On TCP Performance in a Heterogeneous Network: A Survey

Chadi Barakat, Eitan Altman and Walid Dabbous
INRIA
2004, route des Lucioles
06902 Sophia-Antipolis Cedex, France
E-mail: {cbarakat,altman,dabbous}@sophia.inria.fr

Abstract

Transmission media carrying Internet traffic present a wide range of characteristics some of which as transmission errors, long end-to-end delay, bandwidth asymmetry, may cause a degradation in TCP performance. Many works have studied the performance of TCP over these media. Most of these works focus on a particular network type. In this work, we study TCP performance independently of the type of the network by considering the different possible characteristics of the connection path. We present the problems and the different proposed solutions. This study permits us to understand the limitations of the actual solutions and the required modifications to let TCP cope with a heterogeneous Internet on an end-to-end basis.

Keywords
TCP, Flow and Congestion Control, Error control, Performance.

1 Introduction

TCP (Transmission Control Protocol) provides a reliable connection-oriented in-order service to many of todays Internet applications. Given the simple best effort service provided by IP, TCP must cope with the different transmission media crossed by Internet traffic. This mission of TCP is becoming difficult with the increase in the heterogeneity of the Internet. High speed links (Fiber optic), long and variable delay paths (Satellite links), lossy links (Wireless networks), asymmetric paths (Hybrid satellite networks) and others, are becoming widely embedded in the Internet. Many works have studied by experimentations [5, 6, 10, 12], analytical modeling [3, 8, 13, 14] and simulations [7, 8, 9, 11], the performance of TCP in this new environment. Most of these works have focused on a particular environment: satellite networks, mobile networks, etc. They have shown some problems in the operation of TCP. 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. A large number of solutions have been proposed. Some solutions suggest modifications to TCP to help it to cope with these new paths. Other solutions keep the protocol unchanged and hide the problem from TCP. Typically, modifications which vary according to the problem to hide are proposed inside the network.

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 characteristics of a path crossed by TCP traffic, focusing on bandwidth-delay product (BDP), round-trip time (RTT), non-congestion losses and bandwidth asymmetry. After overviewing TCP in the next section, we study the impact of each of these characteristics on TCP and we present the proposed solutions as well as our comments in four corresponding sections.

2 Overview of TCP

TCP is a reliable window-based ACK-clocked flow control protocol. It uses an additive-increase multiplicative-decrease strategy for changing its window as a function of network conditions [1, 12]. Starting from one packet, or a larger value as we will see later, the window is increased exponentially by one packet for every non-duplicate ACK until the source estimate of the network capacity is reached. By capacity of the network, sometimes called the pipe size, we mean the maximum number of packets that can be fit on the path. This is the Slow Start (SS) phase and the capacity estimate is called SS threshold (`ssthresh`). SS aims to alleviate the burstiness of TCP while filling quickly the pipe. Once `ssthresh` is reached, the source switches to a slower increase in the window by one packet for every window's worth of ACKs. This phase, called Congestion Avoidance (CA), aims to probe slowly the network for any extra bandwidth. The window increase is interrupted when a loss is detected. Two mechanisms are available for the detection of losses: the expiration of a retransmission timer (Timeout) or the receipt of three duplicate ACKs (Fast Retransmit (FRXT)). The source supposes that the network is in congestion and sets its estimate of the capacity to half the current window. Tahoe [7, 12], the first version of TCP that implements congestion control, sets the window at this point to one packet and uses SS to reach the new `ssthresh`. Slow starting after every loss detection deteriorates the performance given the low bandwidth utilization during SS. When the loss is detected via Timeout, SS is unavoidable since the ACK clock has stopped and SS is required to fill smoothly the pipe. However, in the FRXT case, ACKs still arrive at the source and losses can be recovered without SS. This is the objective of the new versions of TCP (Reno, New Reno, SACK, etc.) [7] that call a Fast Recovery (FRCV) algorithm to retransmit the losses while maintaining enough packets in the network to preserve the ACK clock. Once losses are recovered, this algorithm ends and a normal CA is called. If FRCV fails, the ACK stream stops, a Timeout occurs and the source resorts to SS as with Tahoe.

