The Internet Protocol Journal

June 2006 Volume 9, Number 2

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

You can download IPJ back issues and find subscription information at: **www.cisco.com/ipj**

From The Editor

In our June 2000 issue we wrote: "Two protocols used in the Internet are so important that they deserve special attention: the *Internet Protocol* (IP) from which this journal takes its name, and the *Transmission Control Protocol* (TCP). IP is fundamental to Internet addressing and routing, while TCP provides a reliable transport service that is used by most Internet applications, including interactive Telnet, file transfer, electronic mail, and Web page access via HTTP. Because of the critical importance of TCP to the operation of the Internet, it has received much attention in the research community over the years. As a result, numerous improvements to implementations of TCP have been developed and deployed." We return to TCP in this issue with a look at its performance at gigabit speeds. Geoff Huston describes numerous research proposals related to TCP and discusses lessons learned by operators and researchers involved with this protocol.

My first encounter with the Internet (then called the ARPANET) took place in 1976 when I visited the *Norwegian Defence Research Establishment* (NDRE) at Kjeller, about 20 kilometers from Oslo, Norway. At NDRE, one of the researchers, named Pål, showed me a teletype terminal that was connected through the ARPANET to a host computer at SRI International in Menlo Park, California. After a few minutes, the teletype started printing messages from someone called "Geoff" on the other end of the line. Pål typed back, passing on questions from myself about the weather in California and so on. I later learned that the host computer was a PDP-10 model KA10 running the TENEX operating system. TENEX could "link" two terminals together so that anything typed on one terminal would appear on the other, and conversely. This primitive "chat" system is the forerunner of today's *Instant Messaging* (IM) environment. David Strom gives an overview of the current state of IM solutions in our second article.

The article "Working with IP Addresses" in our last issue sparked several comments, some of which are included in our Letters to the Editor section. A few readers also noticed some errors in the article, so we have included the corrections in this issue. We very much appreciate your feedback. Please send your comments to: **ipj@cisco.com**

—Ole J. Jacobsen, Editor and Publisher
ole@cisco.com

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*.

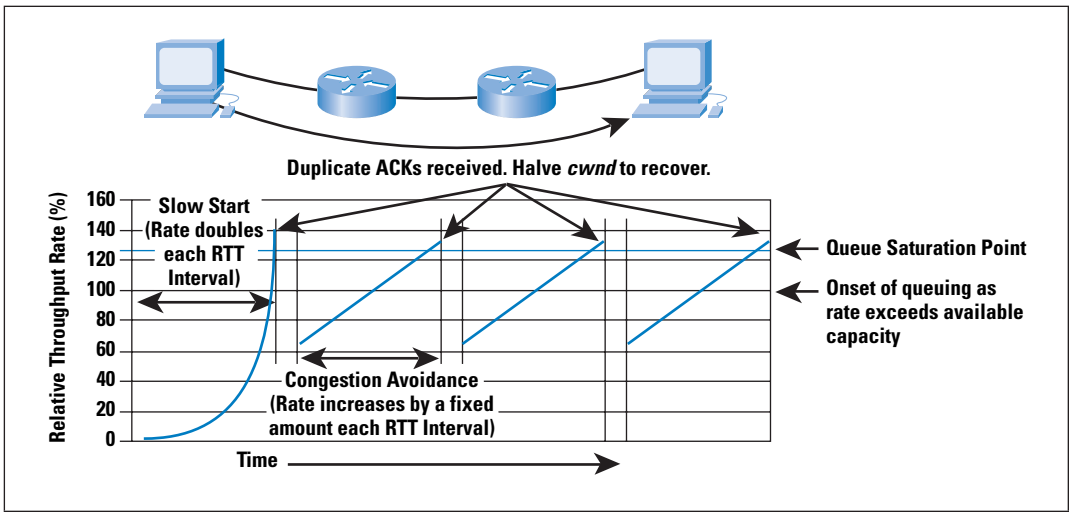
- *Slow start* mode is the initial mode of operation of TCP in any session, as well as its "reset" mode. In this mode, TCP sends two packets in response to each ACK packet that advances the sender's window. In approximate terms (*delayed* ACKs notwithstanding), this mode allows TCP to double its sending 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.

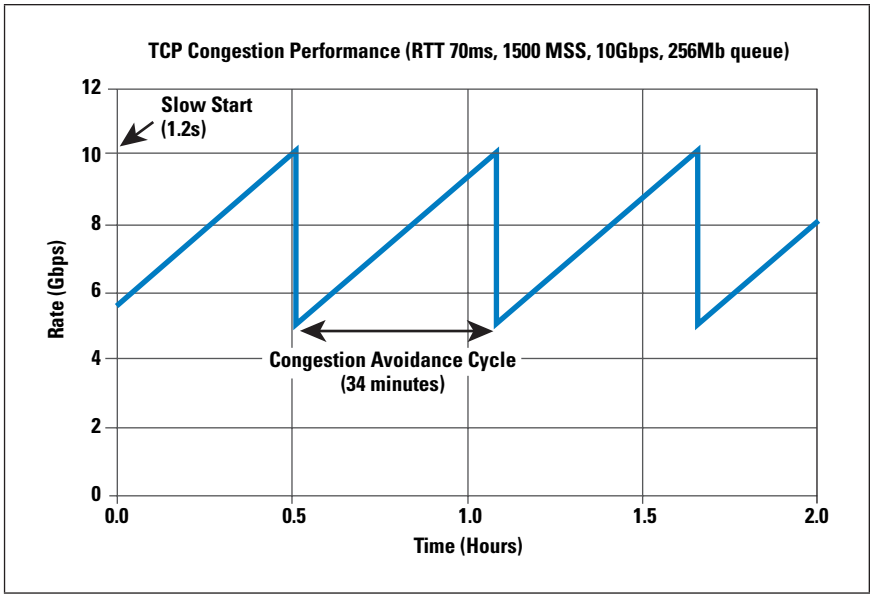
In a simple slow start model the sending speed doubles every 70 ms, so after 17 RTT intervals where the sending rate has doubled for each interval, or after some 1.2 seconds have elapsed, the transfer speed reaches 11.2 Gbps (assuming a theoretical host with sufficiently fast hardware components, sufficiently 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).

