Gigabit TCP

by Geoff Huston, APNIC

In looking back over some 30 years of experience with the Internet, the critical component of the Internet Protocol Suite that has been the foundation of its success as the technology of choice for the global communications system is the *Internet Protocol* (IP) itself, working an overlay protocol that can span almost any form of communications media. But I would also like to nominate another contender for a critical role within IP, namely the reliable transport protocol that sits on top of IP, the *Transmission Control Protocol* (TCP), and its evolution over time. In support of this nomination is the fact that the end-to-end rate-adaptive control algorithm that was adopted by TCP represented a truly radical shift from the reliable gateway-to-gateway virtual circuit flow control systems used by other protocols of similar vintage. It is also interesting to note that TCP is not designed to operate at any particular speed, but it attempts to operate at a speed that uses its fair share of all available network capacity along the network path. The fundamental property of the TCP flow control algorithm is that it attempts to be maximally efficient while also attempting to be maximally fair.

Previous articles on this topic, "TCP Performance"[12] and "The Future for TCP"[13] looked at the design assumptions behind TCP and its performance characteristics. The essential characteristic of TCP is that it attempts to establish a dynamic equilibrium with other concurrent sessions and opportunistically use all available network capacity. It achieves this by constantly altering its flow characteristics, continually probing the network to see if higher speeds are supportable, while also being prepared to immediately decrease the current sending rate in the face of received signals of network congestion.

In a world where network infrastructure capacity and complexity are related to network cost and delivered data is related to network revenue, TCP fits in well. The minimal assumptions that TCP makes about the capability of network components permit networks to be constructed using simple transmission capabilities and simple switching systems. "Simple" often is synonymous with cheap and scalable, and there is no exception here. TCP also attempts to maximize data delivery through adaptive end-to-end flow rate control and careful management of retransmission events. In other words, TCP is an enabler for cheaper networking for both the provider and consumer. For the consumer the offer of fast cheap communications has been a big motivation in the increase in demand for Internet-based services, and this—more than any other factor—has been the major enabling factor for the increased use of the Internet itself. "Cheap" is often enough in this world, and TCP certainly helps to make data communications efficient and therefore cheap.

Although TCP is highly effective in many networking environments, that does not mean it is highly effective in every environment. For example:

- In those wireless environments where there is significant wireless noise, TCP may confuse the outcome of radio-based signal corruption and the corresponding packet drop with the outcome of network congestion, and consequently the TCP session may back off its sending rate too early and back off for too long.
- TCP also backs off too early when the network routers have insufficient buffer space. This effect is more subtle, but it is related to the coarseness of the TCP algorithm and the consequent burstiness of TCP packet sequences. These bursts, which occur at up to twice the bottleneck capacity rate, are smoothed out by network buffers. Buffer exhaustion in the interior of the network causes packet drop, which causes the generation of a loss signal to the active TCP session, which, in turn, either halves its sending rate or—in the worst case—resets the session state and restarts with a single packet exchange. Particularly in wide-area networks, where the end-to-end delay-bandwidth product becomes a significant factor, TCP uses the network buffers to sustain a steady-state throughput that matches the available network capacity. Where the interior buffers are underconfigured in memory it is not possible to even out the TCP bursts to continuously flow through the constrained point at the available data rate.
- TCP also asks its end hosts to have local capacity equal to the available network capacity on the forward and reverse paths. The reason is that TCP does not discard data until the remote end has reliably acknowledged it, so the sending host has to retain a copy of the data for the time it takes to send the data plus the time for the remote end to send the matching acknowledgement.

Even accounting for these limitations, it is true to say that TCP works amazingly well in most environments. Nevertheless, one area is proving to be quite a fundamental challenge to TCP as we know it, and that is the domain of wide-area, very-high-speed data transfer.

Very-High-Speed TCP

End host computers, even laptop computers these days, are typically equipped with Gigabit Ethernet interfaces, and have gigabytes of memory and internal data channels that can move gigabits of data per second between memory and the network interface. Current IP networks are constructed using multigigabit circuits and high-capacity switches and routers (assuming there is still a quantitative difference between these two forms of packet switching equipment). If the end hosts and the network both can support gigabit transmissions then a TCP session should be able to operate end to end at gigabits per second, and achieve the same efficiency at gigabit speeds as it does today at megabit speeds—right?

Well, no, not exactly!

This conclusion is not obvious, particularly when the TCP Land Speed Record is now at some 7Gbps across a distance that spans 30,000 km of network. What is going on?

Let's return to the basics of TCP to understand some of the variables with very-high-speed TCP. TCP operates in one of two states, that of *slow start* and *congestion avoidance*.

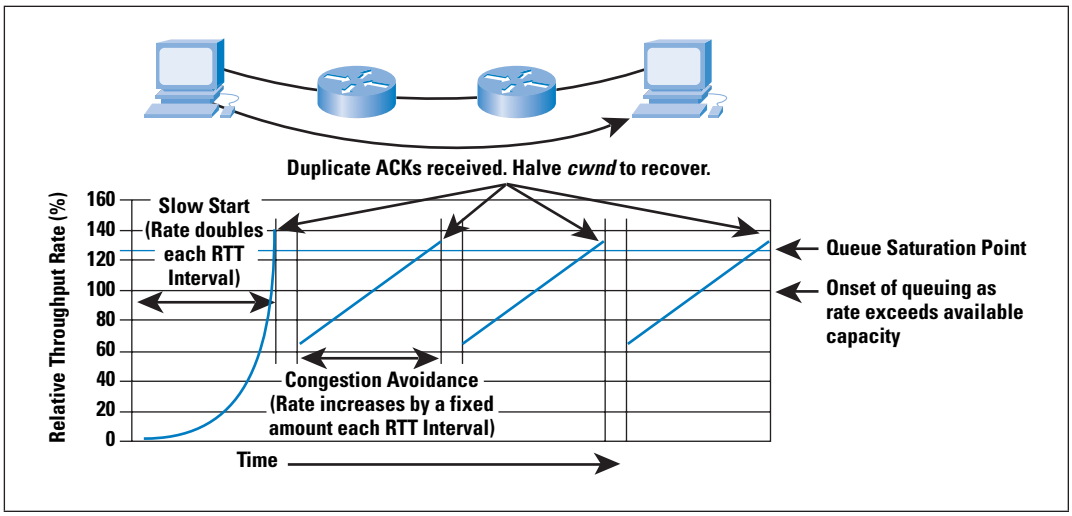
- *Slow start* mode is the initial mode of operation of TCP in any session, as well as its "reset" mode. In this mode, TCP sends two packets in response to each ACK packet that advances the sender's window. In approximate terms (*delayed* ACKs notwithstanding), this mode allows TCP to double its sending rate in each successive lossless *round-trip-time* (RTT) interval. The rate increase is exponential, effectively doubling each RTT interval, and the rate increase is bursty, effectively sending data into the network at twice the bottleneck capacity during this phase.

  Sending data into the network at twice the bottleneck data speed is possible because of the "ACK clocking" property of TCP. Disregarding the complications of the TCP delayed ACK mechanism for a second, a TCP receiver generates a new ACK packet each time a packet arrives at the receiver. The sending rate of the ACKs is, in effect, the same as the receiving rate for the data packets. Assuming a one-way data transfer, so that ACK packets in the reverse direction are of minimal size, and assuming minimal jitter on the reverse path from the receiver back to the sender, the arrival rate of ACKs at the sender is comparable to the arrival rate of data packets at the receiver. In other words, the return ACK rate is comparable to the bottleneck capacity of the forward network path from sender to receiver. Sending two packets per received ACK is effectively sending packets into the network at twice the bottleneck capacity. At the bottleneck point the switching unit receives twice the amount of data than it can transmit to the output device over a period that corresponds to the delay-bandwidth product of the bottleneck link. Hence the comment that TCP is a *bursty* protocol, particularly at startup. For this reason TCP tends to operate more effectively across network switching elements that are generously endowed with memory, or have for each output port a buffer capacity roughly equal to the delay-bandwidth product of the link that is attached to that port.

