NASA/CR—1999-209167

Study and Simulation of Enhancements for TCP Performance Over Noisy High Latency Links

Craig Partridge, Tim Shepard and Robert Coulter
BBN Technologies, Cambridge, Massachusetts

Prepared under Contract NAS3–96014

National Aeronautics and
Space Administration

Glenn Research Center

August 1999

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076
Price Code: A05

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100
Price Code: A05

Final Report

Study and Simulation of Enhancements for TCP Performance Over Noisy High Latency Links

Contract: NAS3-96014
Contract Period of Performance: September 1997 to January 1999

BBN Technologies
10 Moulton Street
Cambridge, MA

Principal Investigator: Dr. Craig Partridge
Program Manager: Robert Coulter

Overview

This contract was a continuation of a project started the previous year. The goal is to better understand how TCP behaves over noisy, high latency links such as satellite links and propose improvements to TCP implementations such that TCP might better handle such links.

Major Accomplishments

Advocacy: Project members attended various Internet technical meetings to speak as advocates for improving TCP performance over satellite links. The particular meetings attended were:

- Meetings of the Internet Engineering Task Force (IETF). During the period of performance, the IETF has had an active working group investigating TCP performance issues. Members of this project attended the December 1997 and March 1998 IETF meetings.
- Meetings of the Internet End-to-End Research Group (E2E). The End-To-End research group is where many of the innovative ideas for TCP work have been initially developed in the past ten years. Dr. Partridge is a member of E2E and attended three meetings of the research group in 1998.
- Dr. Partridge gave a keynote speech at the NASA Lewis sponsored workshop on "Satellite Networks: Architectures, Applications, and Technologies" held June 2-4, 1998, in Cleveland, Ohio, at the Sheraton Airport Hotel.

Publications: Project members wrote various papers to highlight the issues of TCP performance over satellite links. Four publications appeared during this contract:

- C. Partridge and T. Shepard, "TCP/IP Performance over Satellite Links," *IEEE Network*, Vol. 11, No. 5, September 1997, pp. 44-49. This paper was written during the previous year of the project but appeared in this year.
- T. Shepard and C. Partridge, "When TCP Starts Up With Four Packets Into Only Three Buffers," *Internet Working Group Requests for Comments*, no. 2416, September 1998. A study that helped justify the IETF's decision to allow TCP to send more data in the initial round-trip.
- M. Allman, S. Floyd and C. Partridge, "Increasing TCP's Initial Window," *Internet Working Group Requests for Comments*, no. 2414, September 1998. The IETF document that approved allowing TCP to send more data in the initial round-trip.
- C. Partridge, "ACK Spacing for High Delay-Bandwidth Paths with Insufficient Buffering." This document was released as an Internet Draft but was not published due to various IETF procedural difficulties.

Implementations: The project also did some implementation work and prepared to do more, to demonstrate improved TCPs. In particular, the group did the following two implementation projects:

- Porting TCP Forward Acknowledgements to NetBSD. Forward Acknowledgements are a mechanism that improve TCP throughput over lossy links.
- Developing initial code to support TCP Pacing. The idea behind TCP Pacing is to space out TCP bursts to reduce loss at undersized queues in the network. We implemented the necessary high-speed timer in the UNIX kernel for this purpose.

End-To-End Research Group
Request for Comments: DRAFT
Category: Informational

C. Partridge
BBN Technologies
Sep 1998

ACK Spacing for High Delay-Bandwidth Paths with Insufficient Buffering

Status of this Memo

An argument is made that the correct way to solve buffering shortages in routers on high delay-bandwidth paths is for routers to space out the TCP acks.

This memo presents thoughts from a discussion held at the July 1997 meeting of the End-To-End (E2E) Research Group. The material presented is a half-baked suggestion and should not be interpreted as an official recommendation of the Research Group. Comments are solicited and should be addressed to the author.

1. Introduction

Suppose you want TCP implementations to be able to fill a 155 Mb/s path. Further suppose that the path includes a satellite in a geosynchronous orbit, so the round trip delay through the path is at least 500 ms, and the delay-bandwidth product is 9.7 megabytes or more.

If we further assume the TCP implementations support TCP Large Windows and PAWS (many do), so they can manage 9.7 MB TCP window, then we can be sure the TCP will eventually start sending at full path rate (unless the satellite channel is very lossy). But it may take a long time to get the TCP up to full speed.

One (of several) possible causes of the delay is a shortage of buffering in routers. To understand this particular problem, consider the following idealized behavior of TCP during slow start. During slow start, for every segment ACKed, the sender transmits two new segments. In effect, this behavior means the sender is transmitting at *twice* the data rate of the segments being ACKed. Keep in mind the separation between ACKs represents (in an ideal world) the rate segments can flow through the bottleneck router in the path. So the sender is bursting data at twice the bottleneck rate, and a queue must be forming during the burst. In the simplest case, the queue is entirely at the bottleneck router, and at the end of the burst, the queue is storing half the data in the burst. (Why

half? During the burst, the sender transmitted at twice the bottleneck rate. Suppose it takes one time unit to send a segment on the bottlenecked link. During the burst the bottleneck will receive two segments in every time unit, but only be able to transmit one segment. The result is a net of one new segment queued every time unit, for the life of the burst.)

TCP will end the slow start phase in response to the first lost datagram. Assuming good quality transmission links, the first lost datagram will be lost because the bottleneck queue overflowed. We would like that loss to occur in the round-trip after the slow start congestion window has reached the delay-bandwidth product. Now consider the buffering required in the bottleneck link during the next to last round trip. The sender will send an entire delay-bandwidth worth of data in one-half a round-trip time (because it sends at twice the channel rate). So for half the round-trip time, the bottleneck router is in the mode of forwarding one segment while receiving two. (For the second half of the round-trip, the router is draining its queue). That means, to avoid losing any segments, the router must have buffering equal to half the delay-bandwidth product, or nearly 5 MB.

Most routers do not have anywhere near 5 MB of buffering for a single link. Or, to express this problem another way, because routers do not have this much buffering, the slow start stage will end prematurely, when router buffering is exhausted. The consequence of ending slow start prematurely is severe. At the end of slow start, TCP goes into congestion avoidance, in which the window size is increased much more slowly. So even though the channel is free, because we did not have enough router buffering, we will transmit slowly for a period of time (until the more conservative congestion avoidance algorithm sends enough data to fill the channel).

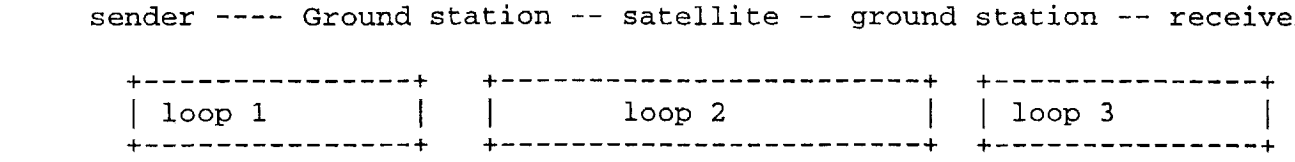
2. What to Do?

So how to get around the shortage of router buffering?

One solution has been proposed, cascading TCPs. We would like to suggest another solution, ACK spacing. Both schemes involve layer violations because they require the router to examine the TCP header.

2.1 Cascading TCPs

One approach is to use cascading TCPs, in which we build a custom TCP for the satellite (or bottleneck) link and insert it between the sender's and receiver's TCPs, as shown below:



This approach can work but is awkward. Among its limitations are: the buffering problem remains (at points of bandwidth mismatches, queues will form); the scheme violates end-to-end semantics of TCP (the sender will get ACKs for data that has not and may never reach the receiver); it constrains the reverse path of the TCP connection to pass through points at which the multiple TCP connections are spliced together (a problem if satellite links are unidirectional); and it doesn't work with end-to-end encryption (i.e. if data above the IP layer is encrypted).

2.2 ACK Spacing

Another approach is to find some way to spread the bursts, either by having the sender spread out the segments, or having the network arrange for the ACKs to arrive at the sender with a two segment spacing (or larger).

Changing the sender is feasible, although it requires very good operating system timers. But it has the disadvantage that only upgraded senders get the performance improvement.

Finding a way for the network to space the ACKs would allow TCP senders to transmit at the right rate, without modification. Furthermore, it can be done by a router. The router simply has to snoop the returning TCP ACKs and spread them out. (Note that if the transmissions are encrypted, in many scenarios the router can still figure out which segments are likely TCP ACKs and spread them out).

There are some difficult issues with this approach. The most notable ones are:

1. What algorithm to use to determine the proper ACK spacing.

2. Related to (1), it may be necessary to known when a TCP is in slow-start vs. congestion-avoidance, as the desired spacing between ACKs is likely to be different in the two phases.

3. What to do about assymetric routes (if anything). The scheme works so long as the router sees the ACKs (it does not have to see the related data). However, if the ACKs do not return through the ACK-spacing router, it is not possible to do ACK spacing.

4. How much, if at all, does ack compression between the respacing point and the sender undo the effects of ack spacing?

5. How much per-flow (soft) state is required in the ACK spacing router?

Despite these challenges the approach has appeal. Changing software in a few routers (particularly those at likely bottleneck links) on high delay-bandwidth paths could give a performance boost to lots of TCP connections.

Security Issues

ACK spacing introduces no new security issues. ACK spacing does not change the contents of any datagram. It simply delays some datagrams in transit, just as a queue might. TCP and other higher layer protocols are already required to work correctly with queueing delays, and indeed, work correctly when encountering far more serious transmission errors such as damage, loss, duplication and reordering [2].

Credit and Disclaimer

The particular idea of ACK spacing was developed by during the meeting by Mark Handley and Van Jacobson in response to an issue raised by the author, and was inspired, in part by ideas to enhance wireless routers to improve TCP performance [1].

Intellectual Property Issues

The author has learned from the IETF that parties may be attempting to patent schemes similar to this one. Readers are advised to check with the IETF to learn of any intellectual property rights issues.

References

1. H. Balakrishnan, V.N. Padmanabhan, S. Seshan and R.H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", Proc. ACM SIGCOMM '96, pp. 256-269.
2. J. Postel, ed. Transmission Control Protocol RFC-793, Internet Requests for Comments, No. 793, September 1981, p. 4.

Network Working Group
Request for Comments: 2414
Category: Experimental

M. Allman
NASA Lewis/Sterling Software
S. Floyd
LBNL
C. Partridge
BBN Technologies
September 1998

Increasing TCP's Initial Window

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document specifies an increase in the permitted initial window for TCP from one segment to roughly 4K bytes. This document discusses the advantages and disadvantages of such a change, outlining experimental results that indicate the costs and benefits of such a change to TCP.