3 Large Bandwidth-Delay Product

The increase in link speed (e.g. Optic Fibers at Gbps) has led to paths of large BDP. TCP window must be able to reach large values in order to use efficiently the available bandwidth. Large windows, up to 2^{30} Bytes, are now possible with the window scale TCP option [2]. But at large windows, a congestion may lead to the loss of many packets from the same connection. An efficient FRCV is then required to correct many losses from the same window. Also, at large BDP, network buffers have an important impact on the performance. These buffers must be well dimensioned and must scale with the BDP.

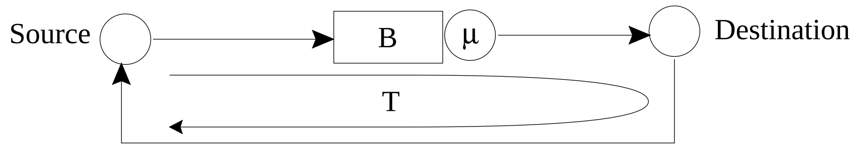

Figure 1: Single node network model

3.1 Fast Recovery

FRCV uses 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 objective is to preserve the ACK clock in order to avoid the Timeout.

The difference between the different versions of TCP is in the estimation of the number of packets in flight during FRCV. Reno [7] considers that every duplicate ACK is a signal that a packet has left the network. The problem of Reno is that it leaves FRCV when an ACK for the first loss in a window is received. This prohibits the source from detecting the other losses with FRXT [7]. A long Timeout is required to detect the other losses. New-Reno [7, 11] has been proposed to overcome this problem. The idea is to stay in FRCV until all the losses in the same window are recovered. Partial ACKs are used to detect the multiple losses in the same window. This avoids the Timeout but cannot result in a recovery faster than one loss per RTT. The source needs to wait for the ACK of the retransmission to discover the next loss. Another problem of Reno and New-Reno is that they rely on ACKs to estimate the number of packets in flight. ACKs can be lost on the return path which results in an underestimation of the number of packets that have left the network, thus in an underutilization of the bandwidth during FRCV and in the case of Reno, a possible failure of FRCV.

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) [2], a TCP option containing the three blocks of contiguous data most recently received at the destination. Many algorithms have been proposed to use this information during FRCV. We find TCP-SACK [7] 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) [1] which relies on SACK in the estimation of the number of packets in the pipe. The number and the identity of packets to transmit during FRCV is decoupled from the ACK clock in contrast to TCP-SACK where the identity is only decoupled.

3.2 Impact of buffer size on TCP performance

SS results in bursts of packets sent at a rate exceeding the bottleneck bandwidth. When the receiver acknowledges every data packet, the rate of these bursts is equal to twice the bottleneck bandwidth. If network buffers are not well dimensioned, they will overflow early during SS before reaching the network capacity. This will result in an underestimation of the available bandwidth and a deterioration in TCP performance.

Early losses during SS were first analyzed in [13]. The network is modeled with a single bottleneck node of bandwidth μ, buffer B and two-way propagation delay T (Figure 1). The authors consider a long Tahoe connection where the aim of SS is to reach quickly without losses a SS threshold equal to half the pipe size ($(B + \mu T)/2$). In the case of a receiver that acknowledges every data packet, they find that a buffer larger than one third the BDP is required ($\mu T/3$). Their analysis can be extended to a SS phase with a different threshold, mainly to the SS phase at the beginning of the connection where `ssthresh` is set to a default value. As an example, it has been proposed in [11] to set the threshold at the beginning of the connection to the BDP in order to switch to CA before the occurrence of losses. This will not work if the buffer is smaller than half the BDP ($\mu T/2$). In [3], we study the problem of early buffer overflow during SS in case of multiple routers in tandem. We show that, due to the high rate at which packets are sent during SS, queues can build up in routers preceding the bottleneck as well. Buffers in these routers must be also well dimensioned otherwise they overflow during SS and limit the performance even though they are faster then the bottleneck. With small buffers, losses during SS are not a signal of network congestion but rather a signal of a transient congestion due to the bursty nature of SS traffic.