- In the other operating mode, that of *congestion avoidance,* TCP sends an additional segment of data for each loss-free round-trip time interval. This increase is additive rather than exponential, increasing the sender's speed at the constant rate of one segment per RTT interval.

TCP undertakes a state transition upon the detection of packet loss. Small-scale packet loss (of the order of 1 or 2 packets per loss event) causes TCP to halve its sending rate and enter congestion avoidance mode, irrespective of whether it was in this mode already. Repetition of this cycle gives the classic sawtooth pattern of TCP behavior, and the related derivation of TCP performance as a function of packet loss rate. Longer sustained packet loss events cause TCP to stop using the current session parameters, recommence the congestion control session using the restart window size, and enter the slow start control mode once again. (See Figure 1).

Figure 1: TCP Behavior



But what happens when two systems are at opposite sides of a continent with a high-speed path between them? How long does it take for a single TCP session to get up to a data transfer rate of 10 Gbps? Can a single session operate at a sustained rate of 10 Gbps?

Let's look at a situation such as the network path from Brisbane, on the eastern side of the Australian continent, to Perth on the western side. The cable path is essentially along the southern coast of the continent, so the RTT delay is 70 ms, implying that there are 14.3 round-trip intervals per second. Let's also assume that the packet size being used is 1500 octets, or 12,000 bits, and the TCP initial window size is a single packet. And let's also assume that the bottleneck capacity of the host-to-host path between Brisbane and Perth is 10 Gbps.

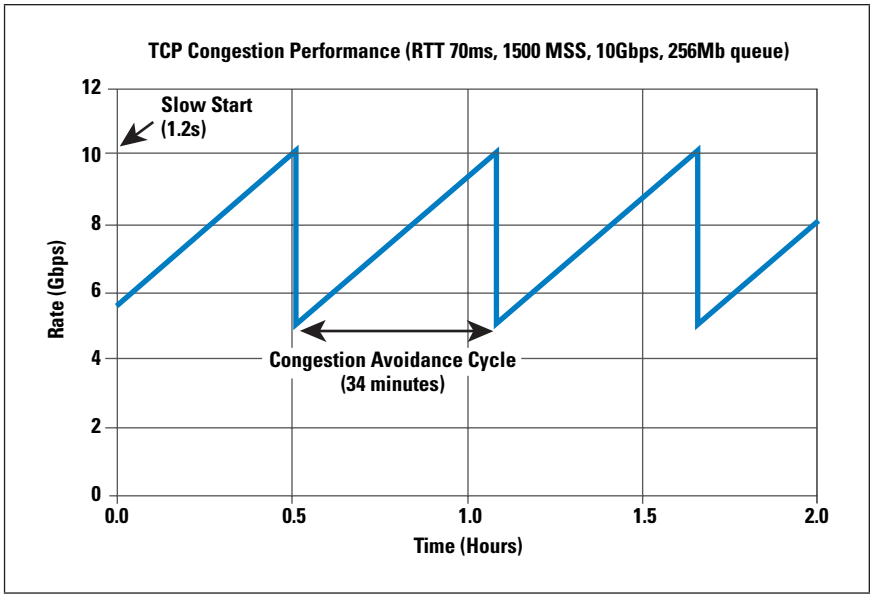
In a simple slow start model the sending speed doubles every 70 ms, so after 17 RTT intervals where the sending rate has doubled for each interval, or after some 1.2 seconds have elapsed, the transfer speed reaches 11.2 Gbps (assuming a theoretical host with sufficiently fast hardware components, sufficiently fast internal data paths, and adequate memory). At this stage let's assume that the sending rate exceeded the buffer capacity at the bottleneck point in the network path. Packet drop will occur, because the critical point buffers in the network path are now saturated.

At the point of reception of an ACK sequence that signals packet loss, the TCP sender's congestion window will halve, as will the TCP sending rate, and TCP will switch to congestion avoidance mode. In congestion avoidance mode the rate increase is 1 segment per RTT, equivalent to sending an additional 12 kilobits per RTT, or, given the session parameters as specified previously, equivalent to a rate increase of 171 kbps each RTT. So how long will it take TCP to recover and get back to a sending rate of 10 Gbps?

If this were a T1 circuit where the available path bandwidth is 1.544 Mbps, and congestion loss occurred at a sending rate of 2 Mbps (higher than the bottleneck transmission capacity due to the effect of queuing buffers within the network), then TCP would rate halve to 1 Mbps and then use congestion avoidance to increase the sending rate back to 2 Mbps. Within the selected parameters of a 70-ms RTT and 1500-byte segment size, this process involves using congestion avoidance to inflate the congestion window from 6 segments to 12. This process takes 0.42 seconds. So as long as the network can operate without packet loss for the session over an order of 1-second intervals, then TCP can comfortably operate at maximal speed in a megabit-per-second network.

What about our 10-Gbps connection? The first estimate is the amount of usable buffer space in the switching elements. Assuming a total of 256 MB of usable queue space on the network path prior to the onset of queue saturation, the TCP session operating in congestion avoidance mode will experience packet loss some 590 RTT intervals after reaching the peak transmission speed of 10 Gbps, or some further 41 seconds, at which point the TCP sending rate in congestion avoidance mode is 10.1 Gbps. For all practical purposes the TCP congestion avoidance mode causes the sawtooth oscillation of this ideal TCP session between 5.0 Gbps and 10.1 Gbps. A single iteration of this sawtooth cycle takes 2062 seconds, or 34 minutes and 22 seconds. The implication here is that the network has to be stable in terms of no packet loss along the path for time scales of the order of tens of minutes (or some billions of packets), and corresponding transmission bit error rates that are less than 10^{-14}. It also implies massive data sets to be transferred, because the amount of data passed in just one TCP congestion avoidance cycle is 1.95 terabytes (1.95×10^{12} bytes). It is also the case that the TCP session cannot make full use of the available network bandwidth, because the average data transfer rate is 7.55 Gbps under these conditions, not 10 Gbps. (See Figure 2).

Figure 2: TCP Behavior at High Speed



Clearly something is unexpected with this scenario, because it certainly looks like it is a difficult and lengthy task to fill a long-haul, high-capacity cable with data, and TCP is not behaving as expected. Although experimenting with the boundaries of TCP is in itself an interesting area of research, some practical problems here could well benefit from this type of high-speed transport.

A commonly quoted example, and certainly one of the more impressive ones is the Large Hadron Collider at CERN:

"The CERN Particle Physics lab in Geneva, Switzerland, successfully transmitted a data stream averaging 600Mbytes per second for 10 days to seven countries in Europe and the US. It was a crucial test of the computing infrastructure for the Large Hadron Collider being built at CERN. The LHC will be the most data intensive physics instrument ever built, generating 1500 Megabytes every second for a decade or more."

—*New Scientist,* 30 April 2005

TCP and the Land Speed Record

The TCP Land Speed Record was originally an informal effort to achieve record-breaking TCP transfer speeds across IP networks. The late 1980s and early 1990s saw some noted milestones, particularly with Van Jacobson's efforts in achieving sustained 10-Mbps and 45-Mbps TCP transfer speeds.

This activity has been incorporated into the Internet2 program, with the introduction of some formal rules about what constitutes a TCP Land Speed effort. In particular, the rules now have times, distances, and TCP constraints, and they call for the use of operational networks. Updates to the record have been posted frequently in recent years, and as of May 2006 the IPv4 single stream record is a TCP session operating at 7.21 Gbps for 30 minutes over 30,000 km of fibre path.

It is certainly possible to have TCP perform for sustained intervals at very high speed, as the land speed records for TCP show, but something else is happening here, and a set of preconditions need to be met before attempting to set a new record:

- First, it is good—indeed essential—to have the network path all to yourself. Any form of packet drop is a major problem here, so the best way to ensure no packets are lost is to keep the network path all to yourself.
- Secondly, it is good—indeed essential—to have a fixed latency. If the objective of the exercise is to reach a steady-state data transmission, then any change in latency, particularly a reduction in latency, has the risk of a period of oversending, which in turn has a risk of packet loss. So keep the network as stable as possible.
- Thirdly, it is good—indeed essential—to have extremely low bit error rates from the underlying transmission media. Data corruption causes checksum failure, which causes packet drop.
- Lastly, it is essential to know in advance both the round-trip latency and the available bandwidth.