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1. TCP Modification

This document specifies an increase in the permitted upper bound for TCP's initial window from one segment to between two and four segments. In most cases, this change results in an upper bound on the initial window of roughly 4K bytes (although given a large segment size, the permitted initial window of two segments could be significantly larger than 4K bytes). The upper bound for the initial window is given more precisely in (1):

```
min (4*MSS, max (2*MSS, 4380 bytes))                (1)
```

Equivalently, the upper bound for the initial window size is based on the maximum segment size (MSS), as follows:

```
If (MSS <= 1095 bytes)
    then win <= 4 * MSS;
If (1095 bytes < MSS < 2190 bytes)
    then win <= 4380;
If (2190 bytes <= MSS)
    then win <= 2 * MSS;
```

This increased initial window is optional: that a TCP MAY start with a larger initial window, not that it SHOULD.

This upper bound for the initial window size represents a change from RFC 2001 [S97], which specifies that the congestion window be initialized to one segment. If implementation experience proves successful, then the intent is for this change to be incorporated into a revision to RFC 2001.

This change applies to the initial window of the connection in the first round trip time (RTT) of transmission following the TCP three-way handshake. Neither the SYN/ACK nor its acknowledgment (ACK) in the three-way handshake should increase the initial window size above that outlined in equation (1). If the SYN or SYN/ACK is lost, the initial window used by a sender after a correctly transmitted SYN MUST be one segment.

TCP implementations use slow start in as many as three different ways: (1) to start a new connection (the initial window); (2) to restart a transmission after a long idle period (the restart window); and (3) to restart after a retransmit timeout (the loss window). The change proposed in this document affects the value of the initial window. Optionally, a TCP MAY set the restart window to the minimum of the value used for the initial window and the current value of cwnd (in other words, using a larger value for the restart window should never increase the size of cwnd). These changes do NOT change the loss window, which must remain 1 segment (to permit the lowest possible window size in the case of severe congestion).

2. Implementation Issues

When larger initial windows are implemented along with Path MTU Discovery [MD90], and the MSS being used is found to be too large, the congestion window 'cwnd' SHOULD be reduced to prevent large bursts of smaller segments. Specifically, 'cwnd' SHOULD be reduced by the ratio of the old segment size to the new segment size.

When larger initial windows are implemented along with Path MTU Discovery [MD90], alternatives are to set the "Don't Fragment" (DF) bit in all segments in the initial window, or to set the "Don't Fragment" (DF) bit in one of the segments. It is an open question which of these two alternatives is best; we would hope that

implementation experiences will shed light on this. In the first case of setting the DF bit in all segments, if the initial packets are too large, then all of the initial packets will be dropped in the network. In the second case of setting the DF bit in only one segment, if the initial packets are too large, then all but one of the initial packets will be fragmented in the network. When the second case is followed, setting the DF bit in the last segment in the initial window provides the least chance for needless retransmissions when the initial segment size is found to be too large, because it minimizes the chances of duplicate ACKs triggering a Fast Retransmit. However, more attention needs to be paid to the interaction between larger initial windows and Path MTU Discovery.

The larger initial window proposed in this document is not intended as an encouragement for web browsers to open multiple simultaneous TCP connections all with large initial windows. When web browsers open simultaneous TCP connections to the same destination, this works against TCP's congestion control mechanisms [FF98], regardless of the size of the initial window. Combining this behavior with larger initial windows further increases the unfairness to other traffic in the network.

3. Advantages of Larger Initial Windows

1. When the initial window is one segment, a receiver employing delayed ACKs [Bra89] is forced to wait for a timeout before generating an ACK. With an initial window of at least two segments, the receiver will generate an ACK after the second data segment arrives. This eliminates the wait on the timeout (often up to 200 msec).

2. For connections transmitting only a small amount of data, a larger initial window reduces the transmission time (assuming at most moderate segment drop rates). For many email (SMTP [Pos82]) and web page (HTTP [BLFN96, FJGFBL97]) transfers that are less than 4K bytes, the larger initial window would reduce the data transfer time to a single RTT.

3. For connections that will be able to use large congestion windows, this modification eliminates up to three RTTs and a delayed ACK timeout during the initial slow-start phase. This

would be of particular benefit for high-bandwidth large-propagation-delay TCP connections, such as those over satellite links.

4. Disadvantages of Larger Initial Windows for the Individual Connection

In high-congestion environments, particularly for routers that have a bias against bursty traffic (as in the typical Drop Tail router queues), a TCP connection can sometimes be better off starting with

an initial window of one segment. There are scenarios where a TCP connection slow-starting from an initial window of one segment might not have segments dropped, while a TCP connection starting with an initial window of four segments might experience unnecessary retransmits due to the inability of the router to handle small bursts. This could result in an unnecessary retransmit timeout. For a large-window connection that is able to recover without a retransmit timeout, this could result in an unnecessarily-early transition from the slow-start to the congestion-avoidance phase of the window increase algorithm. These premature segment drops are unlikely to occur in uncongested networks with sufficient buffering or in moderately-congested networks where the congested router uses active queue management (such as Random Early Detection [FJ93, RFC2309]).

Some TCP connections will receive better performance with the higher initial window even if the burstiness of the initial window results in premature segment drops. This will be true if (1) the TCP connection recovers from the segment drop without a retransmit timeout, and (2) the TCP connection is ultimately limited to a small congestion window by either network congestion or by the receiver's advertised window.

5. Disadvantages of Larger Initial Windows for the Network

In terms of the potential for congestion collapse, we consider two separate potential dangers for the network. The first danger would be a scenario where a large number of segments on congested links were duplicate segments that had already been received at the receiver. The second danger would be a scenario where a large number of segments on congested links were segments that would be dropped later in the network before reaching their final destination.

In terms of the negative effect on other traffic in the network, a potential disadvantage of larger initial windows would be that they increase the general packet drop rate in the network. We discuss these three issues below.

Duplicate segments:

As described in the previous section, the larger initial window could occasionally result in a segment dropped from the initial window, when that segment might not have been dropped if the sender had slow-started from an initial window of one segment. However, Appendix A shows that even in this case, the larger initial window would not result in the transmission of a large number of duplicate segments.

Segments dropped later in the network:

How much would the larger initial window for TCP increase the number of segments on congested links that would be dropped before reaching their final destination? This is a problem that

can only occur for connections with multiple congested links, where some segments might use scarce bandwidth on the first congested link along the path, only to be dropped later along the path.

First, many of the TCP connections will have only one congested link along the path. Segments dropped from these connections do not "waste" scarce bandwidth, and do not contribute to congestion collapse.

However, some network paths will have multiple congested links, and segments dropped from the initial window could use scarce bandwidth along the earlier congested links before ultimately being dropped on subsequent congested links. To the extent that the drop rate is independent of the initial window used by TCP segments, the problem of congested links carrying segments that will be dropped before reaching their destination will be similar for TCP connections that start by sending four segments or one segment.

An increased packet drop rate:

For a network with a high segment drop rate, increasing the TCP initial window could increase the segment drop rate even further. This is in part because routers with Drop Tail queue management have difficulties with bursty traffic in times of congestion. However, given uncorrelated arrivals for TCP connections, the larger TCP initial window should not significantly increase the segment drop rate. Simulation-based explorations of these issues are discussed in Section 7.2.

These potential dangers for the network are explored in simulations and experiments described in the section below. Our judgement would be, while there are dangers of congestion collapse in the current Internet (see [FF98] for a discussion of the dangers of congestion collapse from an increased deployment of UDP connections without end-to-end congestion control), there is no such danger to the network from increasing the TCP initial window to 4K bytes.

6. Typical Levels of Burstiness for TCP Traffic.

Larger TCP initial windows would not dramatically increase the burstiness of TCP traffic in the Internet today, because such traffic is already fairly bursty. Bursts of two and three segments are already typical of TCP [Flo97]; A delayed ACK (covering two previously unacknowledged segments) received during congestion avoidance causes the congestion window to slide and two segments to be sent. The same delayed ACK received during slow start causes the window to slide by two segments and then be incremented by one segment, resulting in a three-segment burst. While not necessarily typical, bursts of four and five segments for TCP are not rare.

Assuming delayed ACKs, a single dropped ACK causes the subsequent ACK to cover four previously unacknowledged segments. During congestion avoidance this leads to a four-segment burst and during slow start a five-segment burst is generated.

There are also changes in progress that reduce the performance problems posed by moderate traffic bursts. One such change is the deployment of higher-speed links in some parts of the network, where a burst of 4K bytes can represent a small quantity of data. A second change, for routers with sufficient buffering, is the deployment of queue management mechanisms such as RED, which is designed to be tolerant of transient traffic bursts.

7. Simulations and Experimental Results

7.1 Studies of TCP Connections using that Larger Initial Window

This section surveys simulations and experiments that have been used to explore the effect of larger initial windows on the TCP connection using that larger window. The first set of experiments explores performance over satellite links. Larger initial windows have been shown to improve performance of TCP connections over satellite channels [All97b]. In this study, an initial window of four segments (512 byte MSS) resulted in throughput improvements of up to 30% (depending upon transfer size). [KAGT98] shows that the use of larger initial windows results in a decrease in transfer time in HTTP tests over the ACTS satellite system. A study involving simulations

of a large number of HTTP transactions over hybrid fiber coax (HFC) indicates that the use of larger initial windows decreases the time required to load WWW pages [Nic97].

A second set of experiments has explored TCP performance over dialup modem links. In experiments over a 28.8 bps dialup channel [All97a, AHO98], a four-segment initial window decreased the transfer time of a 16KB file by roughly 10%, with no accompanying increase in the drop rate. A particular area of concern has been TCP performance over low speed tail circuits (e.g., dialup modem links) with routers with small buffers. A simulation study [SP97] investigated the effects of using a larger initial window on a host connected by a slow modem link and a router with a 3 packet buffer. The study concluded that for the scenario investigated, the use of larger initial windows was not harmful to TCP performance. Questions have been raised concerning the effects of larger initial windows on the transfer time for short transfers in this environment, but these effects have not been quantified. A question has also been raised concerning the possible effect on existing TCP connections sharing the link.

7.2 Studies of Networks using Larger Initial Windows

This section surveys simulations and experiments investigating the impact of the larger window on other TCP connections sharing the path. Experiments in [All97a, AHO98] show that for 16 KB transfers

to 100 Internet hosts, four-segment initial windows resulted in a small increase in the drop rate of 0.04 segments/transfer. While the drop rate increased slightly, the transfer time was reduced by roughly 25% for transfers using the four-segment (512 byte MSS) initial window when compared to an initial window of one segment.

One scenario of concern is heavily loaded links. For instance, a couple of years ago, one of the trans-Atlantic links was so heavily loaded that the correct congestion window size for a connection was about one segment. In this environment, new connections using larger initial windows would be starting with windows that were four times too big. What would the effects be? Do connections thrash?

A simulation study in [PN98] explores the impact of a larger initial window on competing network traffic. In this investigation, HTTP and FTP flows share a single congested gateway (where the number of HTTP and FTP flows varies from one simulation set to another). For each simulation set, the paper examines aggregate link utilization and packet drop rates, median web page delay, and network power for the FTP transfers. The larger initial window generally resulted in increased throughput, slightly-increased packet drop rates, and an increase in overall network power. With the exception of one scenario, the larger initial window resulted in an increase in the

drop rate of less than 1% above the loss rate experienced when using a one-segment initial window; in this scenario, the drop rate increased from 3.5% with one-segment initial windows, to 4.5% with four-segment initial windows. The overall conclusions were that increasing the TCP initial window to three packets (or 4380 bytes) helps to improve perceived performance.