Now in CA, packets are transmitted at approximately the bottleneck bandwidth. A loss occurs when the window reaches the pipe size. The source divides its window by two and starts a new cycle. To get always a throughput approximately equal to the bottleneck bandwidth, the window after the reduction must be larger than the BDP. This requires a buffer larger than the BDP. Note that we are talking about Drop Tail buffers which start to drop incoming packets when the buffer is full. Active buffers as RED (Random Early Detection) [9] start to drop packets when the average queue length exceeds some threshold. When a RED buffer is crossed by a single connection, the threshold should be larger than 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. In case of multiple connections, a lower threshold is sufficient given that a small number of connections reduce their windows upon congestion in contrast to Drop Tail buffers where often all the connections reduce their windows simultaneously.

4 Round-Trip Time

Long RTTs are becoming prevalent with the introduction in the Internet of satellite links. A long RTT reduces the rate at which the window increases which is a function of the number of ACKs received and doesn't account for the RTT. This poses many problems to TCP. First, it increases the duration of SS which is a transitory phase designed to fill quickly but smoothly the pipe [1]. Given the low bandwidth utlization during SS, this deteriorates the performance of TCP transfers particularly short ones (e.g. WEB transfers). Second, it causes an unfairness in the allocation of the bottleneck bandwidth. Many works have shown the bias of TCP against connections with long RTT [13]. Small RTT connections increase their rates more quickly and grab most of the available bandwidth. The average throughput of a connection has been shown to vary as the inverse of T^{α} where α a factor between 1 and 2 [13].

4.1 Improving Slow Start performance

Many solutions have been proposed to reduce the time taken by SS on long delay paths. These solutions can be divided into three categories. Some change the window increase algorithm of TCP, others solve the problem at the application level and the others solve it inside the network.

4.1.1 TCP level solutions

The first proposition was to use a larger window than one packet at the beginning of SS [1]. An initial window of maximum 4 packets has been proposed. Another proposition, called Byte Counting, was to account for the number of bytes covered by an ACK while increasing the window rather than the number of ACKs [1]. To avoid long bursts in case of large gaps in the ACK stream, a limit on the maximum window increase has been proposed (Limited Byte Counting).

These solutions try to solve the problem while preserving the ACK clock. They result in an increase in TCP burstiness and an overload on network buffers. Another type of solutions try to solve the problem by introducing some kind of packet spacing (e.g. Rate Based Spacing proposal [1]). The source transmits directly at a large window without overloading the network. Once the large window is reached, the ACK clock takes the hand. This lets the source avoid a considerable part of SS.

4.1.2 Application level solutions

The problem is solved at the application level without changing TCP. The first solution (e.g. XFTP [1]), consists in establishing many parallel TCP connections for the same transfer. This accelerates the growth of the resultant window but increases the aggressiveness of the transfer and hence the losses in the network. An adaptive mechanism has been proposed for XFTP 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 [1]). Only the first object suffer from the long SS phase. The remaining objects are transfered at a high rate. The low throughput during SS is compensated by the long time we stay in CA.

4.1.3 Network level solutions

The problem is solved inside the network rather than in hosts. It is worthy when a long delay link is located on the path. In order to decrease the RTT, the long delay link is eliminated from the feedback loop by acknowledging packets at the input of this link (A in Figure 2). Packets are then transmitted on the long delay link using an optimized transport protocol (e.g. the STP protocol proposed in [10]). This transport protocol is tuned to increase quickly its transmission rate without the need for a long SS. Once arriving at the output (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, any loss between the input of the link (A) and the destination must be locally retransmitted on behalf the source. Also, ACKs from the receiver must be silently discarded (at B) to not confuse the source. This approach is called TCP-Spoofing.