You can then multiply these two numbers together (RTT and bandwidth), divide by the packet size, round down, and be sure to configure the sending TCP session to have precisely this buffer size, and the receiver to have a slightly larger size. And then start up the session.

The intention here is for TCP to use slow start to the point where the sender runs out of buffer space, at which point it will continue to sit at this buffer speed for as long as the sender, receiver and network path all remain in a stable state. For the example configuration of a 10-Gbps system with 70 ms RTT, setting a buffer limit of 116,000 packets will cause the TCP session to operate at 9.94 Gbps. As long as the latency remains steady (no jitter), with no bit errors, and as long as there is no other cross traffic, in theory this sending rate can be sustained indefinitely, with a steady stream of data packets being matched by a steady stream of ACK packets.

Of course, this situation is artificially constrained. The real concerns here with the protocol are in the manner in which it shares a network path with other concurrent sessions as well as its ability to fill the available network capacity. In other words, what would be good to see is a high-speed, high-volume version of TCP that could coexist on a network with all other forms of traffic, and, perhaps more ambitiously, that this high-speed form of TCP could share the network fairly with other traffic sessions while at the same time making maximal use of the network. The problem with TCP in its current incarnation is that it takes way too long in its additive increase mode (congestion avoidance) to recover its sustainable operating speed when operating at high speed across transcontinental-size network paths. If we want very-high-speed TCP to be effective and efficient, then we need to look at changes to TCP for high-speed operation.

High-Speed TCP

There are two basic approaches to high-speed TCP: parallelism of existing TCP, or changes to TCP to allow faster acceleration rates in a single TCP stream.

Using parallel TCP streams as a means of increasing TCP performance is an approach that has existed for some time. The original HTTP specification, for example, allowed the use of parallel TCP sessions to download each component of a Webpage (although HTTP 1.1 reverted to a sequential download model because the overheads of session startup appeared to exceed the benefits of parallel TCP sessions in this case). Another approach to high-speed file transfer through parallelism is that of GRID FTP. The basic approach is to split up the communications payload into numerous discrete components, and send each of these components simultaneously. Each component of the transfer can be between the same two endpoints (such as GRID FTP), or can be spread across multiple endpoints (as with BitTorrent).

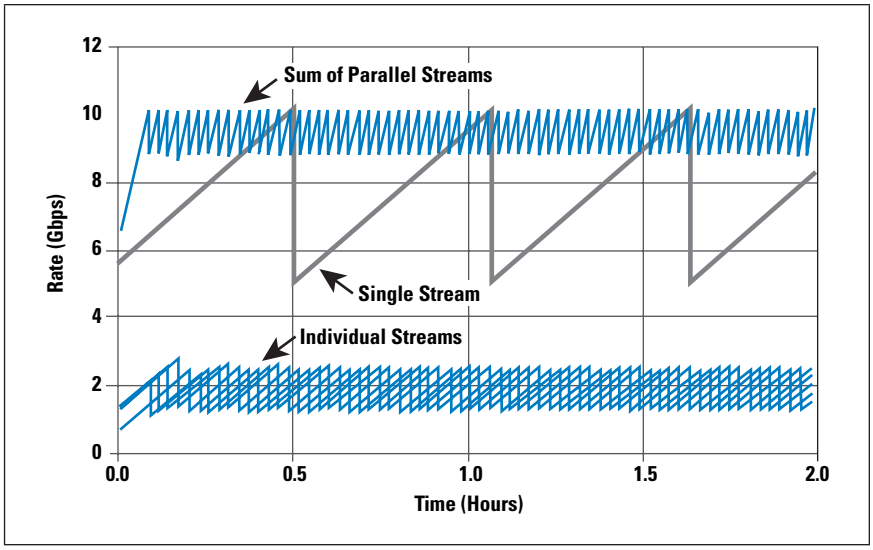
But for parallel TCP to operate correctly, we need to have already assembled all the data (or at a minimum know where all the data components are located). Where the data is being generated in real time (such as observatories or particle colliders) in massive quantities, there may be no choice but to treat the data set as a serial stream and use a high-speed transport protocol to dispatch it. In this case the task is to adjust the basic control algorithms for TCP to allow it to operate at high speed, but also to operate "fairly" on a mixed-traffic high-speed network.

Parallel TCP

Using parallelism as a key to higher speed is a common computing technique, and lies behind many supercomputer architectures. The same can apply to data transfer, where a data set is divided into numerous smaller chunks, and each component chunk is transmitted using its own TCP session. The underlying expectation here is that when using some number, *N,* of parallel TCP sessions, a single packet drop event will most probably cause the fastest of the *N* sessions to rate halve, because the fastest session will have more packets in flight in the network, and is therefore the most likely session to be impacted by a packet drop event. This session will then use congestion avoidance rate increase to recover, implying that the response to a single packet drop is reduction of the sending rate by at most $1/(2N)$. For example, using five parallel TCP sessions, the response to a single packet drop event is to reduce the total sending rate by $1/(2 \times 5)$, or 1/10, as compared to the response from a single TCP session, where a single packet drop event would reduce the sending rate by ½.

A simulated version of five parallel sessions in a 10 Gbps session is shown in Figure 3.

Figure 3: Parallel TCP Simulation: Single vs Parallel Streams



The essential characteristic of the aggregate flow is that under lossless conditions the data flow of N parallel sessions increases at a rate N times faster than a single session in congestion avoidance mode. Also the response to an isolated loss event is that of rate halving of a single flow, so that the total flow rate under ideal conditions is between R and $R \times (2N - 1)/2N$, or a long-term average throughput of $R \times (4N - 1)/4N$. For $N = 100$ our theoretical 10-Gbps connection could now operate at 9.9 Gbps.

Of course practice is different from theory, and a considerable amount of work has looked at the performance of parallel TCP under various conditions, in terms of both maximizing throughput and choosing the most efficient number of parallel active streams to use. Part of the problem is that although simple simulations, such as that used to generate Figure 4, tend to evenly distribute each of the parallel sessions to maximize the throughput, there is the more practical potential that the individual sessions self-synchronize. Because the parallel sessions have a similar range of window sizes, it is possible that at a given point in time a similar number of packets will be in the network path from each stream. If the packet drop event is a multiple packet drop event, such as a tail-drop queue, then it is entirely feasible that numerous parallel streams will experience packet loss simultaneously, and there is the consequential potential for the streams to fall into synchronization.

The two extremes, evenly distributed and tightly synchronized multiple streams, are indicated in Figure 4 The average throughput of parallel synchronized streams is the same as a single stream over extended periods in this simulation, and both are certainly far worse than an evenly distributed set of parallel streams.

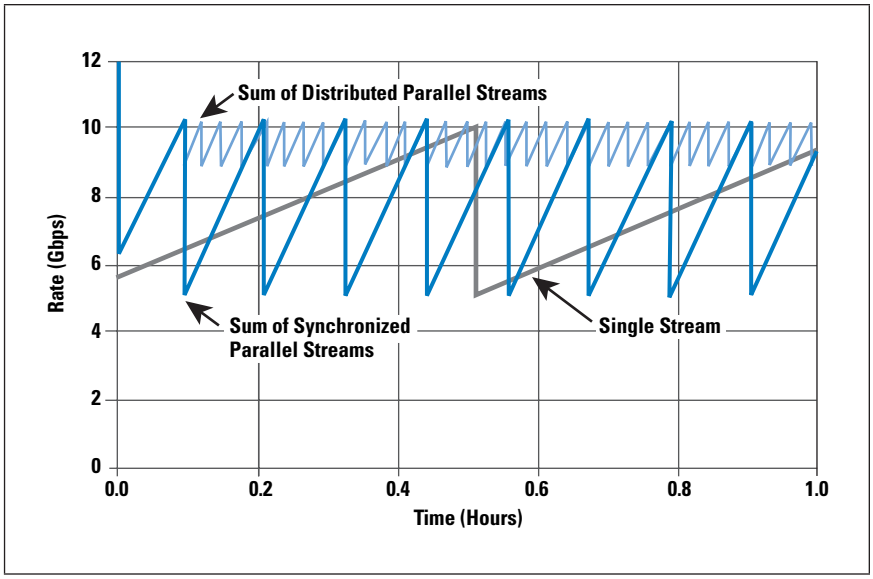


Figure 4: Comparison of Parallel TCP: Synchronized and Distributed Streams

One way to address this problem is to reunite these parallel streams into a single controlled stream that exhibits the same characteristics as evenly spread parallel streams. This approach, MulTCP, is considered in the next section.