Morris [Mor97] investigated larger initial windows in a very congested network with transfers of size 20K. The loss rate in networks where all TCP connections use an initial window of four segments is shown to be 1-2% greater than in a network where all connections use an initial window of one segment. This relationship held in scenarios where the loss rates with one-segment initial windows ranged from 1% to 11%. In addition, in networks where connections used an initial window of four segments, TCP connections spent more time waiting for the retransmit timer (RTO) to expire to resend a segment than was spent when using an initial window of one segment. The time spent waiting for the RTO timer to expire represents idle time when no useful work was being accomplished for that connection. These results show that in a very congested environment, where each connection's share of the bottleneck bandwidth is close to one segment, using a larger initial window can cause a perceptible increase in both loss rates and retransmit timeouts.

8. Security Considerations

This document discusses the initial congestion window permitted for TCP connections. Changing this value does not raise any known new security issues with TCP.

9. Conclusion

This document proposes a small change to TCP that may be beneficial to short-lived TCP connections and those over links with long RTTs (saving several RTTs during the initial slow-start phase).

10. Acknowledgments

We would like to acknowledge Vern Paxson, Tim Shepard, members of the End-to-End-Interest Mailing List, and members of the IETF TCP Implementation Working Group for continuing discussions of these issues for discussions and feedback on this document.

11. References

[All97a] Mark Allman. An Evaluation of TCP with Larger Initial Windows. 40th IETF Meeting -- TCP Implementations WG. December, 1997. Washington, DC.

[AHO98] Mark Allman, Chris Hayes, and Shawn Ostermann, An Evaluation of TCP with Larger Initial Windows, March 1998. Submitted to ACM Computer Communication Review. URL: "http://gigahertz.lerc.nasa.gov/~mallman/papers/initwin.ps".

[All97b] Mark Allman. Improving TCP Performance Over Satellite Channels. Master's thesis, Ohio University, June 1997.

[BLFN96] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[Bra89] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989.

[FF96] Fall, K., and Floyd, S., Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. Computer Communication Review, 26(3), July 1996.

[FF98] Sally Floyd, Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. Submitted to IEEE Transactions on Networking. URL "http://www-nrg.ee.lbl.gov/floyd/end2end-paper.html".

[FJGFBL97] Fielding, R., Mogul, J., Gettys, J., Frystyk, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[FJ93] Floyd, S., and Jacobson, V., Random Early Detection gateways for Congestion Avoidance. IEEE/ACM Transactions

on Networking, V.1 N.4, August 1993, p. 397-413.

[Flo94] Floyd, S., TCP and Explicit Congestion Notification. Computer Communication Review, 24(5):10-23, October 1994.

[Flo96] Floyd, S., Issues of TCP with SACK. Technical report, January 1996. Available from http://www-nrg.ee.lbl.gov/floyd/.

[Flo97] Floyd, S., Increasing TCP's Initial Window. Viewgraphs, 40th IETF Meeting - TCP Implementations WG. December, 1997. URL "ftp://ftp.ee.lbl.gov/talks/sf-tcp-ietf97.ps".

[KAGT98] Hans Kruse, Mark Allman, Jim Griner, Diepchi Tran. HTTP Page Transfer Rates Over Geo-Stationary Satellite Links. March 1998. Proceedings of the Sixth International Conference on Telecommunication Systems. URL "http://gigahertz.lerc.nasa.gov/~mallman/papers/nash98.ps".

[MD90] Mogul, J., and S. Deering, "Path MTU Discovery", RFC 1191, November 1990.

[MMFR96] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[Mor97] Robert Morris. Private communication, 1997. Cited for acknowledgement purposes only.

[Nic97] Kathleen Nichols. Improving Network Simulation with Feedback. Com21, Inc. Technical Report. Available from http://www.com21.com/pages/papers/068.pdf.

[PN98] Poduri, K., and K. Nichols, "Simulation Studies of Increased Initial TCP Window Size", RFC 2415, September 1998.

[Pos82] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, August 1982.

[RF97] Ramakrishnan, K., and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IPv6 and to TCP", Work in Progress.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[S97] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.

[SP97] Shepard, T., and C. Partridge, "When TCP Starts Up With Four Packets Into Only Three Buffers", RFC 2416, September 1998.

12. Author's Addresses

Mark Allman
NASA Lewis Research Center/Sterling Software
21000 Brookpark Road
MS 54-2
Cleveland, OH 44135

EMail: mallman@lerc.nasa.gov
http://gigahertz.lerc.nasa.gov/~mallman/

Sally Floyd
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720

EMail: floyd@ee.lbl.gov

Craig Partridge
BBN Technologies
10 Moulton Street
Cambridge, MA 02138

EMail: craig@bbn.com

13. Appendix - Duplicate Segments

In the current environment (without Explicit Congestion Notification [Flo94] [RF97]), all TCPs use segment drops as indications from the network about the limits of available bandwidth. We argue here that the change to a larger initial window should not result in the sender retransmitting a large number of duplicate segments that have already been received at the receiver.

If one segment is dropped from the initial window, there are three different ways for TCP to recover: (1) Slow-starting from a window of one segment, as is done after a retransmit timeout, or after Fast Retransmit in Tahoe TCP; (2) Fast Recovery without selective acknowledgments (SACK), as is done after three duplicate ACKs in Reno TCP; and (3) Fast Recovery with SACK, for TCP where both the sender and the receiver support the SACK option [MMFR96]. In all three cases, if a single segment is dropped from the initial window, no duplicate segments (i.e., segments that have already been received at the receiver) are transmitted. Note that for a TCP sending four 512-byte segments in the initial window, a single segment drop will not require a retransmit timeout, but can be recovered from using the Fast Retransmit algorithm (unless the retransmit timer expires prematurely). In addition, a single segment dropped from an initial window of three segments might be repaired using the fast retransmit algorithm, depending on which segment is dropped and whether or not delayed ACKs are used. For example, dropping the first segment of a three segment initial window will always require waiting for a timeout. However, dropping the third segment will always allow recovery via the fast retransmit algorithm, as long as no ACKs are lost.

Next we consider scenarios where the initial window contains two to four segments, and at least two of those segments are dropped. If all segments in the initial window are dropped, then clearly no duplicate segments are retransmitted, as the receiver has not yet received any segments. (It is still a possibility that these dropped segments used scarce bandwidth on the way to their drop point; this issue was discussed in Section 5.)

When two segments are dropped from an initial window of three segments, the sender will only send a duplicate segment if the first two of the three segments were dropped, and the sender does not receive a packet with the SACK option acknowledging the third segment.

When two segments are dropped from an initial window of four segments, an examination of the six possible scenarios (which we don't go through here) shows that, depending on the position of the

dropped packets, in the absence of SACK the sender might send one duplicate segment. There are no scenarios in which the sender sends two duplicate segments.

When three segments are dropped from an initial window of four segments, then, in the absence of SACK, it is possible that one duplicate segment will be sent, depending on the position of the dropped segments.

The summary is that in the absence of SACK, there are some scenarios with multiple segment drops from the initial window where one duplicate segment will be transmitted. There are no scenarios where more that one duplicate segment will be transmitted. Our conclusion is that the number of duplicate segments transmitted as a result of a larger initial window should be small.

14. Full Copyright Statement

Network Working Group
Request for Comments: 2416
Category: Informational

T. Shepard
C. Partridge
BBN Technologies
September 1998

When TCP Starts Up With Four Packets Into Only Three Buffers

Status of this Memo

Copyright Notice

Abstract

This memo is to document a simple experiment. The experiment showed that in the case of a TCP receiver behind a 9600 bps modem link at the edge of a fast Internet where there are only 3 buffers before the modem (and the fourth packet of a four-packet start will surely be dropped), no significant degradation in performance is experienced by a TCP sending with a four-packet start when compared with a normal slow start (which starts with just one packet).

Background

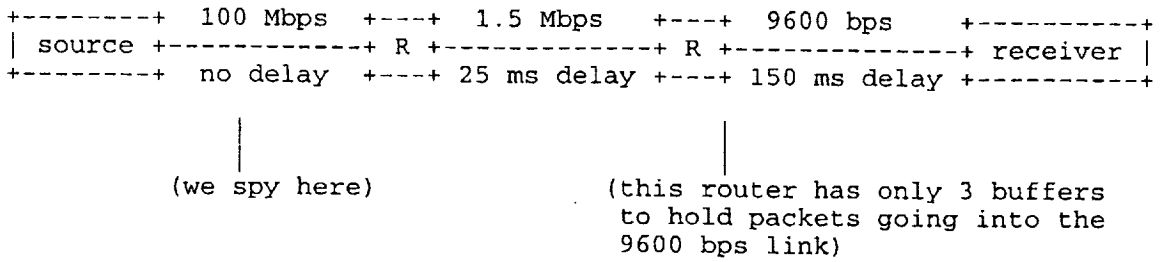
Sally Floyd has proposed that TCPs start their initial slow start by sending as many as four packets (instead of the usual one packet) as a means of getting TCP up-to-speed faster. (Slow starts instigated due to timeouts would still start with just one packet.) Starting with more than one packet might reduce the start-up latency over long-fat pipes by two round-trip times. This proposal is documented further in [1], [2], and in [3] and we assume the reader is familiar with the details of this proposal.

On the end2end-interest mailing list, concern was raised that in the (allegedly common) case where a slow modem is served by a router which only allocates three buffers per modem (one buffer being transmitted while two packets are waiting), that starting with four packets would not be good because the fourth packet is sure to be dropped.

Vern Paxson replied with the comment (among other things) that the four-packet start is no worse than what happens after two round trip times in normal slow start, hence no new problem is introduced by starting with as many as four packets. If there is a problem with a four-packet start, then the problem already exists in a normal slow-start startup after two round trip times when the slow-start algorithm will release into the net four closely spaced packets.

The experiment reported here confirmed Vern Paxson's reasoning.

Scenario and experimental setup



The scenario studied and simulated consists of three links between the source and sink. The first link is a 100 Mbps link with no delay. It connects the sender to a router. (It was included to have a means of logging the returning ACKs at the time they would be seen by the sender.) The second link is a 1.5 Mbps link with a 25 ms one-way delay. (This link was included to roughly model traversing an un-congested, intra-continental piece of the terrestrial Internet.) The third link is a 9600 bps link with a 150 ms one-way delay. It connects the edge of the net to a receiver which is behind the 9600 bps link.

The queue limits for the queues at each end of the first two links were set to 100 (a value sufficiently large that this limit was never a factor). The queue limits at each end of the 9600 bps link were set to 3 packets (which can hold at most two packets while one is being sent).