The main gain in performance comes from not using SS on the long delay link. The window increases quickly which improves the performance. But spoofing still has many drawbacks. First, it breaks the end-to-end semantics of TCP. A packet is acknowledged before reaching its destination. Also, it doesn't work when encryption is accomplished at the IP layer and it introduces a heavy overload on network routers. Further, the transfer is vulnerable to path

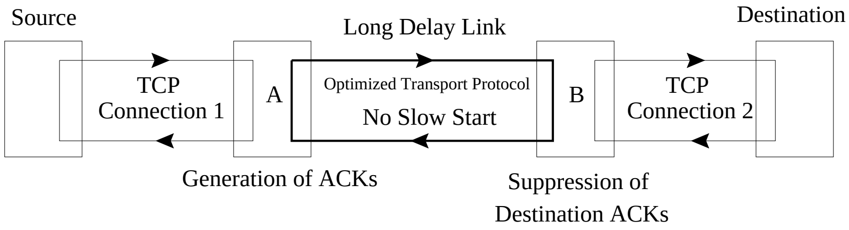

Figure 2: Spoofing : Elimination of the long delay link from the feedback loop

changes and symmetric paths are required to be able to discard the ACKs before they reach the source. Spoofing can be seen as a particular solution to some long delay links. It is interesting when the long delay link is the last hop to the destination. This solution is often used in networks providing a high speed access to the Internet via GEO satellite links.

4.2 Solutions for unfairness

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 RTT connections. The second tries to solve the problem inside the network. Some intelligence is proposed to be added in network nodes in order to allocate the bandwidth fairly among different connections.

As an example of a TCP-level solution, we find the proposition in [8] called Constant Rate algorithm. The window is increased in CA by a factor inversely proportional to RTT^2. The result is a constant increase rate of the throughput regardless of the RTT, thus a better fairness. The first problem in this proposition is the choice of the increase rate. Also, accelerating the window growth while preserving the ACK clock results in large bursts for long RTT connections.

Inside the network, fairness is improved by isolating the different connections from each others. Given that congestion control in TCP is based on losses, isolation means that a congested node must manage its buffer intelligently to distribute drops on the different connections in a way that they get the same throughput. Many buffer management policies have been proposed. Some of these policies such as RED [9] drop incoming packets with a certain probability when the queue length or its average exceed a certain threshold. This distributes losses on the different connections proportionally to their throughput without requiring any per-connection state. However, it has been shown in [15] that dropping packets proportionally to the throughput doesn't lead always to fairness especially if the bottleneck is crossed by a non-responsive traffic. With a FIFO scheduler, the connection share of the bandwidth is proportional to its share of the buffer. A better fairness requires a control of the buffer occupancy of each connection. We find here the other set of policies as Flow RED [15] that try to improve fairness by sharing the buffer space fairly between active connections. This ensures that each connection has at least a certain number of places in the queue which isolates connections sending at small rates from aggressive ones. This improves fairness but at the same time increases buffer management overhead compared to a general drop policy as RED.

Solving the fairness problem at TCP level has the advantage of keeping routers simple, but it is not enough given the prevalence of non-TCP-friendly traffics. Some mechanisms in network nodes are required to protect conservative TCP flows from aggressive ones. Network mechanisms are also required to ensure fairness at a level below or above TCP, say at the user or at the application level. A user (e.g. running XFTP) may cheat and establish many TCP connections in order to increase its share of the bandwidth. The packets generated by this user must be considered by the network as a single flow. This requires an aggregation in flows of TCP connections. The level of aggregation determines the level of fairness we want.

5 Non-congestion losses

TCP considers the loss of packets as a signal of network congestion and reduces its window consequently. This results in severe throughput deterioration when packets are lost for other reasons than congestion. Non-congestion losses are mostly caused by transmission errors. A packet may be corrupted while crossing a bad quality link (e.g. wireless link). The solutions proposed to this problem can be divided into two main categories. The first one consists in hiding the lossy parts of the Internet so that only congestion losses are detected at the source. The second type of solutions consists in enhancing TCP with some mechanisms to help it to distinguish between different types of losses.