If all this analysis of parallel TCP streams sounds a little academic and unrelated to networking today, it is useful to note that many *Internet Service Providers* (ISPs) currently see *BitTorrent* traffic as their highest-volume application. BitTorrent is a peer-to-peer protocol that undertakes transfer of datasets using a highly parallel transfer technique. Under BitTorrent the original dataset is split into blocks, each of which can be downloaded in parallel. The subtle twist here is that the individual sessions do not have the same source points, and the host may take feeds from many different sources simultaneously, as well as offering itself as a feed point for the already downloaded blocks. This behavior exploits the peer-to-peer nature of these networks to a very high extent, potentially not only exploiting parallel TCP sessions for speed gains, but also exploiting diverse network paths and diverse data sources to avoid single path congestion. Considering its effectiveness in terms of maximizing transfer speeds for high-volume datasets and its relative success in truly exploiting the potential of peer-to-peer networks—and of course the dramatic acceptance of BitTorrent and its extensive use—BitTorrent probably merits closer examination, but perhaps that is for another time and an article of its own.

Very High Speed Serial TCP

The other general form of approach is to reexamine the current TCP control algorithm to see if there are parameter or algorithm changes that could allow TCP to undertake a better form of rate adaptation to these high-capacity, long-delay network paths. The aim here is to achieve a good congestion response algorithm that does not amplify transient congestion conditions into sustained disaster areas, while at the same time being able to support high-speed data transfers, thereby making effective use of all available network capacity.

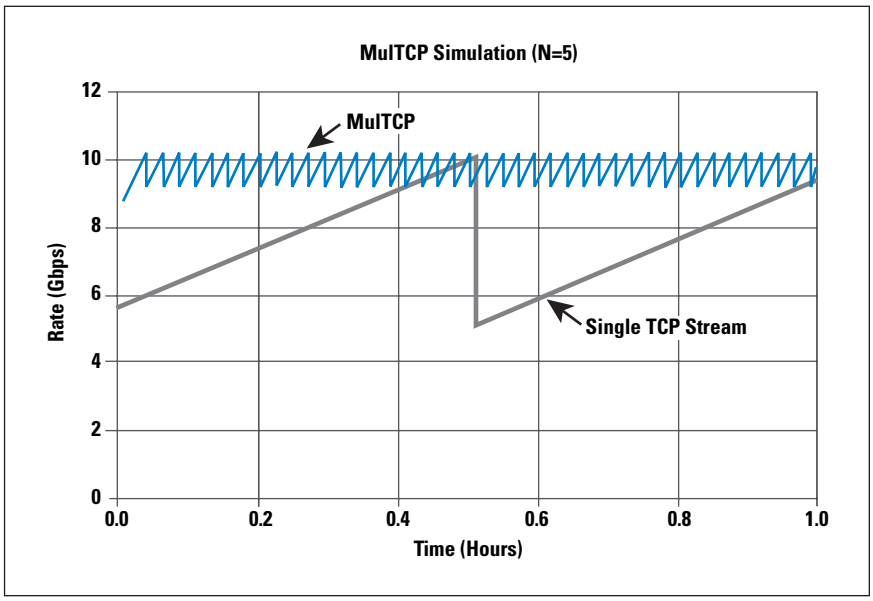
We also want TCP to behave sensibly in the face of other TCP sessions, so that it can share the network with other TCP sessions fairly.

MulTCP

The first of these approaches is *MulTCP*[1], which is a single TCP stream that behaves in a manner equivalent to *N* parallel TCP sessions, where the virtual sessions are evenly distributed in order to achieve the optimal outcome in terms of throughput. The essential changes to TCP are in congestion avoidance mode and the reaction of packet loss. In congestion avoidance mode MulTCP increases its congestion window by *N* segments per RTT, rather than the default of a single segment. Upon packet loss, MulTCP reduces its window by W/(2*N*), rather than the default of W/2. MulTCP uses a slightly different version of slow start, increasing its window by 3 segments per received ACK, rather than the default value of 2.

MulTCP represents a simple change to TCP that does not depart radically from the TCP congestion control algorithm. Of course when choosing an optimal value for *N*, some understanding of the network characteristics would help. If the value for *N* is too high, the MulTCP session has a tendency to claim an unfair amount of network capacity, but if the value is too low, it does not necessarily take full advantage of available network capacity. Figure 5 shows MulTCP compared to a simulation of an equivalent number of parallel TCP streams and a single TCP stream (*N* = 5 in this particular simulation).

Figure 5: MulTCP



Good as this is, there is the lingering impression that we can do better. It would be better not to have to configure the number of virtual parallel sessions; it would be better to support fair outcomes when competing with other concurrent TCP sessions over a range of bandwidths; and it would be better to have a wide range of scaling properties.

There is no shortage of options here for fine-tuning various aspects of TCP to meet some of these preferences, ranging from adaptations applied to the TCP rate control equation to approaches that view the loading onto the network as a power spectrum problem.

HighSpeed TCP

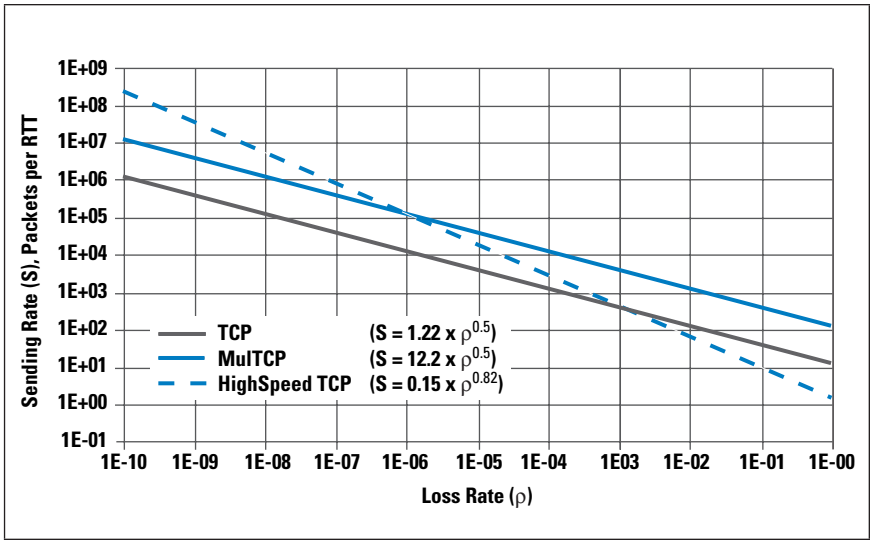
Another approach, described in [2], "HighSpeed TCP for Large Congestion Windows" looks at this from the perspective of the TCP rate equations, developed by Sally Floyd at ICIR.

When TCP operates in congestion avoidance mode at an average speed of W packets per RTT, then the number of packets per RTT varies between $(2/3)W$ and $(4/3)W$. Each cycle takes $(2/3)W$ RTT intervals, and the number of packets per cycle is therefore $(2/3)W^2$ packets. This result implies that the rate can be sustained at W packets per RTT as long as the packet loss rate is 1 packet loss per cycle, or a loss rate, ρ, where $\rho = 1/((2/3)W^2)$. Solving this equation for W gives the average packet rate per RTT of $W = \sqrt{(1.5)}/\sqrt{(\rho)}$. The general rate function for TCP, R, is therefore: $R = (\text{MSS/RTT}) \times (\sqrt{(1.5)}/\sqrt{(\rho)})$, where MSS is the TCP packet size.

