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.

O 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).

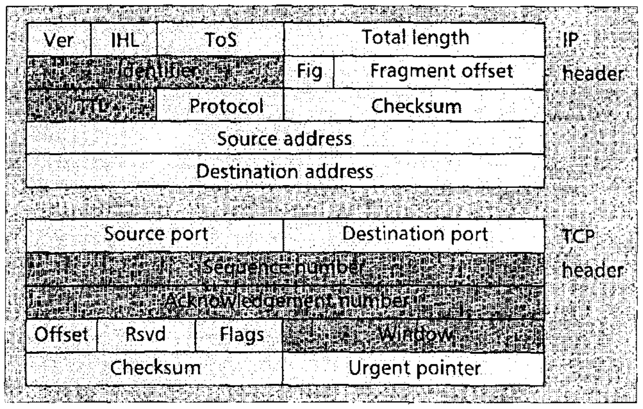
Fragments are identified using a fragment offset field (which indicates the offset of the fragment from the start of the original datagram). Datagrams are uniquely identified by their source, destination, higher layer protocol type, and a 16-bit IP identifier (which must be unique when combined with 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	1.9s	4.1s	11.3s
Slow start data (in bytes)	1,760	76,600	197,870	115,900	2,405,000	6,003,000	4,123,810	8,292,000	20,680,000

■ 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 l/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, "Problem 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.