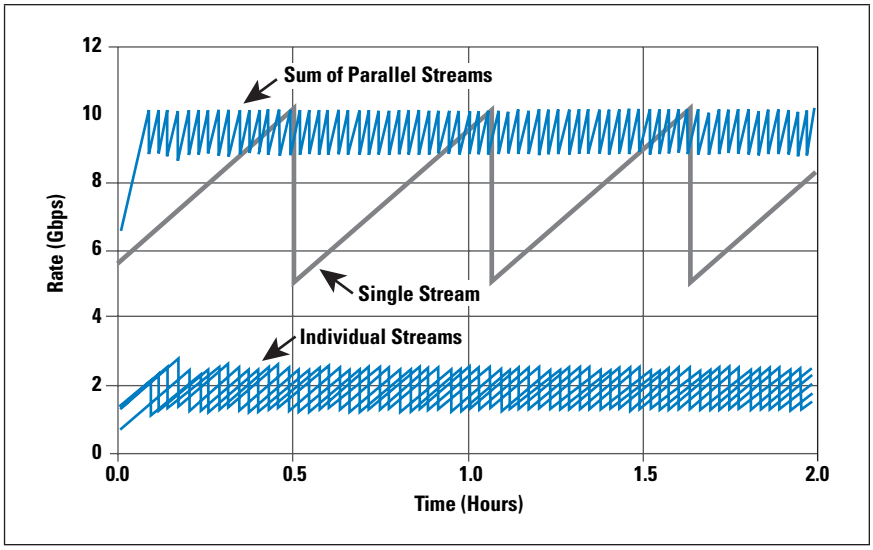
But for parallel TCP to operate correctly, we need to have already assembled all the data (or at a minimum know where all the data components are located). Where the data is being generated in real time (such as observatories or particle colliders) in massive quantities, there may be no choice but to 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.

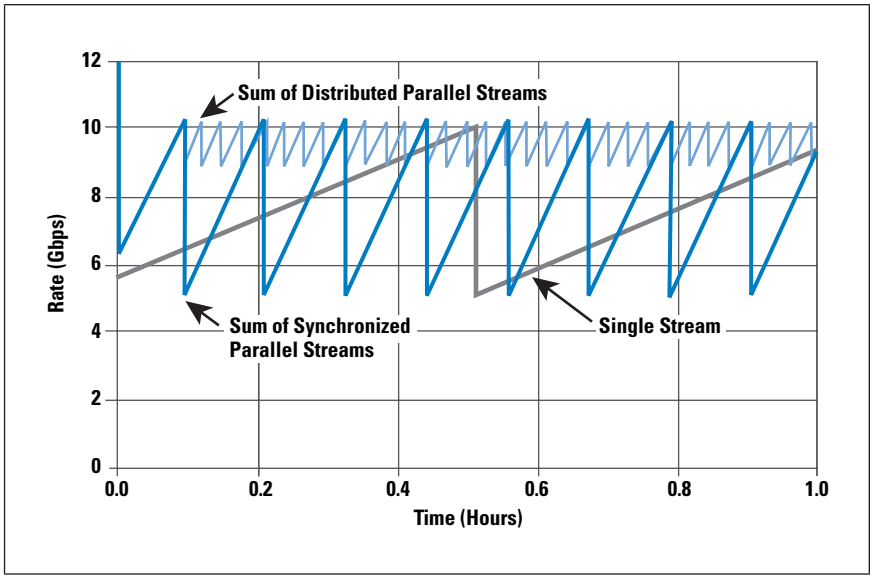


Figure 4: Comparison of Parallel TCP: Synchronized and Distributed Streams

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.

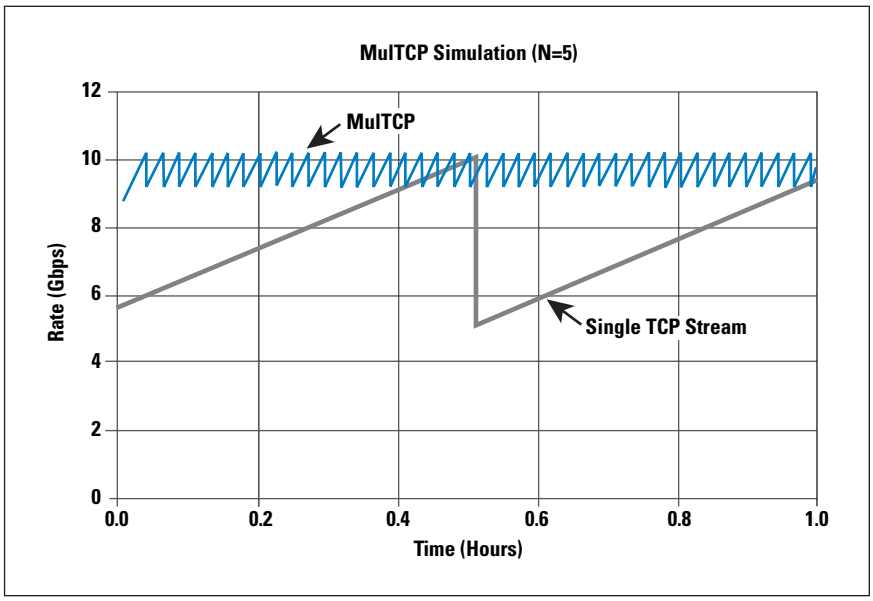
We also want TCP to behave sensibly in the face of other TCP sessions, so that it can share the network with other TCP sessions fairly.