Version 1.2a2 of the the NS simulator (available from LBL) was used to simulate both one-packet and four-packet starts for each of the available TCP algorithms (tahoe, reno, sack, fack) and the conclusion reported here is independent of which TCP algorithm is used (in general, we believe). In this memo, the "tahoe" module will be used to illustrate what happens. In the 4-packet start cases, the "window-init" variable was set to 4, and the TCP implementations were modified to use the value of the window-init variable only on

connection start, but to set cwnd to 1 on other instances of a slow-start. (The tcp.cc module as shipped with ns-1.2a2 would use the window-init value in all cases.)

The packets in simulation are 1024 bytes long for purposes of

determining the time it takes to transmit them through the links. (The TCP modules included with the LBL NS simulator do not simulate the TCP sequence number mechanisms. They use just packet numbers.)

Observations are made of all packets and acknowledgements crossing the 100 Mbps no-delay link, near the sender. (All descriptions below are from this point of view.)

What happens with normal slow start

At time 0.0 packet number 1 is sent.

At time 1.222 an ack is received covering packet number 1, and packets 2 and 3 are sent.

At time 2.444 an ack is received covering packet number 2, and packets 4 and 5 are sent.

At time 3.278 an ack is received covering packet number 3, and packets 6 and 7 are sent.

At time 4.111 an ack is received covering packet number 4, and packets 8 and 9 are sent.

At time 4.944 an ack is received covering packet number 5, and packets 10 and 11 are sent.

At time 5.778 an ack is received covering packet number 6, and packets 12 and 13 are sent.

At time 6.111 a duplicate ack is recieved (covering packet number 6).

At time 7.444 another duplicate ack is received (covering packet number 6).

At time 8.278 a third duplicate ack is received (covering packet number 6) and packet number 7 is retransmitted.

(And the trace continues...)

What happens with a four-packet start

At time 0.0, packets 1, 2, 3, and 4 are sent.

At time 1.222 an ack is received covering packet number 1, and packets 5 and 6 are sent.

At time 2.055 an ack is received covering packet number 2, and packets 7 and 8 are sent.

At time 2.889 an ack is received covering packet number 3, and packets 9 and 10 are sent.

At time 3.722 a duplicate ack is received (covering packet number 3).

At time 4.555 another duplicate ack is received (covering packet number 3).

At time 5.389 a third duplicate ack is received (covering packet number 3) and packet number 4 is retransmitted.

(And the trace continues...)

Discussion

At the point left off in the two traces above, the two different systems are in almost identical states. The two traces from that point on are almost the same, modulo a shift in time of (8.278 - 5.389) = 2.889 seconds and a shift of three packets. If the normal TCP (with the one-packet start) will deliver packet N at time T, then the TCP with the four-packet start will deliver packet N - 3 at time T - 2.889 (seconds).

Note that the time to send three 1024-byte TCP segments through a 9600 bps modem is 2.66 seconds. So at what time does the four-packet-start TCP deliver packet N? At time T - 2.889 + 2.66 = T - 0.229 in most cases, and in some cases earlier, in some cases later, because different packets (by number) experience loss in the two traces.

Thus the four-packet-start TCP is in some sense 0.229 seconds (or about one fifth of a packet) ahead of where the one-packet-start TCP would be. (This is due to the extra time the modem sits idle while waiting for the dally timer to go off in the receiver in the case of the one-packet-start TCP.)

The states of the two systems are not exactly identical. They differ slightly in the round-trip-time estimators because the behavior at the start is not identical. (The observed round trip times may differ by a small amount due to dally timers and due to that the one-packet start experiences more round trip times before the first loss.) In the cases where a retransmit timer did later go off, the additional

difference in timing was much smaller than the 0.229 second difference discribed above.

Conclusion

In this particular case, the four-packet start is not harmful.

Non-conclusions, opinions, and future work

A four-packet start would be very helpful in situations where a long-delay link is involved (as it would reduce transfer times for moderately-sized transfers by as much as two round-trip times). But it remains (in the authors' opinions at this time) an open question whether or not the four-packet start would be safe for the network.

It would be nice to see if this result could be duplicated with real

TCPs, real modems, and real three-buffer limits.

Security Considerations

This document discusses a simulation study of the effects of a proposed change to TCP. Consequently, there are no security considerations directly related to the document. There are also no known security considerations associated with the proposed change.

References

1. S. Floyd, Increasing TCP's Initial Window (January 29, 1997). URL ftp://ftp.ee.lbl.gov/papers/draft.jan29.

2. S. Floyd and M. Allman, Increasing TCP's Initial Window (July, 1997). URL http://gigahertz.lerc.nasa.gov/~mallman/share/draft-ss.txt

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

Authors' Addresses

Tim Shepard
BBN Technologies
10 Moulton Street
Cambridge, MA 02138

EMail: shep@alum.mit.edu

Craig Partridge
BBN Technologies
10 Moulton Street
Cambridge, MA 02138

EMail: craig@bbn.com

IP Switching, Tag Switching, and Routing

Dr. Craig Partridge
Principal Scientist

Overview

- What's a router do?
- Router Technology in 1995
- IP switching
- Multiprotocol Label Switching
- Router Technology in 1997
- IP Switching: Fad or Our Future?

What's a Router Do?

- Routers are the core technology of the Internet
- They connect different types of networks together into one virtual network
  - rules for encapsulating the Internet Protocol in every media type
    - Ethernet, ATM, leased lines (PPP)
  - router extracts IP datagram from inbound packets, and from information in the IP header, decides what outbound network the datagram should be sent on, and encapsulates and sends on outbound network
- They protect networks from each other
  - each router evaluates every datagram for sanity
  - traffic filters protect from Martians, broadcast storms, hackers

Router Technology in 1995

- IP switching invented in 1995
  - what did routers look like then?
- Router technology had not changed in 10 years
  - a simple shared bus interconnecting the interfaces
  - a master processor
    - maintained routing table
    - exchanged routing information with other routers
  - slave processors on interfaces
    - small cache of recent routes
- Performance was bad and future looked worse
  - shared bus at its limits
  - delay between slave processor and master an issue
  - took a long time for master to find entry in routing table
  - performance demands growing faster than Moore's Law

Route Lookup in 1995

- Slave processor looks for route in its cache
  - if a hit, go at about 500 KPPS
  - if a miss, go to master processor (about 2-4 packet times to get there)
- Master processor looks for route in a modified Patricia tree using best prefix match
  - 1 to 30 memory accesses (up to 3 packet times)
  - alongside all the other work master processor did
- Clear scalability problems
  - slave-master getting farther apart as components get faster
  - Patricia on average takes longer the more routes you have

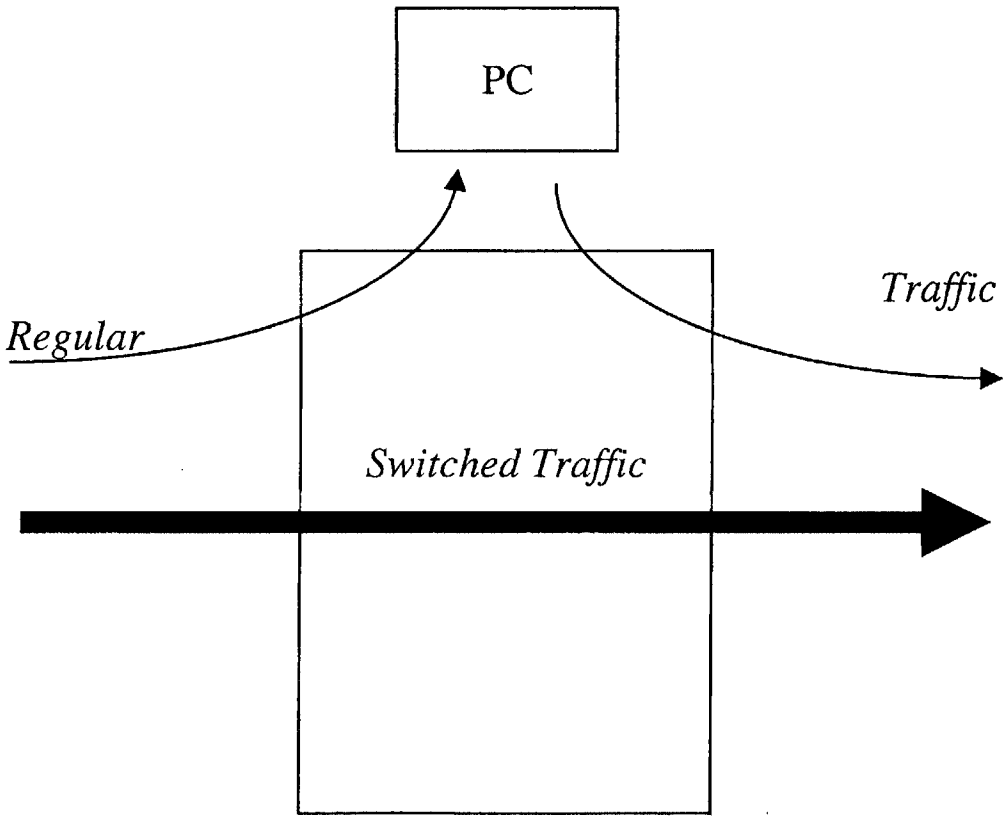
IP Switching

- Developed in 1995 by Ipsilon
- Idea: recycle ATM switches
  - local ATM headed for a fall (cheap switches)
  - quick way to replace bus-based routers with switch-based routers
- Problem: route lookup
  - didn't want to change innards of ATM switch
    - goal was to recycle
    - some way to map datagrams to virtual circuits needed
  - adding sufficiently fast routing system to side of switch would have increased cost 5x or more
    - wanted to just attach a PC to switch
    - or better, use the PC already there, freed of the burden of conforming to ATM Forum standards

How IP Switching Works

- When IP switch boots
  - all traffic goes to PC
- PC runs a pattern matcher on IP traffic
  - looks for stream of datagrams going to the same place
  - historically, large fraction (c. 90%) of traffic is actually carried by only a few flows at any given time
- Once a pattern found, PC creates ATM VC for the flow of datagrams
  - to upstream and downstream IP switches
- Once VCs established, datagrams flow through switch
  - using ATM VC mechanisms
  - no IP processing done at all
  - runs at switch speed, not PC speed

IP Switching



Strengths of IP Switching

- Leverage
  - making the common case very fast
- Using a switch
- Easy to support on most switch platforms
  - many gigabit Ethernet vendors use it

Weaknesses of IP Switching

- What if traffic patterns change and pattern matcher stops working?
  - the IP switch starts running like a PC router
  - your customer gets fired (bad for repeat sales)
- What if you run out of VCs?
  - akin to slave processors with too small a cache
  - similar effects to pattern match failure
- Security
  - once a pattern match is made, there's no checking of traffic
  - limits IP switching to domains of trust
- Fault detection
  - if router crashes and loses VC info, error recovery can be hard
- Only works for ATM