5.1 Hiding non-congestion losses

Non-congestion are recovered locally without the intervention of the source. This can be accomplished at the link level or at TCP level. A link is any transmission medium located between two adjacent routers. The objective is to approach the typical network TCP is tuned to: A network formed of the interconnection of good quality transmission media.

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, ARQ may interfere with TCP mechanisms [6]. If the link layer doesn't provide an in-sequence delivery of packets, TCP packets following the loss keep arriving at the destination and trigger the transmission of duplicate ACKs. These duplicate ACKs reach the source while the link layer is retransmitting the packet. This causes an unnecessary window reduction which we try to avoid. The proposed solution to this problem was to use an ARQ protocol aware of TCP (Figure 3). The link layer suppresses the duplicate ACKs (at R) so that they don't reach the source. If the link layer fails to retransmit the packet, the source will timeout and retransmit the packet itself. This solution is applicable only when the lossy link is the last hop to the destination. If the lossy link is followed by other routers, congestion losses will be hidden which must be avoided.

FEC consists in sending 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 bandwidth. Also, the computation of this redundant information requires extra CPU processing time, memory and blocking delay. But, the advantages of FEC are worth the cost. First, corrupted packets are corrected directly without retransmission which is important for long delay lossy links as satellites ones. Also, FEC doesn't interfere with TCP mechanisms. For these reasons, this technique has been recommended in a satellite environment [2]. Convolutional coding, Viterbi decoding together with interleaving

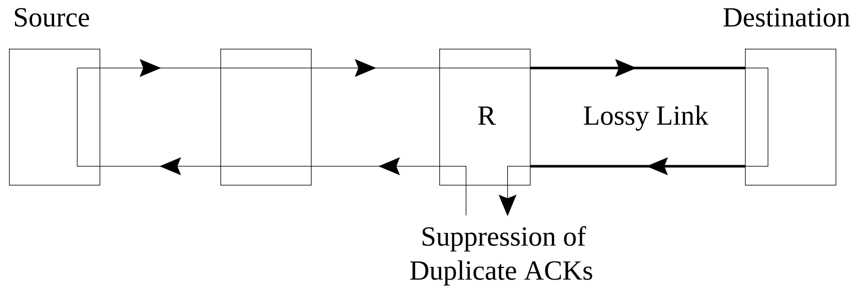

Figure 3: A TCP-aware link layer

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 rather than at the link level. 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 technique has been proposed 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 exist.

The first implementation (e.g. Indirect TCP [6]) consists in terminating the originating TCP connection at the entry of the lossy link. The agent acknowledges the packets and takes care of handing them to the destination. A TCP connection well tuned to a lossy environment (e.g. TCP-SACK [7]) can be established across the lossy network. A different transport protocol can also be used. This solution 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 (Snoop protocol [6]), respects the end-to-end semantics. The intermediate agent doesn't terminate the TCP connection. It just keeps copies of data packets and doesn't generate any artificial ACK. Non-duplicate ACKs sent by the destination are forwarded to the source. Duplicate ACKs are stopped. A packet is retransmitted locally 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. As in the link-level case, an interference may happen between the source and the agent mechanisms. In fact, this solution is no other than a link-level recovery implemented at the TCP level. Again, because it hides all losses, congestion losses must not occur between the Snoop agent and the destination.

5.2 End-to-end solutions

The improvement of the quality of Internet links will not hide all non-congestion losses. First, even if they are concentrated on some links, such kind of losses may appear anywhere in the network and it seems impossible to track them all. Second, local mechanisms such as FEC may fail in the recovery from lost packets. The addition of some end-to-end mechanisms to improve TCP reaction to non-congestion losses should improve further the performance.