Taking this same rate equation approach, what happens for N multiple streams? The ideal answer is that the parallel streams operate N times faster at the same loss rate, or, as a rate equation the number of packets per RTT, W_N, can be expressed as $W_N = N((\sqrt{(1.5)}/\sqrt{(\rho)})$, and each TCP cycle is compressed to an interval of $(2/3)\ (W_N^2/N^2)$.

But perhaps the desired response is not to shift the TCP rate response by a fixed factor of N—as is the intent with MulTCP—but to adaptively increase the sending rate through increasing values of N as the loss rate falls. The proposition made by HighSpeed TCP is to use a TCP response function that preserves the fixed relationship between the logarithm of the sending rate and the logarithm of the packet loss rate, but alters the slope of the function, such that TCP increases its congestion avoidance increment as the packet loss rate falls. This relationship is shown in Figure 6 where the log of the sending rate is compared to the log of the packet loss rate. MulTCP preserves the same relationship between the log of the sending rate and the log of the packet loss rate, but alters the offset, whereas changing the value of the exponent of the packet loss rate causes a different slope in the rate equation.

Figure 6: TCP Response Functions



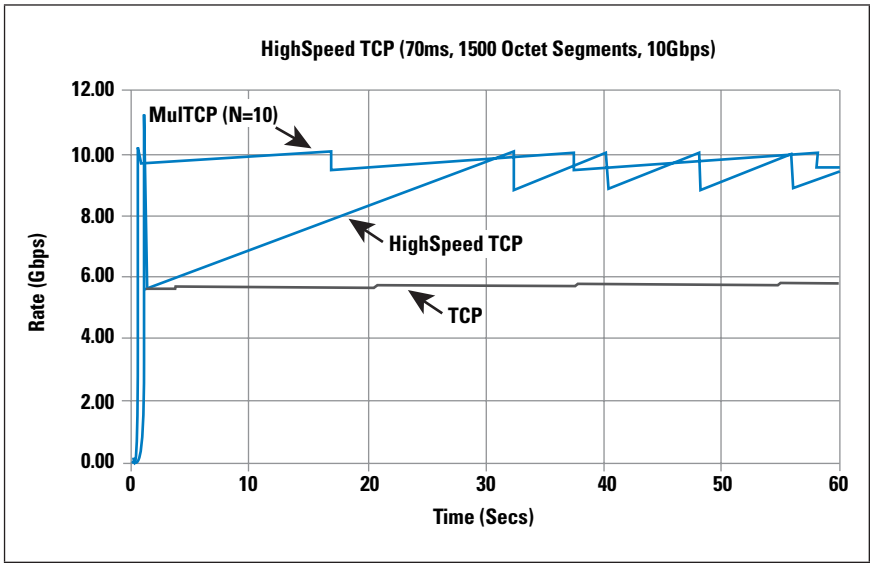
One way to look at the HighSpeed TCP proposal is that it operates in the same fashion as a turbocharger on an engine; the faster the engine is running, the higher the turbo-charged boost to the normal performance of the engine. Below a certain threshold value the TCP congestion avoidance function is unaltered, but when the packet loss rate falls below a certain threshold value then the higher speed congestion avoidance algorithm is invoked. The higher-speed rate equation proposed by HighSpeed TCP is based on achieving a transfer rate of 10 Gbps over a 100-ms latency path with a packet loss rate of 1 in 10 million packets. Working backward from these parameters gives us a rate equation for *W*, the number of packets per RTT interval of $W = 0.12/\rho^{0.835}$, approximately equivalent to a MulTCP session where the number of parallel sessions, *N*, is raised as the TCP rate increases.

This result can be translated into two critical parameters for a modified TCP: the number of segments to be added to the current window size for each lossless RTT time interval, and the number of segments to reduce the window size in response to a packet loss event. Conventional TCP uses values of 1 and (½)*W*, respectively. The HighSpeed TCP approach increases the congestion window by 1 segment for TCP transfer rates up to 10 Mbps, but then uses an increase of some 6 segments per RTT for 100 Mbps, 26 segments at 1 Gbps and 70 segments at 10 Gbps. In other words the faster the TCP rate that has already been achieved, then the greater the rate acceleration. Highspeed TCP also advocates a smaller multiplicative decrease in response to a single packet drop, so that at 10 Mbps the multiplier would be ½, at 100 Mbps the multiplier is 1/3, at 1 Gbps it is 1/5, and at 10 Gbps it is set to 1/10.

What does this process look like? Figure 7 shows a HighSpeed TCP simulation. What is not easy to discern is that during congestion avoidance HighSpeed TCP opens its sending window in increments of 53 through 64 segments each RTT interval, making the rate curve slightly upward during this window expansion phase.

HighSpeed TCP manages to recover from the initial rate halving from slow start in about 30 seconds, and operates at an 8-second cycle, as compared to the 38-minute cycle of a single TCP stream, or a 10-stream MulTCP session that operates at a 21-second cycle.

Figure 7: HighSpeed TCP Simulation



One other aspect of this work concerns the so-called slow start algorithm, which at these speeds is not really slow at all. The final RTT interval in our scenario has TCP attempting to send an additional 50 MB of data in just 70 ms, meaning an additional 33,333 packets are pushed into the network queues. Unless the network path is completely idle at this point, it is likely that hundreds—if not thousands—of these packets will be dropped in this step, pushing TCP back into a restart cycle. HighSpeed TCP has proposed a limited slow start to accompany HighSpeed TCP that limits the inflation of the sending window to a fixed upper rate per RTT to avoid this problem of slow start overwhelming the network and causing the TCP session to continually restart. Other changes for HighSpeed TCP are to extend the limit of three duplicate ACKs before retransmitting to a higher value, and a smoother recovery when a retransmitted packet is itself dropped.

Scalable TCP

Of course HighSpeed TCP is not the only offering in the high-performance TCP stakes.

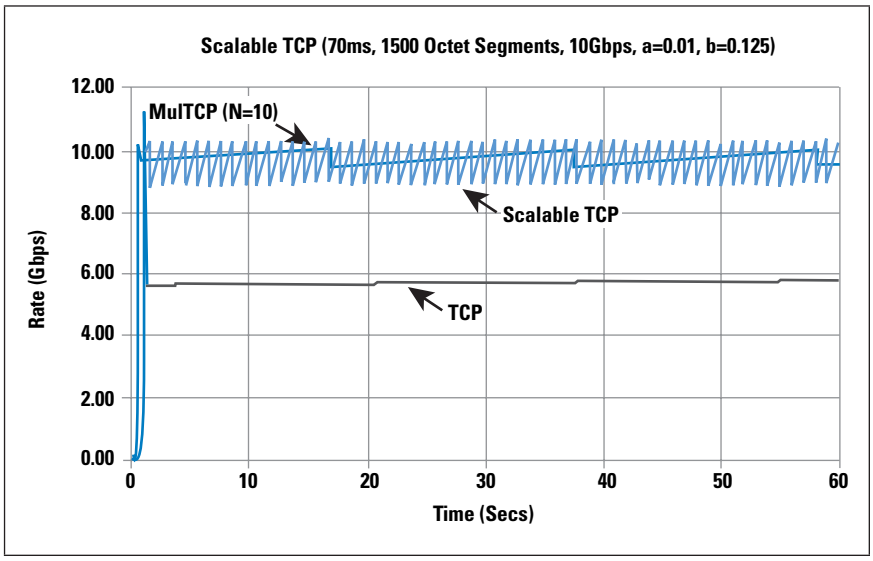
Scalable TCP[3], developed by Tom Kelly at Cambridge University, attempts to break the relationship between TCP window management and the RTT time interval. It does this by noting that in "conventional" TCP, the response to each ACK in congestion avoidance mode is to inflate the sender's congestion window size *(cwnd)* by *(1/cwnd)*, thereby ensuring that the window is inflated by 1 segment each RTT interval. Similarly the window halving on packet loss can be expressed as a reduction in size by *(cwnd/2)*. Scalable TCP replaces the additive function of the window size by the constant value *a*.