Tag Switching

- By mid-1996, Cisco had a marketing problem
  - arguably, IP switching more cost effective for speed
  - defect-for-defect marketing arguments
  - every switch vendor was offering IP switching
- Result was tag switching
- Differences from IP switching
  - tag was a hint you could ignore
    - enhanced security
    - greater robustness against failure
  - tags for all media
    - not just ATM
- Led to creation of Multiprotocol Label Switching
  - though tag switching work continues

Multiprotocol Label Switching

- IETF effort to generalize tag/IP switching
- Datagrams have label stacks
  - hierarchy of labels
  - allows for trunking type applications and better flow label management
- Envisions we may use labels for QoS flows
  - i.e. create custom routes, associate labels with them
- Media and protocol independent
  - not just IP
- But can't ignore label
  - consequence of doing specialized routing

Router Technology in 1997

- Router technology has evolved too
- All major routers now use switched backplanes
- Master-slave processors replaced
  - control processor to manage router
  - forwarding engines to make forwarding decisions
- ASIC-based forwarding engines
- Route lookup algorithms improved
  - Washington University algorithm
  - Luleå algorithm

Route Lookup in 1997

- Forwarding engine looks for route in its complete copy of routing table
  - control processor keeps table up to date
- Forwarding engine uses one of a few algorithms
  - Patricia (but not much)
  - Washington University algorithm (5 memory access)
  - Luleå algorithm (1-6 accesses)
  - CAM (1 access but small table size)

Implications of New Router Technology

- For small routing tables, vendors putting routing in a chip
  - great for corporate backbones
  - cost < $1K for forwarding engine
  - drops right into gigabit Ethernet switch
  - 6 to 12 MPPS today
- For large routing tables, memory access costs are reduced
  - easier to go very fast
  - 6 to 8 MPPS today
  - about $5K per forwarding engine
- Switched backplane gives speed

IP Switching: Fad or Our Future?

- There's no longer a performance gap
  - routers can keep up with need
  - in corporate environments, ASIC-based forwarding engines are cheaper and more robust than IP switching
- Forwarding lookups much less of a bottleneck
  - reduces the need for hints
- So we'll only do IP switching if it gives a service benefit
  - Quality of Service tagging?

Does TCP Work over Satellite Links or Not?

Dr. Craig Partridge
BBN Technologies

The Puzzle

- TCP was designed to work over satellites
  - the goal of the research project that created TCP was to link SATNET and ARPANET
- TCP's theoretical maximum data rate is 15 Gb/s
  - faster than any satellite link
- So why do people feel TCP doesn't work over satellites?

The Reasons

- Bad reasons
  - out of date implementations
  - misconfigured TCPs
  - poor testing technique
- Good reasons
  - high bandwidth
  - TCP startup delays

Out of Date Implementations

- TCP, as specified in 1981, had limitations
  - max data rate of 1 Mb/s over GEO link
  - 286 Mb/s overall max
- Limitations repaired in early 1990s
  - PAWS and big windows
  - max data rate now 15 Gb/s over GEO
  - 8 Tb/s overall max
  - make sure you're up to date!

Misconfigured TCPs

- An up-to-date implementation doesn't help if you don't configure it
- Many TCPs shipped with 64KB maximum window size
  - 64KB/250ms is 2 Mb/s
- Must turn on PAWS and large windows and set default window to be large!

Bad Testing

- Lots of people test performance by taking TCP, out of the box, and FTPing a megabyte of data
- That's stupid
  - TCP may be misconfigured
  - 1 MB is far too small for anything but a LAN
- Configure the TCP, then transfer a gigabyte

High Bandwidth

- Moving from 56 Kb/s links to megabit links has implications
  - more data in flight (more sender effort to fill the pipe)
  - error rates must go down proportionately
  - big transfers get most of the benefit

Big Transfers Get Most of the Benefit

- Faster isn't really faster
  - speed of light says a 1 bit pulse takes the same time it always did
  - faster really means bits are thinner on the wire
- Big transfers win; small transfers don't
  - big transfers get all their bits on the line sooner
- Web transfers are small...
  - transfer time dominated by transmission delay

TCP Startup Delays

- On startup, TCP probes a link to learn how much capacity is available
  - goal is to avoid overloading network; and
  - fairly sharing with existing connections
- Startup time depends on delay and bandwidth
  - on a GEO at 155 Mb/s, it takes 11+ seconds
  - first 20 GB are sent during this probe stage

How Do We Go Forward?

- Recognize that terrestrial guys have the same problem with startup
- So look for general solutions
  - Hoes' algorithm
  - pacing
- Avoid link-specific algorithms
  - spoofing

NASA/CR—1999-209167

An Overview of TCP/IP Performance

Craig Partridge

Outline

- A brief overview of how TCP/IP works
- IP performance issues
  - TTL and fragmentation
- TCP performance issues
  - Large Windows, PAWS
  - Slow Start, Congestion Avoidance

Overview - IP

- IP is the Internet Protocol
- Combines multiple distinct networks into one virtual network
  - designed in the 1970s to hook ARPANET to satellite and packet radio networks
- Extremely simple
  - each datagram self contained
  - 20 byte header on each datagram

Overview - IP cont.

- IP service is very simple
  - each router does its best to forward the datagram on to its destination
  - but, datagrams may be
    - lost
    - misrouted
    - damaged
    - duplicated
- Result is a very robust but unreliable service

Overview - TCP

- TCP's job is to make IP service reliable
- TCP creates a reliable byte stream over the IP layer
  - another protocol, UDP, gives direct access to IP's unreliable service
- Most performance complexity is accordingly in TCP

Overview - TCP Connections

- TCP creates connections between applications
  - two conversations between different applications on the same two hosts are two TCP connections
  - applications identified by *ports*
- All TCP flow control is per application

Overview - TCP Reliability

- TCP achieves reliability using three mechanisms
  - Sequence Numbers
    - every byte is numbered
  - Window size
    - receiver limits data in flight to avoid overruns
  - Positive Acknowledgments
    - receiver indicates what data it has received
    - acks are cumulative

Overview - TCP Startup

- At startup, a TCP connection negotiates
  - sequence numbers
  - options (PAWS, Large Win, MSS)
- It then starts the slow start algorithm
  - slow start attempts to assess current congestion (if any) in the path
  - scales transmission rate to available bandwidth

TCP Steady State

- After slow start, TCP transmits steadily, in the absence of loss
- Transmission speed governed by
  - receiver window size
  - bandwidth of link

Overview - TCP Congestion

- If TCP determines there is congestion it does congestion avoidance
  - congestion window
- Three parts
  - react to congestion by setting congestion window to 1/2 current value
  - do slow start up to new estimate
  - linearly increase congestion window (probe) past current estimate

Performance

- We now shift our attention from basic mechanisms to performance
- IP performance first
  - not a big deal
- Then TCP performance
  - where most of the issues lie

Performance - IP

- IP has two performance limiting features
- IP Time To Live
  - ensures datagrams don't live forever
  - places requirements on upper layers
- IP Fragmentation
  - copes with dissimilar MTUs
  - limit on number of fragments alive at one time

Performance - TCP Sequence #'s

- TCP has a richer set of performance limits
- First one is the sequence space
- IP TTL says data may live for a few minutes in the network
- At high speeds, TCP wraps its entire sequence space in a few seconds

Performance - PAWS

- The solution for the sequence space problem is PAWS
  - Protection Against Wrapped Sequence numbers
- Adds a timestamp to every TCP segment
- Downward compatible
  - negotiated at connection startup

Performance - TCP Window Size

- The max size of the TCP receiver's window limits throughput
  - 64KB per round-trip
- Solution is to extend the window size
  - Window Scale option
  - negotiated like PAWS at startup

Performance - Slow Start

- Mentioned that TCP does slow start
- Sends 50% more data every round-trip until hits receiver window or loss
  - takes time proportional to RTT $\log_{1.5}$ BW
  - for high bandwidth paths or high RTT paths, that's a problem
  - Most WWW connections never leave slow start!

Performance - Cong. Avoidance

- TCP uses congestion avoidance after initial slow start
- Performance similar to slow start but takes longer to speed up
- TCP guesses there is congestion if it sees loss
  - conservative assumption
  - a problem on long-delay moderate error links

TCP Enhancements for Web Performance

Internetworking Research Dept
BBN Technologies

The Challenge

- Most commercial TCP implementations do not include key performance features developed in the past 6 years
- Web and other apps suffer
  - web servers cannot serve as many clients
  - data transmission slower than it needs to be

The Opportunity

- Internetworking Research has several of the worlds TCP experts
  - 70+ years of TCP experience
- Linux is a popular web OS
  - 30% of the market (MS's major competitor)
  - software is free and open
- Upgrade Linux TCP and raise the bar for the entire web market

Details...

- What improvements would we make?
- How long will it take?

Improvements

- Better slow start
  - get TCP sending full rate more quickl)
- Forward Acknowledgements
  - improve TCP recover from loss
- Dynamic window buffer management
  - improve TCP throughput AND
  - sharply increase number of clients a server can service

More Improvements

- Security fixes
  - make TCP less vulnerable to attacks like SYN attacks
- Pacing
  - reduce unnecessary packet loss
  - improve TCP throughput
- Code tuning
  - reduce server load by 25% or more

How Long Would This Take?

- Initial distribution, 9 months
  - put it out in the community and let folks use it
- Complete project, 18 months
  - we have to support our distribution until users are sure it is OK
  - there will be bugs... TCP is *very* complex

TCP/IP Performance over Satellite Links

Craig Partridge and Timothy J. Shepard
BBN Technologies

Reprinted from
IEEE NETWORK
September/October 1997

TCP/IP Performance over Satellite Links

Craig Partridge and Timothy J. Shepard
BBN Technologies

Abstract

Achieving high data rates using TCP/IP over satellite networks can be difficult. This article explains some of the reasons TCP/IP has difficulty with satellite links. We present solutions to some problems, and describe the state of the research on some of the unsolved problems.

Overview of TCP/IP impact performance. We then present issues specific to satellites aninformal) about how well TCP/IP performs over satellite links. Some reports indicate TCP/IP throughput is poor. Others report that TCP/IP throughput is quite good. It is very difficult to determine which reports deserve more credence.

This article tries to clarify the situation. Our approach is to first discuss TCP/IP performance analytically, indicating what features of TCP/IP impact performance. We then present issues specific to satellites and their solutions, if known.

An Overview of TCP and IP Performance

TCP/IP is a surprising complex protocol suite and more than one person has written an entire book on the details of its operation.[1] Rather than try to summarize all of TCP/IP, our goal in this section is to present those aspects of TCP/IP that most directly affect TCP/IP throughput. More specifically, we will focus on a particular aspect of throughput, namely the effective transmission rate of valid data (sometimes called goodput) that a TCP/IP connection can achieve.