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

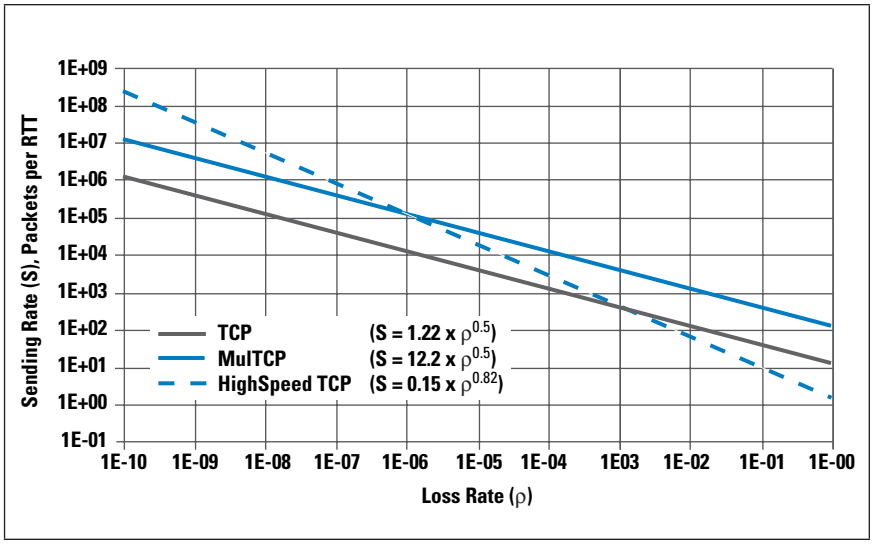
Another approach, described in [2], "HighSpeed TCP for Large Congestion Windows" looks at this from the perspective of the TCP rate equations, developed by Sally Floyd at ICIR.

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



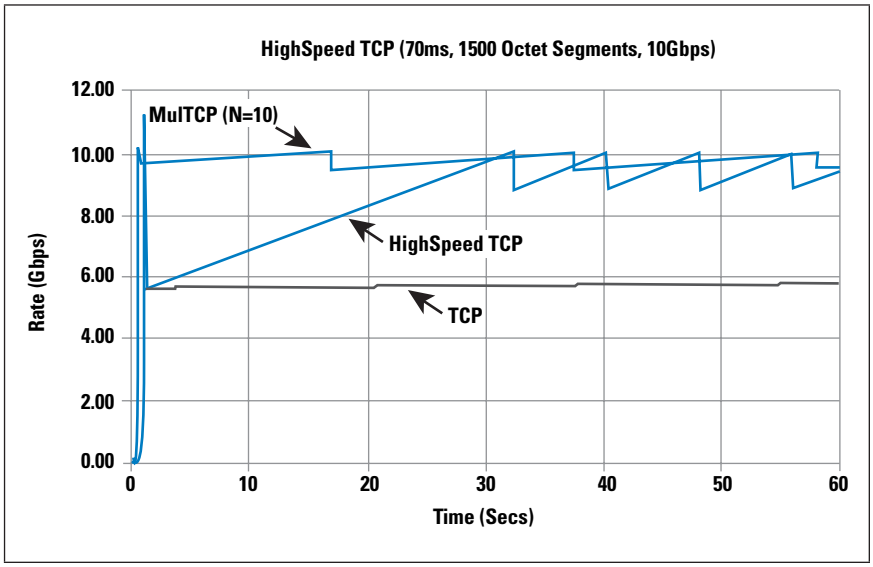
One way to look at the HighSpeed TCP proposal is that it operates in the same fashion as a turbocharger on an engine; the faster the engine is running, the higher the turbo-charged boost to the normal performance of the engine. Below a certain threshold value the TCP congestion avoidance function is 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.

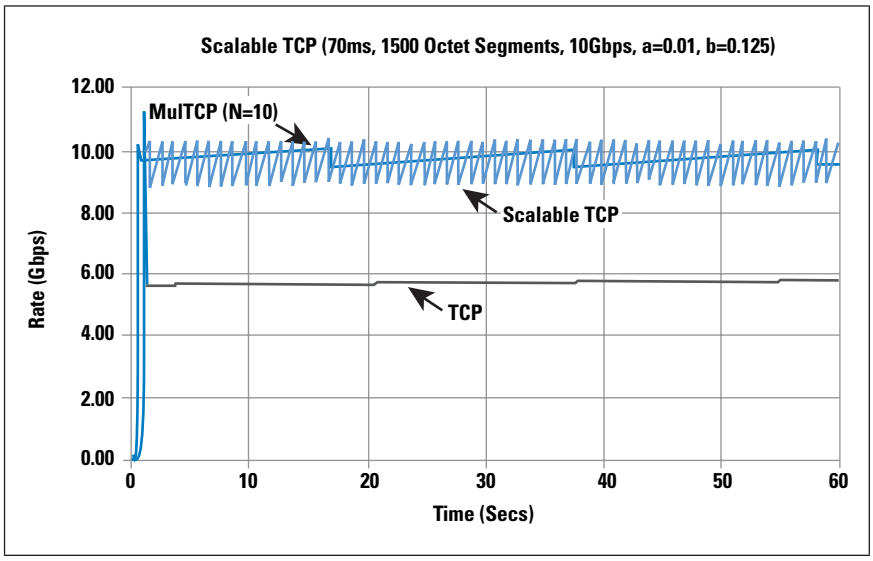
Scalable TCP[3], developed by Tom Kelly at Cambridge University, attempts to break the relationship between TCP window management and the RTT time interval. It does this by noting that in "conventional" TCP, the response to each ACK in congestion avoidance mode is to inflate the sender's congestion window 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.