The multiplicative decrease is expressed as a fraction *b*, which is applied to the current congestion window size.

In Scalable TCP, for each ACK the congestion window is inflated by the constant value *a*, and upon packet loss the window is reduced by the fraction *b*. The relative performance of Scalable TCP as compared to conventional TCP and MulTCP is shown in Figure 8.

The essential characteristic of Scalable TCP is the use of a multiplicative increase in the congestion window, rather than a linear increase, effectively creating a higher frequency of oscillation of the TCP session, probing upward at a higher rate and more frequently than HighSpeed TCP or MulTCP. The frequency of oscillation of Scalable TCP is independent of the RTT interval, and the frequency can be expressed as $f = log(1 - b) / log(1 + a)$. In this respect, longer networks paths exhibit similar behavior to shorter paths at the bottleneck point. Scalable TCP also has a linear relationship between the log of the packet loss rate and the log of the sending rate, with a greater slope of HighSpeed TCP.

Figure 8: Scalable TCP



BIC and CUBIC

The common concern here is that TCP underperforms in those areas of application where there is a high bandwidth-delay product. The common problem observed here is that the additive window inflation algorithm used by TCP can be very inefficient in long-delay, high-speed environments. As can be seen in Figure 10, the ACK response for TCP is a congestion window inflation operation where the amount of inflation of the window is a function of the current window size and some additional scaling factor.

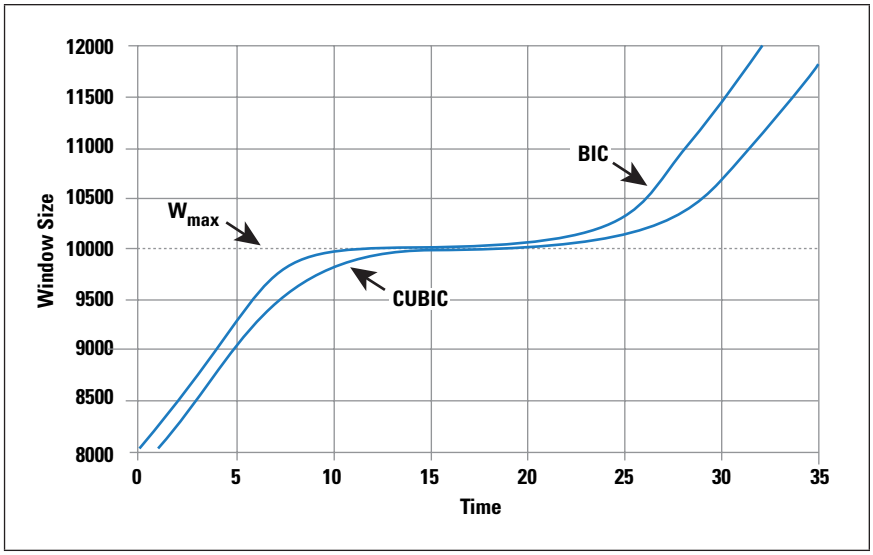
Binary Increase Congestion Control (BIC)[4] takes a different view, by assuming that TCP is actively searching for a packet sending rate that is on the threshold of triggering packet loss, and uses a binary chop search algorithm to achieve this efficiently.

When BIC performs a window reduction in response to packet drop, it remembers the previous maximum window size, as well as the current window setting. With each lossless RTT interval BIC attempts to inflate the congestion window by one half of the difference between the current window size and the previous maximum window size. In this way BIC quickly attempts to recover from the previous window reduction, and, as BIC approaches the old maximum value, it slows down its window inflation rate, halving its rate of window inflation each RTT. This process is not quite as drastic as it may sound, because BIC also uses a maximum inflation constant to limit the amount of rate change in any single RTT interval. The resultant behaviour is a hybrid of a linear and a non-linear response, where the initial window inflation after a window reduction is a linear increase, but as the window approaches the previous point where packet loss occurred the rate of window increase slows down. BIC uses the complementary approach to window inflation when the current window size passes the previous loss point. Initially further window inflation is small, and the size of the window inflation value doubles for each RTT, up to a limit value, beyond which the window inflation is once more linear.

BIC can be too aggressive in low RTT networks and in slower speed situations, leading to a refinement of BIC, namely CUBIC[5]. CUBIC uses a third-order polynomial function to govern the window inflation algorithm, rather than the exponential function used by BIC. The cubic function is a function of the elapsed time since the previous window reduction, rather than the implicit use by BIC of an RTT counter, so that CUBIC can produce fairer outcomes in a situation of multiple flows with different RTTs. CUBIC also limits the window adjustment in any single RTT interval to a maximum value, so the initial window adjustments after a reduction are linear. Here the new window size, W, is calculated as $W = C(t - K)^3 + W_{max}$, where C is a constant scaling factor, t is the elapsed time since the last window reduction event, W_{max} is the size of the window prior to the most recent reduction and K is a calculated value: $K = (W_{max}\ \beta / C)1/3$. This function is more stable when the window size approaches the previous window size W_{max}. The use of a time interval rather than an RTT counter in the window size adjustment is intended to make CUBIC more sensitive to concurrent TCP sessions, particularly in short RTT environments.

Figure 9 shows the relative adjustments for BIC and CUBIC, using a single time base. The essential difference between the two algorithms is evident in that the CUBIC algorithm attempts to reduce the amount of change in the window size when near the value where packet drop was previously encountered.

Figure 9: Window Adjustment for BIC and CUBIC



Westwood

The "steady state" mode of TCP operation is one that is characterized by the "sawtooth" pattern of rate oscillation. The additive increase is the means of exploring for viable sending rates while not causing transient congestion events by accelerating the sending rate too quickly. The multiplicative decrease is the means by which TCP reacts to a packet loss event that is interpreted as being symptomatic of network congestion along the sending path.

BIC and CUBIC concentrate on the rate increase function, attempting to provide for greater stability for TCP sessions as they converge to a long-term available sending rate. The other perspective is to examine the multiplicative decrease function, to see if there is further information that a TCP session can use to modify this rate decrease function.

The approach taken by Westwood[6], and a subsequent refinement, Westwood+[7], is to concentrate on the halving by TCP of its congestion window in response to packet loss (as signaled by three duplicate ACK packets). The conventional TCP algorithm of halving the congestion window can be refined by the observation that the stream of return ACK packets actually provides an indication of the current bottleneck capacity of the network path, as well as an ongoing refinement of the minimum RTT of the network path. The Westwood algorithm maintains a bandwidth estimate by tracking the TCP acknowledgement value and the inter-arrival time between ACK packets in order to estimate the current network path bottleneck bandwidth. This technique is similar to the "Packet Pair" approach, and that used in the TCP Vegas. In the case of the Westwood approach the bandwidth estimate is based on the receiving ACK rate, and is used to set the congestion window, rather than the TCP send window. The Westwood sender keeps track of the minimum RTT interval, as well as a bandwidth estimate based on the return ACK stream. In response to a packet loss event, Westwood does not halve the congestion window, but instead sets it to the bandwidth estimate times the minimum RTT value.

If the current RTT equals the minimum RTT, implying that there are no queue delays over the entire network path, then the sending rate is set to the bandwidth of the network path. If the current RTT is greater than the minimum RTT, the sending rate is set to a value that is lower than the bandwidth estimate, and allows for additive increase to once again probe for the threshold sending rate when packet loss occurs.

The major concern here is the potential variation in inter-ACK timing, and although Westwood uses every available data and ACK pairing to refine the current bandwidth estimate, the approach also uses a low pass filter to ensure that the bandwidth estimate remains relatively stable over time. The practical result here is that the receiver may be performing some form of ACK distortion, such as a delayed ACK response, and the network path contains jitter components in both the forward and reverse direction, so that ACK sequences can arrive back at the sender with a high variance of inter-ACK arrival times. Westwood+ further refines this technique to account for a false high reading of the bandwidth estimate due to ACK compression, using a minimum measurement interval of the greater of the RTT or 50 ms.