IP Throughput Issues

IP (the Internet Protocol) is the network layer protocol in the TCP/IP protocol suite. IP's function is to provide a protocol to integrate heterogeneous networks together. In brief, a media-specific way to encapsulate IP datagrams is defined for each media (e.g., satellite, Ethernet, or Asynchronous Transfer Mode). Devices called *routers* move IP datagrams between the different media and their encapsulations. Routers pass IP datagrams between different media according to routing information in the IP datagram. This mesh of different media interconnected by routers forms an IP *internet*, in which all hosts on the integrated mesh can communicate with each other using IP.[2]

The actual service IP implements is unreliable datagram delivery. IP simply promises to make a reasonable effort to deliver every datagram to its destination. However IP is free to occasionally lose datagrams, deliver datagrams with errors in them, and duplicate and reorder datagrams.

Because IP provides such a simple service, one might assume that IP places no limits on throughput. Broadly speaking, this assumption is correct. IP places no constraints on how fast a system can generate or receive datagrams. A system transmits IP datagrams as fast as it can generate them. However, IP does have two features that can affect throughput: the IP Time to Live and IP Fragmentation.

IP Time To Live — In certain situations, IP datagrams may loop among a set of routers. These loops are sometimes transient (a datagram may loop for a while and then proceed to its destination) or long-lived. To protect against datagrams circulating semipermanently, IP places a limit on how long a datagram may live in the network.

The limit is imposed by a Time To Live (TTL) field in the IP datagram. The field is decremented at least once at every router the datagram encounters and when the TTL reaches zero, the datagram is discarded.

Originally, the IP specification also required that the TTL also be decremented at least once per second. Since the TTL field is 8-bits wide, this means a datagram could live for approximately 4.25 minutes. In practice, the injunction to decrement the TTL once a second is ignored, but, perversely, specifications for higher layer protocols like TCP usually assume that the maximum time a datagram can live in the network is only two minutes.

This work was funded by NASA Lewis Research Center.

[1] *Two very good books on the subject are [1] and [2].*

[2] *The term internet is a generic word for a group of interconnected networks. The Internet is the global IP internet. Recently the term intranet has evolved from its original meaning (an adjective meaning on a single physical network [3]) into a popular way to describe an IP internet entirely within an organization.*

The significance of the maximum datagram lifetime is that it means higher layer protocols must be careful not to send two similar datagrams (in particular, two datagrams which could be confused for each other) within a few minutes of each other. This limitation is particularly important for sequence numbers. If a higher layer protocol numbers its datagrams, it must ensure that it does not generate two datagrams with the same sequence number within a few minutes of each other, lest IP deliver the second datagram first and confuse the receiver. We discuss this issue more in the next section when we discuss TCP sequence space issues.

IP Fragmentation — Different network media have different limits on the maximum datagram size. This limit is typically referred to as the Maximum Transmission Unit (MTU). When a router is moving a datagram from one media to another, it may discover that the datagram, which was of legal size on the inbound media, is too big for the outbound media. To get around this problem, IP supports fragmentation and reassembly, in which a router can break the datagram up into smaller datagrams to fit on the outbound media. The smaller datagrams are reassembled into the original larger datagram at the destination (not the intermediate hops).

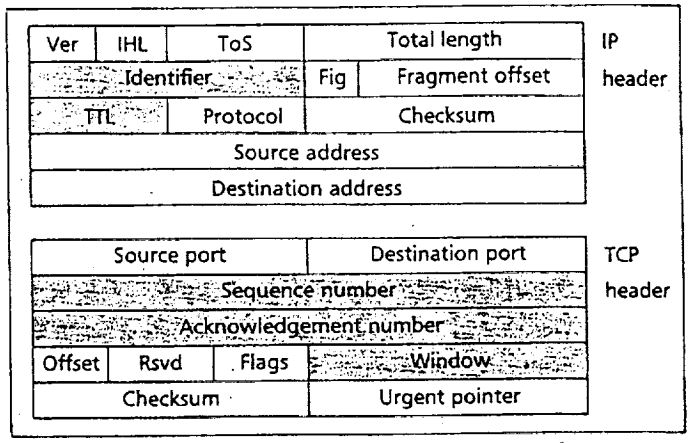
Fragments are identified using a fragment offset field (which indicates the offset of the fragment from the start of the original datagram). Datagrams are uniquely identified by their source, destination, higher layer protocol type, and a 16-bit IP identifier (which must be unique when combined with the source, destination and protocol type).

Observe that there's a clear link between the TTL field and the IP identifier (first identified by [4]). An IP source must ensure that it does not send two datagrams with the same IP identifier to the same destination, using the same protocol within a maximum datagram lifetime, or fragments of two different datagrams may be incorrectly combined. Since the IP identifier is only 16 bits, if the maximum datagram lifetime is two minutes, we are limited to a transmission rate of only 546 datagrams per second. That's clearly not fast enough. The maximum IP datagram size is 64 KB, so 546 datagrams is, at best, a bit less than 300 Mb/s.

The problem of worrying about IP identifier consumption has largely been solved by the development of MTU Discovery a technique for IP sources to discover the MTU of the path to a destination [5]. MTU Discovery is a mechanism that allows hosts to determine the MTU of a path reliably. The existence of MTU discovery allows hosts to set the Don't Fragment (DF) bit in the IP header, to prohibit fragmentation, because the hosts will learn through MTU discovery if their datagrams are too big. Sources that set the DF bit need not worry about the possibility of having two identifiers active at the same time. Systems that do not implement MTU discovery (and thus cannot set the DF bit) need to be careful about this problem.

TCP Throughput Issues

The Transmission Control Protocol (TCP) is the primary transport protocol in the TCP/IP protocol suite. It implements a reliable byte stream over the unreliable datagram service provided by IP. As part of implementing the reliable service, TCP is also responsible for flow and congestion control: ensuring that data is transmitted at a rate consistent with the capacities of both the receiver and the intermediate links in the network path. Since there may be multiple TCP connections active in a link, TCP is also responsible for ensuring that a link's capacity is responsibly shared among the connections using it. As a result, most throughput issues are rooted in TCP.



■ Figure 1. *TCP and IP header fields that affect throughput.*

This section examines the major features of TCP that affect performance. Many of these performance issues have been discovered over the past few years as link transmission speeds have increased and so called high *delay-bandwidth* paths[3] (paths where the product of the path delay and available path bandwidth is big) have become common. To begin to illustrate the challenge, consider that in the 1970s when TCP was being developed, the typical long link was a 56 kb/s circuit across the United States, with a delay-bandwidth product of approximately 0.250 x 56,000 bits or 1.8 KB, while today's Internet contains 2.4 Gb/s circuits crossing the US, which boast a delay-bandwidth product of 75 MB.

Throughput Expectations — Before presenting the performance issues for TCP, it is worth talking briefly about throughput goals.

TCP throughput determines how fast most applications can move data across a network. Application protocols such as HTTP (the World Wide Web protocol), and the File Transfer Protocol (FTP), rely on TCP to carry their data. So TCP performance directly impacts application performance.

While there are no formal TCP performance standards, TCP experts generally expect that, when sending large datagrams (to minimize the overhead of the TCP and IP headers), a TCP connection should be able to fill the available bandwidth of a path and to share the bandwidth with other users. If a link is otherwise idle, a TCP connection is expected to be able to fill it. If a link is shared with three other users, we expect each TCP to get a reasonable share of the bandwidth.

These expectations reflect a mix of practical concerns. When users of TCP acquire faster data lines, they expect their TCP transfers to run faster. And users acquire faster lines for different reasons. Some need faster lines because as their aggregate traffic has increased, they have more applications that need network access. Others have a particular application that requires more bandwidth. The requirement that TCP share a link effectively reflects the needs of aggregation; all users of a faster link should see improvement. The requirement that TCP fill an otherwise idle link reflects the needs of more specialized applications.

TCP Sequence Numbers — TCP keeps track of all data in transit by assigning each byte a unique sequence number. The receiver acknowledges received data by sending an acknowl-

[3] *To avoid confusion, we note that the data networking community, unlike some engineering communities, uses the term bandwidth interchangeably with bitrate.*

edgment which indicates that the receiver has received all data up to a particular byte number.

TCP allocates its sequence numbers from a 32-bit wraparound sequence space. To ensure that a given sequence number uniquely identifies a particular byte, TCP requires that no two bytes with the same sequence number be active in the network at the same time. Recall the early discussion of IP datagram lifetime indicated a datagram was assumed to live for up to two minutes. Thus when TCP sends a byte in an IP datagram, the sequence number of that byte cannot be reused for two minutes. Unfortunately, a 32-bit sequence space spread over two minutes gives a maximum data rate of only 286 Mb/s.

To fix this problem, the Internet End-to-End Research Group devised a set of TCP options and algorithms to extend the sequence space. These changes were adopted by the Internet Engineering Task Force (IETF) and are now part of the TCP standard. The option is a timestamp option [6] which concatenates a timestamp to the 32-bit sequence number. Comparing timestamps using an algorithm called PAWS (Protection Against Wrapped Sequence numbers) makes it possible to distinguish between two identical sequence numbers sent less than two minutes apart.

Depending on the actual granularity of the timestamp (the IETF recommends between 1 second and 1 millisecond), this extension is sufficient for link speeds of between 8 Gb/s and 8 Tb/s (terabits per second).

TCP Transmission Window — The purpose of the transmission window is to allow the receiving TCP to control how much data is being sent to it at any given time. The receiver advertises a window size to the sender. The window measures, in bytes, the amount of unacknowledged data that the sender can have in transit to the receiver. The distinction between the sequence numbers and the window is that sequence numbers are designed to allow the sender to keep track of the data in flight, while the window's purpose is to allow the receiver to control the rate at which it receives data.

Obviously, if a receiver advertises a small window (due, perhaps, to buffer limitations) it is impossible for TCP to achieve high transmission rates. And many implementations do not offer a very large window size (a few kilobytes is typical).

However, there is a more serious problem. The standard TCP window size cannot exceed 64 KB, because the field in the TCP header used to advertise the window is only 16 bits wide. This limits the TCP effective bandwidth to 2^{16} bytes divided by the round-trip time of the path [7]. For long delay links, such as those through satellites with a geosynchronous orbit (GEO), this limit gives a maximum data rate of just under 1 Mb/s.

As part of the changes to add timestamps to the sequence numbers, the End-To-End Research Group and IETF also enhanced TCP to negotiate a window scaling option. The option multiplies the value in the window field by a constant. The effect is that the window can only be adjusted in units of the multiplier. So if the multiplier is 4, an increase of 1 in the advertised window means the receiver is opening the window by 4 bytes.

The window size is limited by the sequence space (the window must be no larger than one half of the sequence space so that it is unambiguously clear that a byte is inside or outside the window). So the maximum multiplier permitted is 2^{14}. This means the maximum window size is 2^{30} and the maximum date rate over a GEO satellite link is approximately 15 Gb/s. Given we have achieved Tb/s data rates in terrestrial fiber, this value is depressingly small, but in the absence of a major change to the TCP header format it is not clear how to fix the problem.

Slow Start — When a TCP connection starts up, the TCP specification requires the connection to be conservative and assume that the available bandwidth to the receiver is small. TCP is supposed to use an algorithm called *slow start* [8], to probe the path to learn how much bandwidth is available.