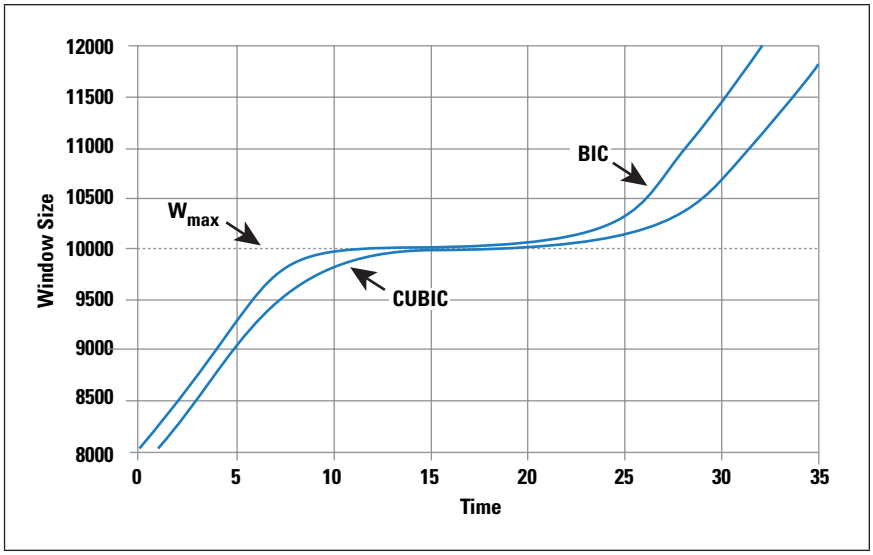
Binary Increase Congestion Control (BIC)[4] takes a different view, by assuming that TCP is actively searching for a packet sending rate that is on the threshold of triggering packet loss, and uses a binary chop search algorithm to achieve this efficiently.

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:

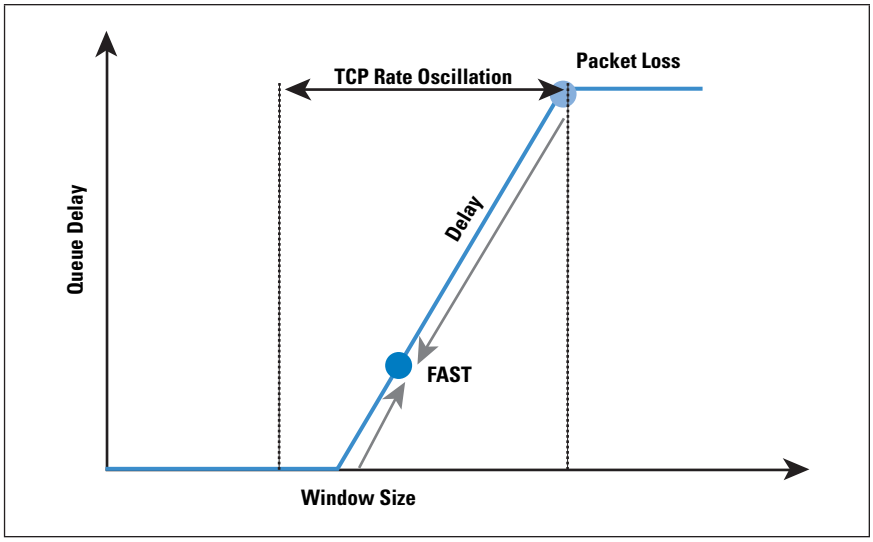
Type	Control Method	Trigger	Response
TCP	AIMD(1,0.5)	ACK response Loss response	W = W + 1/W W = W – W × 0.5
MulTCP	AIMD(N,1/2N)	ACK response Loss response	W = W + N/W W = W – W × 1/2N
HighSpeed TCP	AIMD(a(w), b(w))	ACK response Loss response	W = W + a(W)/W W = W – W × b(W)
Scalable TCP	MIMD(1/100, 1/8)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

How Instant Messaging Is Transforming the Enterprise Network

by David Strom

Instant Messaging (IM) has come of age and is close to becoming one of those protocols that offers something for everyone. Once the province of chatty teens looking to replace phone conversations with electronic ones, IM is now a corporate mainstay and part of a new breed of applications that are built around "presence detection," the ability to determine when someone—or something—is online and available to communicate.

Indeed, IM is rapidly spreading across the corporate world and becoming an able replacement for overflowing voicemail and e-mail inboxes that are clogged with spam and buried in irrelevant and non-time-sensitive postings. If you must get through to a busy corporate executive, IM is becoming the fastest and most effective method of communicating. Move over CrackBerry.

IM offers several benefits today, having taken some lessons learned by other Internet protocols of the past. First, it has a solid user and developer base. Second, it has a relatively simple building-block structure like the best of Internet protocols, with well-defined clients and servers. Third, interoperability efforts are beginning to pay off among the leading independent and private IM systems. Fourth, open-source rules are making inroads in all the right places. Fifth, Microsoft is a friend (for once) of IM and helping matters—rather than playing its usual monopolist role in this space, the company is actually encouraging future developments and interoperability. Finally, a new collection of advanced applications is taking hold that will take advantage of the existing Internet and IM infrastructure and create some very sophisticated IM applications.