The intention here is to ensure that TCP does not over-correct when it reduces its congestion window, so that the problems relating to the slow inflation rate of the window are less critical for overall TCP performance. The critical part of this work lies in the filtering technique that takes a noisy sequence of measurement samples and applies an anti-aliasing filter followed by a low-pass discrete-time filter to the data stream in order to generate a reasonably accurate available bandwidth estimate. This estimate is coupled with the minimum RTT measurement to provide a lower bound for the TCP congestion window setting following detection of packet loss and subsequent fast retransmit repair of the data stream. If the packet loss is caused by network congestion the new setting will be lower than the threshold bandwidth (lower by the ratio RTT_{min} / $RTT_{current}$), so that the new sending rate will also allow the queued backlog of traffic along the path to clear. If the packet loss is caused by media corruption, the RTT value will be closer to the minimum RTT value, in which case the TCP session-rate backoff is smaller, allowing for a faster recovery of the previous data rate.

Although this approach has direct application in environments where the probability of bit-level corruption is intermittently high, such as often encountered with wireless systems, it also has some application to the long-delay, high-speed TCP environment. The rate backoff of TCP Westwood is one that is based on the RTT_{min} / $RTT_{current}$ ratio, rather than rate halving in conventional TCP, or a constant ratio, such as used in MulTCP, allowing the TCP session to oscillate its sending rate closer to the achievable bandwidth rather than performing a relatively high-impact rate backoff in response to every packet loss event.

H-TCP

The observation made by the proponents of H-TCP[9] is that better TCP outcomes on high-speed networks is achieved by modifying TCP behavior to make the time interval between congestion events smaller. The signal that TCP has taken up its available bandwidth is a congestion event, and by increasing the frequency of these events TCP will track this resource metric with greater accuracy. To achieve this tracking, the H-TCP proponents argue that both the window increase and decrease functions may be altered, but in deciding whether to alter these functions, and in what way, they argue that a critical factor lies in the level of sensitivity to other concurrent network flows, and the ability to converge to stable resource allocations to various concurrent flows.

> "While such modifications might appear straightforward, it has been shown that they often negatively impact the behaviour of networks of TCP flows. High-speed TCP and BIC-TCP can exhibit extremely slow convergence following network disturbances such as the start-up of new flows; Scalable-TCP is a multiplicative-increase multiplicative-decrease strategy and as such it is known that it may fail to converge to fairness in drop-tail networks."
>
> Work-in-progress: **`draft-leith-tcp-htcp-01.txt`**

H-TCP argues for minimal changes to the window control functions, observing that in terms of fairness a flow with a large congestion window should, in absolute terms, reduce the size of their window by a larger amount that smaller-sized flows, as a means of readily establishing a dynamic equilibrium between established TCP flows and new flows entering the same network path.

H-TCP proposes a timer-based response function to window inflation, where for an initial period, the existing value of one segment per RTT is maintained, but after this period the inflation function is a function of the time since the last congestion event, using an order-2 polynomial function where the window increment in each RTT interval, $\alpha = (\frac{1}{2}T2 + 10T + 1)$, where T is the elapsed time since the last packet loss event. This equation is further modified by the current window reduction factor β where $\alpha' = 2 \times (1 - \beta) \times \alpha$.

The window reduction multiplicative factor, β, is based on the variance of the RTT interval , and β is set to RTT_{min} / RTT_{max} for the previous congestion interval, unless the RTT has a variance of more than 20 percent, in which case the value of ½ is used.

H-TCP appears to represent a further step along the evolutionary path for TCP, taking the adaptive window inflation function of HighSpeed TCP, using an elapsed timer as a control parameter as was done in Scalable TCP, and using the RTT ratio as the basis for the moderation of the window reduction value from Westwood.

FAST

FAST[10] is another approach to high-speed TCP. FAST is probably best viewed in context in terms of the per packet response of the various high speed TCP approaches, as indicated in the following Control and Response table:

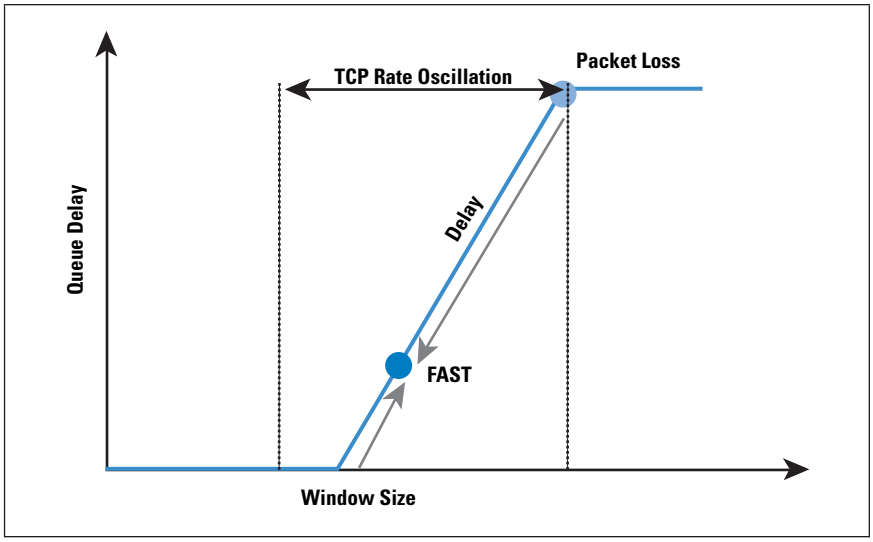
Type	Control Method	Trigger	Response
TCP	AIMD(1,0.5)	ACK response Loss response	W = W + 1/W W = W – W × 0.5
MulTCP	AIMD(N,1/2N)	ACK response Loss response	W = W + N/W W = W – W × 1/2N
HighSpeed TCP	AIMD(a(w), b(w))	ACK response Loss response	W = W + a(W)/W W = W – W × b(W)
Scalable TCP	MIMD(1/100, 1/8)	ACK response Loss response	W = W + 1/100 W = W – W × 1/8
FAST	RTT Variation	RTT	W = W × (base RTT/RTT) + α

All these approaches share a common structure of window adjustment, where the sender's window is adjusted according to a control function and a flow gain. TCP, MulTCP, HighSpeed TCP, Scalable TCP, BIC, CUBIC, Westwood, and H-TCP all operate according to a congestion measure that is based on ACK clocking and a packet loss trigger. What is happening in these models is that a bottleneck point on the network path has reached a level of saturation such that the bottleneck queue is full and packet loss is occurring. It is noted that the build up of the queue prior to packet loss would have caused a deterioration of the RTT.

This fact leads to the observation made by FAST, that another form of congestion signalling is one that is based on RTT variance, or cumulative queuing delay variance. FAST is based on this latter form of congestion signalling.

FAST attempts to stabilize the packet flow at a rate that also stabilizes queue delay, by basing its window adjustment, and therefore its sending rate, such that the RTT interval is stabilized. The window response function is based on adjusting the window size by the proportionate amount that the current RTT varies from the average RTT measurement. If the current RTT is lower than the average, then window size is increased, and if the current RTT is higher then window size is decreased. The amount of window adjustment is based on the proportionate difference between the two values, leading to the observation that FAST exponentially converges to a base RTT flow state. By comparison, conventional TCP has no converged state, but instead oscillates between the rate at which packet loss occurs and some lower rate (Figure 10).

Figure 10: TCP Response Function vs. FAST



FAST maintains an exponential weighted average RTT measurement and adjusts its window in proportion to the amount by which the current RTT measurement differs from the weighted average RTT measurement. It is harder to provide a graph of a simulation of FAST as compared to the other TCP methods, and the more instructive material has been gathered from various experiments using FAST.

XCP — End-to-End and Network Signalling

It is possible to also call in the assistance of the routers on the path and call on them to mark packets with signaling information relating to current congestion levels. This approach was first explored with the concept of ECN, or *Explicit Congestion Notification,* and has been generalized into a transport flow control protocol, called XCP,[11] where feedback relating to network load is based on explicit signals provided by routers relating to their relative sustainable load levels. Interestingly this digresses from the original design approach of TCP, where the TCP signaling is set up as effectively a heartbeat signal being exchanged by the end systems, and the TCP flow control process is based upon interpretation of the distortions of this heartbeat signal by the network.