The slow start algorithm is quite simple and based on data sent per round trip. At the start, the sending TCP sends one TCP segment (datagram) and waits for an acknowledgment. When it gets the acknowledgment, it sends two segments. Many TCPs acknowledge every other segment they receive,[4] so the slow start algorithm effectively sends 50 percent more data every round trip. It continues this process (sending 50 percent more data each round trip) until a segment is lost. This loss is interpreted as indicating congestion and the connection scales back to a more conservative approach (described in the next section) for probing bandwidth for the rest of the connection.

There are two problems with the slow start algorithm on high-speed networks. First, the probing algorithm can take a long time to get up to speed. The time required to get up to speed is $R(1 + \log_{1.5}(DB/l))$, where R is the round-trip time, DB is the delay-bandwidth product and l is the average segment length. If we are trying to fill a pipe with a single TCP connection (and, if the TCP connection is the sole user of the link, filling the link is considered the canonical goal), then DB should be the product of the bandwidth available to the connection and the round-trip time.

An important point is that as the bandwidth goes up or round-trip time increases, or both, this startup time can be quite long. For instance, on a Gb/s GEO satellite link with a 0.5 second round-trip time, it takes 29 round-trip times or 14.5 seconds to finish startup. If the link is otherwise idle, during that period most of the link bandwidth will be unused (wasted).

Even worse is that, in many cases, the entire transfer will complete before the slow start algorithm has finished. The user will never experience the full link bandwidth. All the transfer time will be spent in slow start. This problem is particularly severe for HTTP (the World Wide Web protocol), which is notorious for starting a new TCP connection for every item on a page.[5] This poor protocol design is a (major) reason Web performance on the Internet is perceived as poor: the Web protocols never let TCP get up to full speed.

Currently, the IETF is in the early stages of considering a change to allow TCPs to transmit more than one segment (the current proposal permits between two and four segments) at the beginning of the initial slow start. If there is capacity in the path, this change will reduce the slow start by up to three round-trip times. This change mostly benefits shorter transfers that never get out of slow start.

The second problem is interpreting loss as indicating congestion. TCP has no easy way to distinguish losses due to transmission errors from losses due to congestion, so it makes the conservative assumption that all losses are due to congestion. However, as was shown in an unpublished experiment at MIT, given the loss of a TCP segment early in the slow start process, TCP will then set its initial estimate of the available bandwidth far too low. And since the probing algorithm becomes linear rather than exponential after the initial estimate is set, the time to get to full transmission rate can be very long. On a gigabit GEO link, it could be several hours!

---

[4] TCP acknowledgments are cumulative, so one acknowledgment can acknowledge multiple segments. Sending one acknowledgment for every two segments reduces the return path bandwidth consumed by the acknowledgments.

[5] A problem now being alleviated by the HTTP 1.1 specification [9].

	1.5 Mb/s			45 Mb/s			155 Mb/s		
	LAN	LEO	GEO	LAN	LEO	GEO	LAN	LEO	GEO
Requires PAWS	No	No	No	No	No	No	Yes	Yes	Yes
Requires large windows	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Slow start time	0.01s	1.8s	5.6s	0.2s	3.5s	9.8s	0.9s	4.1s	11.3s
Slow start data (in bytes)	1,760	76,600	197,870	115,900	2,405,000	6,003,000	4,123,814	8,292,000	20,650,000

■ Table 1. *Summary of satellite and TCP interactions.*

Congestion Avoidance — Throughout a TCP connection, TCP runs a congestion avoidance algorithm which is similar to the slow start algorithm and was described in the same paper by Jacobson [8]. Essentially, the sending TCP maintains a congestion window, an estimate of the actual available bandwidth of the path to the receiver. This estimate is set initially by the slow start at the start of the connection. Then the estimate is varied up and down during the life of the connection based on indications of congestion (or the absence thereof). In general, congestion is assumed to be indicated by loss of one or more datagrams.

The basic estimation algorithm is as follows. Every round trip, the sending TCP increases its estimate of the available bandwidth by one maximum-sized segment. Whenever the sender either finds a segment was lost (conservatively assumed to be due to congestion) or receives an indication from the network (e.g., an ICMP Source Quench) that congestion exists, the sender halves its estimate of the available bandwidth. The sender then resumes the one segment per round-trip probing algorithm. (In certain, extreme, loss situations, the sender will do a slow start).

Like the slow start algorithm, the major issue with this algorithm is that over high-delay-bandwidth links, a datagram lost to transmission error will trigger a low estimate of the available bandwidth, and the linear probing algorithm will take a long time to recover.

Another issue is that the rate of improvement under congestion avoidance is a function of the delay-bandwidth product. Basically congestion avoidance allows a sender to increase its window by one segment, for every round-trip time's worth of data sent. In other words, congestion avoidance increases the transmission rate by $1/DB$ each round trip [10, 11].

Selective Acknowledgments — Recently the Internet Engineering Task Force has approved an extension to TCP called Selective Acknowledgments (SACKs) [12]. SACKs make it possible for TCP to acknowledge data received out of order. Previously TCP had only been able to acknowledge data received in order.

SACKs have two major benefits. First, they improve the efficiency of TCP retransmissions by reducing the retransmission period. Historically, TCP has used a retransmission algorithm that emulates selective-repeat ARQ using the information provided by in-order acknowledgments. This algorithm works, but takes roughly one round-trip time per lost segment to recover. SACK allows a TCP to retransmit multiple missing segments in a round trip. Second, and more importantly, work by Mathis and Mahdavi [12] has shown that with SACKs a TCP can better evaluate the available path bandwidth in a period of successive losses and avoid doing a slow start.

Inter-Relations — It is important to keep in mind that all the various TCP mechanisms are interrelated, especially when applied to problems of high performance. If the sequence space and window size are not large enough, no improvement to congestion windows will help, since TCP cannot go fast enough anyway. Also, if the receiver chooses a small window size, it takes precedence over the congestion window, and can limit throughput.

More broadly, tinkering with TCP algorithms tends to show odd interrelations. For instance, the individual TCP Vegas performance improvements [13, 14] were shown to work only when applied together applying only some of the changes actually degraded performance. And there are also known TCP syndromes where the congestion window gets misestimated, causing the estimation algorithm to briefly thrash before converging on a congestion window. (The best known is a case where a router has too little buffer space, causing bursts of datagrams to be lost even though there is link capacity to carry all the datagrams).

Satellites and TCP/IP Throughput

For the rest of this article we apply the general discussion of the previous section to the specific problem of achieving high throughput over satellite links. First, we point out the need to implement the extensions to the TCP sequence space and window size. Then we discuss the relationship between slow start and performance over satellite links and some possible solutions.

Currently satellites offer a range of channel bandwidths, from the very small (a compressed phone circuit of a few kb/s) to the very large (the Advanced Communications and Telecommunications Satellite with 622-Mb/s circuits). They also have a range of delays, from relatively small delays of low earth orbit (LEO) satellites to the much larger delays of GEO satellites. Our concern is making TCP/IP work well over those ranges.

General Performance

Many of the problems described in the previous section on TCP/IP performance were ones that became acute only over high-delay-bandwidth paths. One of the first things to note is that all but the slowest satellite links are, by definition, high-delay-bandwidth paths, because the transmission delays to and from the satellite from the Earth's surface are large.

Table 1 illustrates for a range of common bandwidths, when the TCP enhancements of PAWS and large windows are required to fully utilize the bandwidth on a LAN link with 5 ms one-way delay, a LEO link (100 ms one-way) and GEO (250 ms one-way) link, for a range of link speeds. We also indicate how long slow start takes to get to full link speed, assuming 1 KB datagrams (a typical size) are transmitted and how much data is transferred during the slow start phase.

The table highlights some key challenges for satellites (and also for transcontinental terrestrial links, which have delays similar to LEO satellite links). One simply cannot get a TCP/IP implementation to perform well at higher speeds unless it supports large windows, and at speeds past about 100 Mb/s, PAWS. Thus anyone who has not had their TCP/IP software upgraded with PAWS and large windows will not be able to achieve high performance over a satellite link.

Buffer Size in segments	p	Link Rates		
		1.5 Mb/s	45 Mb/s	155 Mb/s
10	4	3×10^6	9×10^7	3.1×10^8
100	13	9.8×10^6	2.9×10^8	1×10^9
1000	44	3.3×10^7	9.9×10^8	3.4×10^9

■ Table 2. *Approximate number of bits sent over GEO link during congestion avoidance.*

Slow Start Revisited

Another point of Table 1 is that the initial slow start period can be quite long and involve large quantities of data. Particularly striking is the column for 155 Mb/s transfers. Between 8 and 21 megabytes of data are sent over a satellite link during slow start at 155 Mb/s. Even at 1.5 Mb/s a GEO link must carry nearly 200 KB before slow start ends. Few data transfers on the Internet are megabytes long. Many are a few kilobytes. All of which says that satellite links will look slow and inefficient for the average data transmission. Interestingly enough, long-distance terrestrial links will also look slow. Their delays are comparable to those of LEO links.

Furthermore, observe that the table helps explain the variation in reported TCP goodput over satellite links. Short data transfers will never achieve full link rate. In many cases, a gigabyte file transfer or larger is probably required to ensure throughput figures are not heavily influenced by slow start.

Obviously some sort of solution to reduce the slow start transient would be desirable. But finding a solution isn't easy.

One obvious solution is to dispense with slow start and just start sending as fast as one can until data is dropped, and then slow down. This approach is known to be disasterous. Indeed, slow start was invented in an environment in which TCP implementations behaved this way and were driving the Internet into congestion collapse. As one example of how this scheme goes wrong, consider a Gb/s capable TCP launching several 100s of megabits of data over a path that turns out to have only 9.6 kb/s of bandwidth. There's a tremendous bandwidth mismatch which will cause datagrams to be discarded or suffer long queuing delays.

As this example illustrates, one of the important problems is that a sending TCP has no idea, when it starts sending, how much bandwidth a particular transmission path has. In the absence of knowledge, a TCP should be conservative. And slow start is conservative — it starts by sending just one datagram in the first round trip.

However, it is clear that somehow we need to be able to give TCP more information about the path if we are to avoid the peril of having TCP chronically spend its time in slow start. One nice aspect of this problem is that it is not specific to satellites. Terrestrial lines need a solution too, and thus if we can find a general solution that works for both satellites and terrestrial lines, everyone will be happy to adopt it.

Improving Slow Start — If the TCP had more information about the path, it could presumably skip at least some of the slow start process possibly by starting the slow start at a somewhat higher rate than one datagram. (The IETF initiative to use a slightly larger beginning transmission size for the initial slow start is a step in this direction). But actually learning the properties of the path is hard. IP keeps no path bandwidth information, so TCP cannot ask the network about path properties. And while there are ways to estimate path bandwidth dynamically, such as packet-pair [12, 13], the estimates can easily be distorted in the presence of cross traffic.

TCP Spoofing — Another idea for getting around slow start is a practice known as "TCP spoofing," described in [14]. The idea calls for a router near the satellite link to send back acknowledgments for the TCP data to give the sender the illusion of a short delay path. The router then suppresses acknowledgments returning from the receiver, and takes responsibility for retransmitting any segments lost downstream of the router.