Let's examine more closely where IM originated, where it is going, and what the specific implications are for each of these developments and for networking professionals. As a warning, this article by its very nature takes some positions on products and vendors. These opinions are solely those of the author, and they represent nothing wider or more inclusive.

User Base

The IM servers are operated by either public network or private entities. The major difference between the two is that the public systems operate across the Internet and can be accessed by any users who download the appropriate client software and create their own identity. Message traffic is usually transmitted in plaintext and without any encryption whatsoever.

The private IM systems are usually maintained by a corporate IT department and operate behind firewalls; they offer message encryption, message retention, and archiving; prepopulated buddy lists that are integrated into the corporate authentication and directory servers; and better security and privacy that are specific to a particular set of corporate users. These private systems are not available to the public and are designed strictly for employee communications or communications among particular trading partners of the corporation

The four most popular public IM systems are currently all in corporate hands: Microsoft, Yahoo, eBay/Skype, and AOL. Actually, we should make that five systems because AOL owns two separate networks, *AOL Instant Messenger* (AIM) and *I seek you* (ICQ). Introduced in November 1996, ICQ was actually the first general-purpose IM system combining presence or a list of contacts with the ability to send messages. Other popular systems include the open-source Jabber and Tencent QQ, the latter very popular in China. Estimates vary widely as to the total number of nonduplicated users—because many people have multiple accounts and use multiple systems—but it is safe to say that more than 150 million users are active across all these systems at any moment. The most recent estimates of active users are as follows:[1]

IM System	Estimate of Active Users
AIM	53 million active users
ICQ	15 million active users
Skype	10 million active users
MSN Messenger	29 million active users
Yahoo Messenger	21 million active users
Jabber	13.5 million enterprise users
Tencent QQ	10 million active users

Why IM Is So Popular for Businesses

But these numbers are more about individuals using IM. They hide the real story over the past several years, the rise of IM as a solid enterprise communications tool. Corporate IM usage has skyrocked the last several years, and one survey has found IM users in more than 50 percent of American corporations[2]. As mentioned earlier, there are public and private IM systems. The vast majority of the private IM systems are for institutional use for communications inside a company or among several suppliers, customers, and other trading partners.

The largest players in the private IM space are Microsoft Office Live Communications Server and IBM/Lotus' Sametime, although Jabber Corporation (not to be confused with the Jabber Software Foundation) is also gaining a strong following. We will discuss more about the role of open source in a moment, but first let's examine the reasons why IM has become so popular among so many business users.

First, workers have become more mobile and more difficult to track down. As secretarial support disappears and voicemail becomes more the norm, you want to know when people are actually at their desk—or laptop—these days. Staffs are more far-flung, and the global village becomes a lot smaller when you use IM to "talk" to someone halfway across the planet and get an immediate response. Finding someone who is available requires more than just making a phone call or exchanging e-mail messages. IM automatically tells you who is available—and who is not—at any given hour of the day.

Second, e-mail is no longer the productivity tool it once was because pipes are clogged with spam, viruses, and phishing attacks. Getting a quick response—that is, within minutes—through e-mail now seems so quaint, so "last year."

Third, IM enables better collaboration and a tighter sense of community. With IM, you can educate an entire team, give the team feedback in real time, develop relationships, and cement the team together. It is a nice antidote and countermeasure to connect all these home-based and remote workers.

Fourth, the next generation of IM is not just about text chats; it also offers solid integration with voice and video. Voice and video calling is now part of Microsoft, Yahoo, Apple, and AOL IM software as well as part of the Skype network, which pioneered the feature. These audio and video extensions are becoming more popular with the private Lotus and Microsoft systems as well.

Finally, the real-time features of IM and its ability to track someone down no matter where they are located are attractive to customers, partners, and suppliers that need a guaranteed method of communication. IM is becoming the critical technology ingredient for corporations that are looking for faster response times, tying their customers closer together, and enabling teleworkers to communicate across the globe.

Components

Following are some definitions and explanations for those unfamiliar with the world of IM. Every IM network is composed of clients, servers, and protocols to connect them.

Each IM client has three major pieces:

- A buddy list or roster of friends with whom you wish to communicate—The list is organized by groups that you specify, such as "friends," "work colleagues," "family," and so forth. The list indicates who is online, who is available to talk to, and who is offline or blocked by the user from communicating. Users organize their buddies in different ways and have complete control over the categories, naming conventions, and the like.

- A separate window that shows the text chats in process—Users type in this window and view the responses of their correspondents.
- Any additional features for video and audio chats and for file transfers between users

The last item bears some further discussion. All major IM products are moving beyond their roots of simple text chats toward more integrated and sophisticated communications, including real-time voice and video calls. Indeed, the mixture of *Voice over IP* (VoIP) and IM is a potent and popular one, accounting for the rapid uptake in Skype's adoption around the world. To use Skype as an example (although Yahoo has begun offering similar phone calling features in its IM client, and the others are soon to follow), users can make phone calls to the land-line phone numbers for a few pennies per minute—even calls to numbers in other countries. This is part of its attraction, along with voice mailboxes that are attached to a particular IM username.

The IM server maintains the directory of user accounts and keeps track of who is online, and in most cases routes messages among users. The major difference between an IM server and a *Simple Mail Transfer Protocol* (SMTP) e-mail server is that the IM server operates in real time, sending messages back and forth between two users as they finish typing a line of text. The servers also pass information in real time as to the availability of various users in the directory, when they come online and change their "status" message.

