEC, forms an efficient means for hiding random losses from TCP. However, random losses still exist as in the case of the failure of the FEC technique. Congestion can be detected by the increase of the RTT as done by TCP Vegas. This mechanism in congestion detection is not enough to decide that a loss is random and that it can be retransmitted without window reduction. For this reason, TCP Vegas still reduces its window by a factor of two upon loss detection. We believe that an explicit information from the network is required to help the source distinguish between the types of losses. Random losses can be caused by many factors and can appear at many levels. In general, it is difficult to detect these losses in network nodes or at the receiver. Also, it is difficult to inform the source of their occurrence. Congestion, in contrast, can be previewed by routers using RED routers as an example. Thus, instead of dropping packets, the router can inform the source explicitly of the occurrence of a congestion in the network as with ECN [18]. Here, the source reduces its window upon the reception of ECN signals. If all the hosts and routers in the network are ECN compliant, a loss can be considered as mostly due to random loss. The source may retransmit it directly without (or with a slight) window reduction. In addition to transmission errors, this may form a solution for packets lost due to transient congestion in network nodes. Indeed, a transient congestion will not increase the average queue length in RED routers and thus ECN messages are not sent. The difficulty with such solution is that it requires a deployment of RED ECN-capable routers in all the network. Also, it requires a change to TCP senders. One of the changes is to keep unchanged the slow start threshold and the window size at the beginning of a Fast Recovery phase. The effect of such reaction to losses on the efficiency of network resources utlization must be deeply investigated. The effect on the other versions of TCP must be also evaluated. The current versions of TCP may get a low performance when contending for network resources with another version that doesn't reduce its throughput upon loss detection.

We believe that the two main problems of TCP are burstiness and the coupling between congestion detection and error control. Solutions to these problems may require a change to the protocol and to the network but we think they might be necessary to make TCP cope with the heterogeneity of todays transmission media on an *end-to-end basis*.

References

[1] M. Allman, "On the Generation and Use of TCP Acknowledgments", Computer Communication Review, Vol.28, No.5, October 1998.

[2] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", RFC 2414, September 1998.

[3] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", RFC 2488, January 1999.

[4] M. Allman, H. Kruse, and S. Ostermann, "An Application-Level Solution to TCP's Satellite Inefficiencies", Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS), Rye, New York, November 1996.

[5] E. Altman, J. Bolot, P. Nain, D. Elouadghiri- M. Erramdani, P. Brown, and D. Collange, Performance Modeling of TCP/IP in a Wide-Area Network, INRIA-France, Research Report N=3142 (1997) (available in http://www.inria.fr:80/RRRT/RR-3142.html). A shorter version in 34th IEEE Conference on Decision and Control, pp. 368-373, New Orleans, Louisiana, December 1995.

[6] M. Aron and P. Druschel, "TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control", Rice Computer Science, Technical Report TR98-318.

[7] "The ATM Forum Traffic Management Specification Version 4.0", ATM Forum Traffic Management AF-TM-0056.000, April 1996.

[8] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", in Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS), May 1995.

[9] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communications, Vol.13, No.8, October 1995.

[10] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance", in ACM MobiCom, September 1997.

[11] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz, "A comparison of Mechanisims for Improving TCP Performance over Wireless Links", in ACM SIGCOMM, August 1996.

[12] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks", Journal of Computer Networks and ISDN, Vol.17, No.1, pp. 1-14, June 1989.

[13] Robert Durst, Gregory Miller, and Eric Travis, "TCP Extensions for Space Communications", in ACM MobiComm, November 1996.

[14] B. R. Elbert, "The Satellite Communication Applications Handbook", Artech House, Boston, London, 1997.

[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2068, January 1997.

[16] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", Computer Communication Review, Vol.26, No.3, July 1996.

[17] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic", Computer Communication Review, Vol.21, No.5, October 1991.

[18] S. Floyd, "TCP and Explicit Congestion Notification", Computer Communication Review, Vol.24, No.5, October 1994.

[19] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 2582, April 1999.

[20] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, Vol.1, No.4, p. 397-413, August 1993.

[21] R. Goyal, R. Jain, S. Kota, M. Goyal, S. Fahmy, and B. Vandalore, "Traffic Management for TCP/IP over Satellite-ATM Networks", IEEE Communications Magazine, March 1999.

[22] E. Hashem, "Analysis of random drop for gateway congestion control", Report LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA, 1989.

[23] T. Henderson and R.H. Katz, "Transport Protocols for Internet-Compatible Satellite Networks", IEEE Journal on Selected Areas in Communications, Vol.17, No.2, February 1999.

[24] T. Henderson, E. Sahoria, S. McCanne, and R. H. Katz, "Improving Fairness of TCP Congestion Avoidance", IEEE Globecomm Conference, Sydney, Australia, November 1998.

[25] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", SIGCOMM Symposium on Communications Architectures and Protocols, August 1996.

[26] V. Jacobson, "Congestion avoidance and control", in ACM SIGCOMM, Stanford, CA, USA, August 1988.

[27] V. Jacobson, "Compressing TCP/IP Headers for Low-speed Serial Links", RFC 1144, February 1990.

[28] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[29] 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, pp. 336-350, June 1997.

[30] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance", INFOCOM'97.

[31] D. Lin and R. Morris, "Dynamics of Random Early Detection", In ACM SIGCOMM, Cannes, France, September 1997.

[32] I. Minei and R. Cohen, "High-Speed Internet Access Through Unidirectional Geostationary Satellite Channels", IEEE Journal on Selected Areas in Communications, Vol.17, No.2, February 1999.

[33] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", in ACM SIGCOMM, August 1996.

[34] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[35] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: a Simple Model and its Empirical Validation", UMASS CMPSCI Tech Report TR98-008, February 1998.

[36] J. D. Parsons, "The Mobile Radio Propagation Channel", Pentech Press, London, 1992.

[37] V. Paxson, "End-to-End Internet Packet Dynamics", In ACM SIGCOMM, Cannes, France, September 1997.

[38] J. Postel, "Transmission Control Protocol", RFC 793, September 1981.

[39] N. Samaraweera and G. Fairhurst, "Reinforcement of TCP/IP Error Recovery for Wireless Communications", Computer Communication Review, Vol. 28, No.2, pp30-38, April 1998.

[40] W. Stevens, "TCP Slow-Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.

[41] L. Zhang, S. Shenker, and D.D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", In ACM SIGCOMM, September 1991.

[42] Y. Zhang, D. DeLucia, B. Ryu, and S. Dao, "Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential", INET'97, Kuala Lumpur, Malaysia, June 1997.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399