Two approaches exist in the literature. The first one consists in informing the source explicitly of the occurrence of a non-congestion loss via an Explicit Loss Notification (ELN) signal [6]. The source reacts by retransmitting the lost 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 a cellular network. 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 approach consists in improving the control of congestion provided by TCP rather than the recovery from non-congestion losses. We mention it here because it consists a step towards a solution to the problem of losses on an end-to-end-basis. The 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. As solutions, we find the Vegas version of TCP [4] and the Explicit Congestion Notification (ECN) proposal [1]. 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 sent 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), congestion losses will considerably decrease. The remaining losses could be considered as mostly caused by problems other than congestion. Given that non-congestion losses require 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 severely to losses.

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 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 instead. A reduction of the window is necessary in this case. For these reasons, these solutions still consider losses as a congestion signal and reduce their window consequently.

6 Bandwidth Asymmetry

TCP uses the ACK clock to predict what is happening inside the network. It assumes implicitly that the reverse channel has enough bandwidth so that ACKs traverse it without being disturbed. This was almost true with the so-called symmetric networks where the forward and the reverse directions have the same bandwidth. However, some of todays networks (e.g. Direct Broadcast Satellite networks and Asymmetric Digital Subscriber Loop networks) tend to increase the capacity of the forward direction using satellite links or cables, whereas a low speed channel, such as a dial-up modem line, is used to carry ACKs back to the source (Figure 4). Even if ACKs are of small size with respect to data packets, the reverse channel is unable to carry the high rate of ACKs. The result is congestion and losses on the ACK channel (at A).

The congestion in A increases the RTT of the connection and causes losses of ACKs. The increase in the RTT reduces the throughput and increases the end-to-end delay. Also, it slows the window growth 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

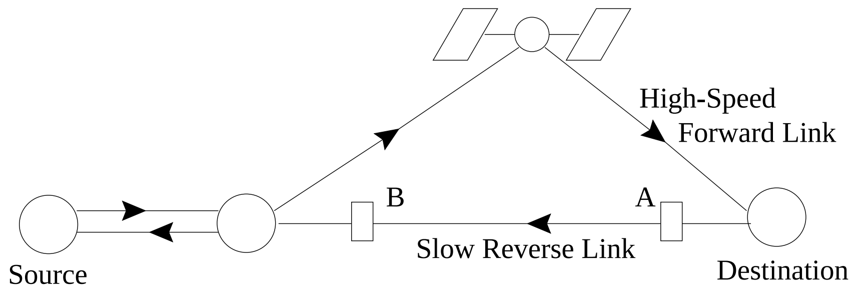

Figure 4: An asymmetric path

the transmission. The window slides quickly upon the receipt of an ACK covering multiple lost ACKs and a burst of packets is sent which may overwhelm the network buffers on the forward direction [5, 14]. 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.

The proposed solutions to this problem can be divided into receiver side solutions which try to solve the problem by reducing the congestion on the return path, and source side solutions which try to reduce TCP burstiness.

The first receiver side solution consists in compressing the headers of TCP/IP packets on the slow channel (A-B) to increase its capacity in terms of ACKs per unit of time (e.g. SLIP header compression[1]). It profits from the fact that most of the information in a TCP/IP header doesn't change along the connection lifetime.

The other solutions propose to reduce the rate of ACKs so as to avoid congestion. The first proposition is to delay ACKs at the destination [5]. 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 [5] keeps the destination unchanged and filters ACKs at A. When an ACK arrives, the buffer is scanned to see if another ACK (or a certain number of ACKs) of the same connection is buffered. If so, the old ACK is substituted by the new one. ACKs are filtered to match their rate to the rate of the reverse channel. Normally, in the absence of artificial filtering, ACKs are filtered sometime later when the buffer gets full. The advantage of this solution is that the filtering is accomplished before the increase in the RTT.