Users can typically set their availability in one of many different modes:

- Online and ready to receive messages
- Away from the computer, in which case correspondents receive a message saying so (or whatever the user wishes to be displayed)
- Unavailable or offline
- Blocked from anyone's view for privacy reasons

This status message can be changed at the user's discretion and is one of the main attractions for teens and other hypercommunicators. You can actually track what people are doing (or at least, saying that they are doing), by monitoring their status messages. (I am at the beach, I am taking a nap, I am at lunch, I am having coffee, and so forth.) For my teenaged daughter, this is one way she documents her life and one way that her friends can keep track of her—having a cell phone is not enough! There are numerous third-party add-ins to enhance your away message with clever graphics, hyperlinks to various Websites, and other effluvia as well.

The combination of instant access and persistent status indicator is at the core of why IM is such a powerful application. In a single window on your computer, you have a list of all your correspondents and can quickly determine who is online and who is not.

The blocking ability for some systems works universally, meaning that your presence is cloaked for everyone, as well as for specific users that you do not wish to communicate with or know your particular status, such as ex-spouses or ex-colleagues.

In most IM networks, you can be signed on from only one computer at any given moment. If you attempt to sign on from a second machine, you get an error message or your first computer is automatically logged out of the system. This is one way for the network to keep track of where you are located, because you can be in only one place at any given time.

Each server uses the TCP/IP Internet infrastructure and communicates with its clients over an assigned port number across the Internet. These ports can be blocked or proxied to different numbers, depending on the network administrator's policies toward IM traffic. Typical port numbers follow:

IM System	Port Numbers
ICQ	4000
AIM	5190–3
XMPP	5222–3
MSNP (Microsoft)	1863
YMSG (Yahoo)	5050
Skype	80, 443, and others

Notice an interesting thing about Skype's protocol: there is no single assigned port number. Users can set one of the ports in its configuration settings, but Skype uses a series of ports to communicate.[3] This setup suggests several concerns, which we address next.

The Dark Side

Although these are all compelling reasons for the rise of IM across the corporate network, all is not constructive with IM. This section discusses problems specifically germane to Skype and problems with all IM products in general.

When the Skype client is installed on a computer, it picks a random port to communicate with other Skype computers, using what is believed to be a form of *Request for Comments* (RFC) 3489[4]. This process is similar to many network-based games and peer-to-peer file-sharing products—no surprise because the developers of Skype worked on the Kazaa music file-sharing software. Because of its programming model, Skype is adept at traversing *Network Address Translation* (NAT) routers and can usually find a communications path to the outside world. Skype also encrypts all its message traffic, and this fact coupled with random port usage and its peer-to-peer programming model makes it look very similar to some malicious code that is unleashed across your network.

This is part of its charm and its challenge: network administrators who want to block Skype usage usually have a very difficult time figuring out how to do so[5], and may have to resort to third-party blocking products or clever configurations. One of the papers listed in [3] shows a way to block Skype using the popular open-source Squid caching proxy: not only do you have to prevent outbound *User Datagram Protocol* (UDP) connections over port 443, but you also must prevent connections to numeric IP addresses.

Although Skype has its own problems because of the way it is designed, there are several significant drawbacks to widespread adoption and deployment of any IM application. IM is not immune to infections, and just as its popularity is on the increase, so are ways to send malicious payloads and attacks too. What makes matters worse with IM versus say, e-mail, is its very instant nature: an infection can easily spread across a network in a matter of seconds, given that users are logged in, have long lists of users, and tend to think that any message coming from their respondents is more trusted than the average e-mail. In addition, Internet chat has long been a mechanism for controlling large-scale bot-nets of zombie computers, whose owners are unaware of such usage. Numerous virus authors have used exploits in Internet Relay Chat, for example, to control their villains across the Internet.

To avoid these problems, many corporations have either designed their own or are using one of several commercial IM protection products to screen incoming messages for particular patterns and methods of attack. The IM protection products work just like antivirus products work with e-mail messages: they download pattern files on a regular basis from a central server, and perform deep packet inspection across a perimeter to determine what is malicious and what is not.

Interoperability

Each public IM system is an island unto itself: users on one cannot easily communicate with users of another, unless one of two things happens:

- A user runs one of the multisystem client programs that allows them to sign in to multiple systems concurrently. Still, using these types of products means that just the user can communicate with his or her "buddies" across systems. Many mostly free products that enable this are available[6].
- A private IM operator can combine more than one protocol inside the IM server application. This approach means that clients need not know or care about other IM protocols, such as using Microsoft's Live Communications Server 2005[7].

But variables are changing on the interoperability scene to make life better for IM users. First, efforts are under way among the major operators to form better relationships with each other:

In October 2005, Yahoo and Microsoft announced plans to introduce interoperability between MSN and Yahoo Messenger by mid-2006, using *Session Initiation Protocols* (SIPs). In December 2005, AOL and Google announced a strategic partnership deal where Google Talk users can talk with AIM and ICQ users provided they have an identity at AOL.

Second, both Microsoft and Apple have made efforts to include multi-protocol IM clients as part of their desktop operating systems. Apple's iChat in its latest Mac OS 10.4 Tiger, as an example, now supports AIM, Google Talk, and Jabber. Microsoft has announced plans to support other networks in its next release of Windows Vista, expected later this year.

Finally, the private IM systems of Microsoft and Lotus both support multiple IM protocols, and are widening their support for others, making them more useful for corporations.

Still, with all this activity, the IM interoperability scene is pretty poor: think where e-mail was in the early 1990s with custom-crafted gateways and the like so that an MCIMail user could send messages to a CompuServe user.