There are a number of problems with this scheme. First, the router must do a considerable amount of work after it sends an acknowledgment. It must buffer the data segment because the original sender is now free to discard its copy (the segment has been acknowledged) and so if the segment gets lost between the router and the receiver, the router has to take full responsibility for retransmitting it. One side effect of this behavior is that if a queue builds up, it is likely to be a queue of TCP segments that the router is holding for possible retransmission. Unlike IP datagrams, this data cannot be deleted until the router gets the relevant acknowledgments from the receiver.

Second, spoofing requires symmetric paths: the data and acknowledgments must flow along the same path through the router. However, in much of the Internet, asymmetric paths are quite common [15].

Third, spoofing is vulnerable to unexpected failures. If a path changes or the router crashes, data may be lost. Data may even be lost after the sender has finished sending and, based on the router's acknowledgments, reported data successfully transferred.

Fourth, it doesn't work if the data in the IP datagram is encrypted because the router will be unable to read the TCP header.

Cascading TCP — Cascading TCP, also know as split TCP, is a idea where a TCP connection is divided into multiple TCP connections, with a special TCP connection running over the satellite link. The thought behind this idea is that the TCP running over the satellite link can be modified, with knowledge of the satellite's properties, to run faster.

Because each TCP connection is terminated, cascading TCP is not vulnerable to asymmetric paths. And in cases where applications actively participate in TCP connection management (such as Web caching) it works well. But otherwise cascading TCP has the same problems as TCP spoofing.

Error Rates for Satellite Paths

Experience suggests that satellite paths have higher error rates than terrestrial lines. In some cases, the error rates are as high as 1 in 10^{-5}.

Higher error rates matter for two reasons. First, they cause errors in datagrams, which will have to be retransmitted. Second, as noted above, TCP typically interprets loss as a sign of congestion and goes back into a modified version of slow start. Clearly we need to either reduce the error rate to a level acceptable to TCP or find a way to let TCP know that the datagram loss is due to transmission errors, not congestion (and thus TCP should not reduce its transmission rate).

Acceptable Error Rates — What is an acceptable link error rate in a TCP/IP environment? There is no hard and fast answer to this problem. This section presents one way to think about the problem for satellites: looking at TCP's natural frequency of congestion avoidance starts, and seeking an error rate that is substantially less than that frequency.

Suppose we consider the performance of a single established TCP over an otherwise idle link. Once past the initial slow start, the established TCP connection with data to send will alternate between two modes:

- Performing congestion avoidance until a segment is dropped, at which point the TCP falls back to half its window size and resumes congestion avoidance

• Occasionally performing a slow start when loss becomes severe.

During much of the congestion avoidance phase, the TCP will typically be using the path at or near full capacity. Roughly speaking this phase lasts p round-trip times, where p is the largest value such that the following inequality is true:

$$\sum_{j=1}^{p} i \leq b$$

where b is the buffering in segments at the bottleneck in the path. (Why this equation? In congestion avoidance the TCP is sending an additional segment every round trip. Suppose we start congestion avoidance at exactly the right window size, namely the delay-bandwidth product. In the first round trip of congestion avoidance the TCP will be sending one segment more than the capacity of the path, so this segment will end up sitting in a queue. In the second round trip, the TCP will send two segments more than the capacity and these two segments will join the first one segment in the queue. And so forth, until the queue is filled and a segment is dropped.) Table 2 shows the number of bits sent during the congestion avoidance phase for a range of GEO link speeds, buffer sizes and values of p.

Clearly we would like to avoid terminating the congestion avoidance phase early, since it causes TCP to underestimate the available bandwidth. Turning this point around, we can say that a link should have an effective error rate sufficiently low that it is very unlikely that the congestion avoidance phase will be prematurely ended by a transmission error. Table 2 suggests this requirement means that satellite error rates on higher-speed links need to be on the order of 1 in 10^{12} or better. That's about the edge of the projected error rates for new satellites. The ACTS satellite routinely sends 10^{13} bits of data without an error. Proposed Ka band systems are aiming for an effective error rate of about 1 in 10^{12}.

Teaching TCP to Ignore Transmission Errors — As an alternative to, or in conjunction with, reducing satellite error rates we might wish to teach TCP to be more intelligent about handling transmission errors. There are basically two approaches: either TCP can explicitly be told that link errors are occurring or TCP can infer that link errors are occurring.

NASA has funded some experiments with explicit error notification as part of a broader study on very long space links done at Mitre [16]. One general challenge in explicit notification is that TCP and IP rarely know that transmission errors have occurred because transmission layers discard the errored datagrams without passing them to TCP and IP.

Having TCP infer which errors are due to transmission errors rather than congestion also presents challenges. One has to find a way for TCP to distinguish congestion from transmission errors reliably, using only information provided by TCP acknowledgments. And the algorithm better never make a mistake, because a failure to respond to congestion loss can exacerbate network congestion. So far as we know, no one has experimented with inferring transmission errors.

Conclusions

Satellite links are today's high-delay-bandwidth paths. Tomorrow high-delay-bandwidth paths will be everywhere. (Consider that some carriers are already installing terrestrial OC-768 [40 Gb/s] network links.) So most of the problems described in this article need to be solved not just for satellites but for high-delay paths in general.

The first step to achieving high performance is making sure the sending and receive TCP implementations contain all the modern features (large windows, PAWS, and SACK) and that the TCP window space is larger than the delay-bandwidth product of the path. Any user worried about high performance should take these steps now.

The next step is to find ways to further improve the performance of TCP over long delay paths and in particular, reduce the impact of slow start. Slow start provides an essential service; the issue is whether there are ways to reduce its start up time, especially when the connection first starts. Because long delay satellite links are only an instance of the larger problem of high-delay bandwidth paths, the authors are less interested in point solutions that only address the performance problems for satellites. We look with hope for solutions that benefit both terrestrial and satellite links.

References

[1] D. E. Comer, *Internetworking with TCP/IP, Vol. I: Principles, Protocols and Architecture, 2nd ed.*, Prentice Hall, 1991.

[2] W. R. Stevens, *TCP/IP Illustrated, Vol. I*, Addison Wesley, 1994.

[3] J. Postel, "Internet Protocol; RFC-791," Internet Requests for Comments, no. 791, Sept. 1981.

[4] C. A. Kent and J. C. Mogul, "Fragmentation Considered Harmful," *Proc. of ACM SIGCOMM '87*, Stowe, VT, 11–13, Aug. 1987, pp. 390–401.

[5] J. Mogul and S. Deering, "Path MTU Discovery; RFC-1191," Internet Requests for Comments, no. 1191, Nov. 1990.

[6] D. Borman, R. Braden, and V. Jacobson, "TCP Extensions for High Performance; RFC-1323," Internet Requests for Comments, no. 1323, May 1992.

[7] A. McKenzie, "Ptoblem with the TCP Big Window Option; RFC-1110," Internet Requests for Comments, no. 1110, Aug. 1989.

[8] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM '88*, Stanford, CA, Aug. 1988, pp. 314–329.

[9] H. F. Nielsen *et al.*, "Network Performance Effects of HTTP/1.1, CSS1, and PNG," *Proc. ACM SIGCOMM '97*, Sept. 1997.

[10] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, Sept. 1992.

[11] S. Floyd, Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic., 21, *Computer Communication Review*, Oct. 1991.

[12] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control," *Proc. ACM SIGCOMM '96*, Aug. 1996, pp. 281–291.

[13] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Avoidance and Control," *Proc. ACM SIGCOMM '94*, Aug. 1994, pp. 24–35.

[14] Z. Liu *et al.*, "Evaluation of TCP Vegas: Emulation and Experiment," *Proc. ACM SIGCOMM '95*, Aug. 1995, pp. 185–196.

[15] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. ACM SIGCOMM '91*, Zurich, Sept. 1991, pp. 3–16.

[16] J. C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," *Proc. ACM SIGCOMM '97*, Aug. 1996, pp. 270–280.

[17] Y. Zhang *et al.*, "Satellite Communications in the Global Internet—Issues, Pitfalls, and Potential," *Proc. INET '97*, 1997.

[18] V. Paxson, "End-to-End Routing Behavior in the Internet," *Proc. ACM SIGCOMM '97*, Aug. 1996, pp. 25–38.

[19] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP Extensions for Space Communications," *Proc. ACM MobiComm '97*, Nov. 1996.

Additional Reading

[1] M. Mathis, J. Mahdavi, and S. Floyd, A. Romanow, and TCP Selective Acknowledgments Options; RFC-2018, Internet Requests for Comments, no. 2018, Oct. 1996.

[2] M. Allman *et al.*, "TCP Performance Over Satellite Links," Proc. Fifth Intl. Conf. on Telecommunications Systems, Nashville, TN, March 1997.

[3] T. V. Lakshman and U. Madhow, "Window-Based Congestion Control in Networks with High Bandwidth-Delay Products," Proc. 3rd ORSA Telecommunications Conference, March 1995.

Biographies

CRAIG PARTRIDGE [SM] (craig@bbn.com) is a Principal Scientist at BBN Technologies where he does research on gigabit and terabit networks. He is the former Editor-in-Chief of *IEEE Network* and *ACM Computer Communication Review*. He is also a consulting associate professor at Stanford University and received his Ph.D. from Harvard University.

TIMOTHY SHEPARD [M] (shep@bbn.com) is a Scientist at BBN Technologies. While a student at MIT, he studied the performance behavior of TCP implementations, which led to the development of a graphical method of TCP packet trace analysis. His interests are in the engineering of large-scale complex systems, particularly those involving microwaves and millions of computers.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1999	Final Contractor Report

4. TITLE AND SUBTITLE

Study and Simulation of Enhancements for TCP Performance Over Noisy High Latency Links

5. FUNDING NUMBERS

WU–632–50–5A–00
NAS3–96014

6. AUTHOR(S)

Craig Partridge, Tim Shepard and Robert Coulter

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

BBN Technologies
10 Moulton Street
Cambridge, Massachusetts

8. PERFORMING ORGANIZATION REPORT NUMBER

E–11761

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135–3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CR—1999-209167

11. SUPPLEMENTARY NOTES

Project Manager, William Ivancic, Communications Technology Division, NASA Glenn Research Center, organization code 5610, (216) 433–3494.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category: 17 Distribution: Nonstandard

This publication is available from the NASA Center for AeroSpace Information, (301) 621–0390.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The goal of this study is to better understand how TCP behaves over noisy, high-latency links such as satellite links and propose improvements to TCP implementations such that TCP might better handle such links. This report is comprised of a series of smaller reports, presentations and recommendations. Included in these documents are a summary of the TCP enhancement techniques for large windows, protect against wrap around (PAWS), use of selective acknowledgements (SACK), increasing TCP's initial window and recommendations to implement TCP pacing.

14. SUBJECT TERMS	15. NUMBER OF PAGES
TCP; Satellite; Protocol; Internet	93
	16. PRICE CODE: A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	