Solutions at the sender side are proposed to reduce the burstiness of TCP. Note that this problem is caused by the reliance of TCP on the ACK clock and we believe that it cannot be completely solved without any kind of packet spacing. First, a limit on the size of bursts sent by TCP has been proposed. But, in case of a systematic loss of ACKs, limiting the size of bursts limits the throughput of the connection. Second, it has been proposed in [5] to reconstruct the ACK clock at the output of the slow channel (at B). When an ACK arrives at B, all the missing ACKs are generated, spaced by a time interval derived from the average rate at which ACKs leave the slow channel. This reconstruction may consist a solution to this particular problem. However, the general problem of TCP burstiness upon loss of ACKs still exists.

7 Conclusions

In this work, we summarize most of the problems that face TCP in the Internet of today. We present the impact of the new characteristics of Internet paths on TCP performance, mainly the bandwidth-delay product, the round-trip time, the non-congestion losses and the bandwidth asymmetry. The study is accomplished independently of the network type. We outline some of the solutions proposed in the literature. We show the utility of these solutions on a general path having a certain characteristic. We believe that the two main problems of TCP to be solved are burstiness and the coupling between congestion detection and error control. Solutions to these problems are required to make TCP cope with the heterogeneity of todays transmission media on an end-to-end basis.

References

[1] M. Allman et al., " Ongoing TCP Research Related to Satellites", *Internet Draft*, work in progress, Sep. 1999.

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

[3] C. Barakat and E. Altman, "Analysis of TCP with Several Bottleneck Nodes", *IEEE GLOBECOM*, Dec. 1999.

[4] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", *IEEE JSAC*, Oct. 1995.

[5] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance", *ACM MOBICOM*, Sep. 1997.

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

[7] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *Computer Communication Review*, Jul. 1996.

[8] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic", *Computer Communication Review*, Oct. 1991.

[9] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, Aug. 1993.

[10] T. Henderson and R.H. Katz, "Transport Protocols for Internet-Compatible Satellite Networks", *IEEE JSAC*, Feb. 1999.

[11] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *ACM SIGCOMM*, Aug. 1996.

[12] V. Jacobson, "Congestion avoidance and control", *ACM SIGCOMM*, Aug. 1988.

[13] 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*, Jun. 1997.

[14] 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", *IEEE INFOCOM*, Apr.1997.

[15] D. Lin and R. Morris, "Dynamics of Random Early Detection", *ACM SIGCOMM*, Sep. 1997.

Biographies

CHADI BARAKAT received his diploma degree in electrical and electronics engineering from the Lebanese University in Beirut in 1997. In 1998, he obtained the DEA degree in networks and distributed systems from the University of Nice Sophia Antipolis, France. Since 1998, he is working towards a PhD degree within the MISTRAL team at INRIA Sophia Antipolis. His main research topics are congestion and error control in computer networks, performance evaluation of communication protocols, and integration of new transmission media such as satellite networks into the Internet.

EITAN ALTMAN received the B.Sc. degree in electrical engineering (1984), the B.A. degree in physics (1984) and the Ph.D. degree in electrical engineering (1990), all from the Technion-Israel Institute, Haifa. In (1990) he further received his B.Mus. degree in music composition in Tel-Aviv university. Since 1990, he has been with INRIA (National research institute in informatics and control) in Sophia-Antipolis, France. His current research interests include performance evaluation and control of telecommunication networks, stochastic control and dynamic games. In recent years, he has applied control theoretical techniques in several joint projects with the French telecommunications company – France Télécom.

WALID DABBOUS graduated from the Faculty of Engineering of the Lebanese University in Beirut in 1986. He obtained his DEA and his Doctorat d'Université from the University of Paris XI in 1987 and 1991 respectively. He joined the RODEO Team within INRIA in 1987. He is a staff researcher at INRIA since 1991, and leader of the RODEO team since 1996. His main research topics are high performance communication protocols, congestion control, reliable multicast protocols, audio and video conferencing over the Internet, efficient and flexible protocol architecture design and the integration of new transmission media such as satellite links in the Internet. He is co-chair of the UDLR working group at the IETF.