Setting up two systems to talk to each other is neither simple nor obvious, and each pair of systems must be done separately. So to add Google Talk to Trillian, a user would need to provide the server host name (**`talk.google.com`**) and port number (5222). (By the way, GoogleTalk has the most helpful instructions on how to set up a variety of third-party applications to connect to its servers.)

But that is not all—even if a user follows these instructions to set up cross-system connections, most systems can exchange only plaintext messages. Video and voice chats between disparate systems are not generally supported, although Apple's iChat has done the best job so far in this arena. And even if users take the multiple-client approach, the structure of their buddy lists is not always maintained and sometimes is presented in a single group of buddies, rather than separated into the groups that were specified when initially setting up the IM account.

The other concern for cross-systems interoperability is a lack of support for privacy or online status. All of the IM systems have the ability to create blacklists, or lists of users that cannot view your online status. These blacklists are not necessarily preserved when running the multiple client systems.

The Rise of Open Source

There is hope on the interoperability scene, however, and that hope is spelled *open source*. The Jabber group of programmers is growing, and the community is aggressively establishing a more pluralistic IM society. These steps revolve around software using the protocol called the *Extensible Messaging and Presence Protocol* (XMPP), the IETF's formalization of the core protocols created by the Jabber open-source community in 1999, and contained in four RFCs[8, 9, 10, and 11].

Jeremie Miller developed the original Jabber server in 1998. Now the project has reached critical mass. Notable is the wide number of different server and client formulations that support XMPP. Jabber.com sells a commercial license, along with a combination of *General Public License* (GPL)-based licensed servers and other commercial versions. The project has supported the efforts of dozens of client implementations[12]. Last year, support reached a new milestone with Google Talk and more recently the Gizmo Project using these protocols.

Numerous efforts are under way with these clients to extend basic IM functions into new areas, including providing more sophisticated and secure communications, the ability to have multiple identities presented (**`david@strom.com`** for work colleagues, **`dstrom@gmail.com`** for personal communications) from the same IM client, and support for more interoperable communications between Jabber and private IM systems.

At the heart of XMPP is the *Extensible Markup Language* (XML) constructs and basic protocols. The core "transport" layer for XMPP is an XML streaming protocol that makes it possible to exchange fragments of XML between any two network endpoints. Authentication and channel encryption happen at the XML streaming layer using other IETF-standard protocols for *Simple Authentication and Security Layer*[13] and *Transport Layer Security*[14].

Servers can connect to each other for interdomain communications, using the form of address for each user as **`<user@domain>`**—similar to SMTP e-mail, and in many cases, the IM address is the same as one's Internet e-mail address to simplify things.

What is notable about using XMPP is that RFC 3921 also makes it possible to separate the messaging and presence functions if desired (although most deployments offer both). This feature is helpful when building applications-to-applications messaging that does not involve users typing text messages to each other, such as a server sending a network operator an alert when it detects a problem.

The Jabber Software Foundation develops extensions to XMPP through a standards process centered on *Jabber Enhancement Proposals* (JEPs), similar to the RFC process[15]. Currently, more than 30 active proposals have been developed, extending IM into bookmarks, delayed messaging, and other areas.

What Microsoft Is Doing

Microsoft is heavily involved in the IM scene in three important areas. The company operates one of the larger public IM networks, it includes an IM client as part of its Windows operating system, and it sells a private IM server that has some powerful interoperability features called *Live Communications Server* (LCS). What does this mean for the IM community? All good things. Microsoft's MSN and Skype are the more popular IM services outside of North America, and having Skype now a part of eBay is making Microsoft add competitive features such as voice and video chats to its public IM service. Microsoft has actually led the way on IM interoperability with LCS, a fact that can only motivate its competitors to include more pluralist IM offerings of their own. Finally, building in more support for IM in future versions of Windows will help popularize these applications even further.

It was not always this way. Earlier versions of Windows included something called Windows Messenger that was woefully underfeatured and had many bugs. But like so many early Microsoft efforts, technology has improved over time, and now the built-in software that comes with Windows is actually quite competitive with the public IM clients from AOL, Yahoo, and Skype.

Certainly, having Microsoft on one side and open-source efforts on the other is a nice way to encourage development and innovation in the IM arena, and we should expect more here in the future.

Building IM Applications

For most of this article we have addressed the one-to-one aspect of IM. However, IM is evolving into a much more important role, and that is one-to-many communications, and communications between applications instead of actual people. Many vendors have begun selling products in this space, and it is more interesting for several reasons:

First, IM is replacing other means for applications communications. It used to be the case that many network management applications used the *Simple Network Management Protocol* (SNMP) or SMTP protocols to send out their alerts. Now, many applications are using IM messages and taking advantage of the real-time nature of the protocol.

Second, the origins of IM go back to group chat sessions, so group collaboration tools make sense for new IM applications.

Third, even the closed public IM vendors have begun to open their programming interfaces, making it is easier for corporations to build new and sophisticated applications that are presence-aware, in some cases between two computer programs to communicate their status. AOL this year is one such example of opening its IM *application programming interface* (API) kimono, and of course Jabber has always been an open system that has helped lead more of these innovations.

One illustration is with the automotive giant Reynolds and Reynolds, which is using Jabber servers to monitor its own software status at the numerous automotive dealerships around the world. The IT department at Reynolds can quickly see if the company's software is down and take steps to get it working again.

Accredited Home Lenders is using IM to provide its loan brokers a secure and reliable means of communicating in real time with loan specialists to resolve problems with loan applications. And Ecreation built a virtual disk jockey for a Dutch radio station that also broadcasts over the Internet, allowing the station to take requests from listeners around the world through Microsoft's IM network.