XCP appears to be leading into a design approach where the network switching elements play an active role in end-to-end flow control, by effectively signalling to the end systems the current available capacity along the network path. This setup allows the end systems to respond rapidly to available capacity by increasing the packet rate to the point where the routers along the path signal that no further capacity is available, or to back off the sending rate when the routers along the path signal transient congestion conditions.

Whether such an approach of using explicit router-to-end host signals leads to more efficient very high-speed transport protocols remains to be determined, however.

Where Next?

The basic question here is whether we have reached some form of fundamental limitation of the TCP window-based congestion control protocol, or whether it is a case that the window-based control system remains robust at these speeds and distances, but that the manner of control signalling will evolve to adapt to an ever-widening range of speed extremes in this environment.

Rate-based pacing, as used in FAST can certainly help with the problem of the problem of guessing what are "safe" window inflation and reduction increments, and it is an open question as to whether it is even necessary to use a window inflation and deflation algorithm or whether it would be more effective to head in other directions, such as rate control, RTT stability control or adding additional network-generated information into the high-speed control loop. Explicit router-based signaling, such as described in XCP, allows for quite precise controls over the TCP session, although what is lost there is the adaptive ability to deploy the control system over any existing IP network.

However, across all these approaches, the basic TCP objectives remain the same: what we want is a transport protocol that can use the available network capacity as efficiently as possible—and as quickly as possible—minimizing the number of retransmissions and maximizing the effective data throughput.

We also want a protocol that can adapt to other users of the network, and attempt to fairly balance its use with competing claims for network resources.

The various approaches that have been studied to date all represent engineering compromises in one form or another. In attempting to optimize the instantaneous transfer rate the congestion control algorithm may not be responsive to other concurrent transport sessions along the same path. Or in attempting to optimize fairness with other concurrent sessions, the control algorithm may be unresponsive to available network path capacity. The control algorithm may be very unresponsive to dynamic changes in the RTT that may occur during the session because of routing changes in the network path. Which particular metrics of TCP performance are critical in a heterogeneous networking environment is a topic where we have yet to see a clear consensus emerging from the various research efforts.

However, we have learned a few things about TCP that form part of this consideration of where to take TCP in this very-high-speed world:

- The first lesson is that TCP has been so effective in terms of overall network efficiency and mutual fairness because everyone uses much the same form of TCP, with very similar response characteristics. If we all elected to use radically different control functions in each of our TCP implementations then it appears likely that we would have a poorly performing chaotic network subject to extended conditions of complete overload and inefficient network use.

- The second lesson is that a transport protocol does not need to solve media level or application problems. The most general form of transport protocol should not rely on characteristics of specific media, but should use specific responses from the lower layers of the protocol stack in order to function correctly as a transport system.
- The third lesson from TCP is that a transport protocol can become remarkably persistent and be used in contexts that were simply not considered in the original protocol design, so any design should be careful to allow generous margins of use conditions.
- The final lesson is one of fair robustness under competition. Does the protocol negotiate a fair share of the underlying network resource in the face of competing resource claims from concurrent transport flows?

Of all these lessons, the first appears to be the most valuable and probably the most difficult to put into practice. The Internet works as well as it does today largely because we all use the much same transport control protocol. If we want to consider some changes to this control protocol to support higher-speed flows over extended latency, then it would be perhaps reasonable to see if there is a single control structure and a single protocol that we can all use.

So deciding on a single approach for high-speed flows in the high-speed Internet is perhaps the most critical part of this entire agenda of activity. It is one thing to have a collection of differently controlled packet flows each operating at megabits-per-second flow rates on a multi-gigabit network, but it is quite a frightening prospect to have all kinds of different forms of flows each operating at gigabits per second on the same multigigabit network. If we cannot make some progress in reaching a common view of a single high-speed TCP control algorithm then it may indeed be the case that none of these approaches will operate efficiently in a highly diverse high-speed network environment.

Acknowledgment

I must acknowledge the patient efforts of Larry Dunn in reading through numerous iterations of this article, correcting the text and questioning some of my wilder assertions. Thanks Larry.

However, whatever errors may remain are, undoubtedly, all mine.

Further Reading

There is a wealth of reading on this topic, and here any decent search engine can assist. However if you are interested in this topic and want a starting reference that describes it in a very careful and structured manner, then I can recommend the following two sources as a good way to start exploring this topic to gain an overview of the current state of the art in this area:

- "HighSpeed TCP for Large Congestion Windows," S. Floyd, RFC 3649, December 2003.

  Floyd's treatment of this topic is precise, encompassing, and wonderfully presented. If only all RFCs were of this quality.

- Proceedings of the Workshops on Protocols for Fast Long-Distance Networks.

  These workshops have been held in:

  2003: **http://datatag.web.cern.ch/datatag/pfldnet2003/**

  2004: **http://www-didc.lbl.gov/PFLDnet2004/program.htm**

  2005: **http://www.ens-lyon.fr/LIP/RESO/pfldnet2005/**

References

[1] "Differentiated End-to-End Internet Services Using a Weighted Proportional Fair Sharing TCP," J. Crowroft and P. Oechcslin, ACM SIGCOMM *Computer Communication Review,* Volume 28, No. 3, pp. 53–69, July 1998.

[2] "HighSpeed TCP for Large Congestion Windows," S. Floyd, RFC 3649, December 2003.

[3] "Scalable TCP: Improving Performance in High-Speed Wide Area Networks," T. Kelly, ACM SIGCOMM *Computer Communication Review,* Volume 33, No. 2, pp. 83–91, April 2003.

[4] "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," L. Xu, K. Harfoush, and I. Rhee, *Proceedings of IEEE INFOCOMM 2004,* March 2004.

[5] "CUBIC: A New TCP-Friendly High-Speed TCP Variant," I. Rhee, L. Xu, **http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-paper.pdf**, February 2005.

[6] "TCP Westwood: Congestion Window Control Using Bandwidth Estimation," M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, *Proceedings of IEEE Globecom 2001,* Volume 3, pp. 1698–1702, November 2001.

[7] "Linux 2.4 Implementation of Westwood+ TCP with Rate-Halving: A Performance Evaluation over the Internet," A. Dell'Aera, L. A. Greco, and S. Mascolo, Tech. Rep. No. 08/03/S, Politecnico di Bari, **http://deecal03.poliba.it/mascolo/tcp%20westwood/Tech_Rep_08_03_S.pdf**

[8] "End-to-end Internet packet dynamics," V. Paxson, *Proceedings of ACM SIGCOMM 97,* pp. 139–152, 1997.

[9] "H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths," D. Leith, R. Shorten, Work in Progress, June 2005. Internet Draft: **draft-leith-tcp-htcp-00.txt**

[10] "FAST TCP: Motivation, Architecture, Algorithms, Performance," C. Jin, X. Wei, and S. H. Low, *Proceedings of IEEE INFOCOM 2004,* March 2004.

[11] "Congestion Control for High Bandwidth-Delay Product Networks," D. Katabi, M. Handley, and C. Rohrs, ACM SIGCOMM *Computer Communication Review,* Volume 32, No. 4, pp. 89–102, October 2002.

[12] "TCP Performance," Geoff Huston, *The Internet Protocol Journal,* Volume 3, No. 2, June 2000.

[13] "The Future for TCP," Geoff Huston, *The Internet Protocol Journal,* Volume 3, No. 3, September 2000.

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. He has been closely involved with the development of the Internet for almost two decades, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector, and has served time with Telstra, where he was the Chief Scientist in the company's Internet area. Geoff is currently the Internet Research Scientist at the Asia Pacific Network Information Centre (APNIC). He has been a member of the Internet Architecture Board, and currently co-chairs three Working Groups in the IETF. He is author of several Internet-related books. E-mail: **gih@apnic.net**