Even traders have embraced IM. NetEnergy has been using IM for the past three years, and now negotiates trades between buyers and sellers of oil contracts using IM, decreasing errors and enabling faster communications.

Finally, IM figures prominently helping deaf and hard-of-hearing users communicate. In the era before IM, deaf users required a telephone relay operator to type the message to them and speak to the hearing callers. Go America has built a gateway to IM for its **i711.com** Website, so that deaf users can send messages directly to the operator.

Summary

We have tried to paint a comprehensive a picture of what IM is and where it is going. Certainly, the amount of messaging traffic using the various IM protocols is impressive, and will continue to grow as these new applications are created and as more people discover the advantages of using IM. In several instances IM has replaced voicemail for most internal communications, particularly at high-tech companies and places where real-time communications is important. Although IM is not without its problems, there are ways to protect networks from infection and abuse.

For Further Reading

[1] Nielsen//NetRatings, August 2005 study.

[2] Osterman Research survey:
http://www.ostermanresearch.com/results/surveyresults_0905.htm

[3] More details about the underlying Skype protocols, mechanisms for blocking its use, and other helpful tips and tricks for network administrators can be found at this page maintained by Salman A. Baset:
http://www1.cs.columbia.edu/~salman/skype/index.html

[4] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN—Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, March 2003.

[5] A dissection of the Skype protocol along with suggestions about how to block its use can be found in this paper by P. Biondi and F. Desclaux: "Silver Needle in the Skype."
http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf

[6] Adium and iChat for the Mac, Gaim for Windows and Linux, Trillian Pro for Windows, WebMessenger for Windows Mobile/Palm, and others.

[7] Microsoft's Live Communications Server 2005 includes its Public IM connector for an additional charge. Lotus' Sametime has had AIM connectivity for several years, and will support other IM networks later this year.

[8] P. Saint-Andre, ed., "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 3920, October 2004.

[9] P. Saint-Andre, ed., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," RFC 3921, October 2004.

[10] P. Saint-Andre, "Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)," RFC 3922, October 2004.

[11] P. Saint-Andre, "End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)," RFC 3923, October 2004.

[12] A list of software clients that support Jabber protocols can be found at:
http://www.jabber.org/software/clients.shtml

[13] J. Myers, "Simple Authentication and Security Layer (SASL)," RFC 2222, October 1997.

[14] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, January 1999.

[15] Jabber Enhancement proposals are listed at:
http://www.jabber.org/jeps/

DAVID STROM has been writing about Internet protocols and applications for nearly 20 years. Founding editor-in-chief for *Network Computing* magazine, he was most recently the editor-in-chief for **tomshardware.com** and related Websites. Strom has written two books on Internet e-mail (with the doyenne of POP, Marshall T. Rose) and home networking and thousands of magazine articles for most of the leading trade magazines in the IT, computing, and networking fields. He can be reached by e-mail at **david@strom.com**, or by IM: **davidstrom** (AIM and Skype) or **dstrom** (Yahoo, Google Talk, and MSN).

Letters to the Editor

Dear Editor,

In Russ White's "Working with IP Addresses" article (IPJ Volume 9, Number 1), he presents an example subnetting problem ("The Hardest Subnetting Problem") together with a worked solution. While useful as a reinforcement exercise for the rest of the article, care should be exercised before using the steps in the solution "as-is" in a real-world network configuration.

The main problem is that by packing subnets tightly together as shown, growth is restricted in order to guarantee that no address space is wasted. Worse, growth of host numbers on all but the smallest subnet requires renumbering of the subnet or all the smaller subnets allocated after it.

For example, the /26 subnet with 58 hosts will not accommodate more than another four hosts, less than 10-percent growth, without being renumbered.

Since renumbering a network is a nontrivial task even with the tools at our disposal, it is desirable to make it as infrequent as possible.[1]

Allowing for growth will likely but not necessarily waste some address space, but it is preferable to frequent renumbering. It turns out that this example has alternative arrangements of subnets that would permit growth of some subnets without the need to renumber and would lessen the amount of renumbering when it is required.

Using realistic estimates of future hosts rather than current numbers is a simple measure to decrease the frequency of renumbering required. This would also make it obvious that the entire allocation is close to exhaustion and can be exhausted by the need to accommodate as little as six hosts on two subnets that are near full capacity.

Constraints on the supply of IPv4 address space limits how much growth can be accommodated and requires taking a shorter-term rather than longer-term view of growth. For private RFC 1918[2] IP allocations (such as the one used in the example), this applies in only very large organisations, allowing a long-term view to be accommodated.

Unfortunately, the future is hard to predict with any degree of accuracy. In most cases needs for subnet allocation become gradually known over time rather than all at once. The consequences of incorrect estimation can be minimised by using an allocation scheme that allows for as much growth as possible in existing subnets while leaving as much room as possible for future allocations.

This scenario can be achieved by distributing the subnets evenly, weighted by size, across the available address space. The larger the subnet, the more room that needs to be left between it and other large networks. This is particularly important for subnets that are near to capacity. At least the sum of the sizes of neighbouring networks should be allowed. Space close to a network should be reserved for it to grow into, and the remaining space between can be allocated to smaller networks in a recursive fashion. Any allocations in the areas of likely growth should be reclaimable, and preferably these networks should be sparsely populated in order to limit the impact of renumbering on these networks. Working with a diagram of the address space, for example, a linear graph or a binary tree of the address space is a helpful aid.

A more systematic way of distributing the subnets evenly is to use *mirror-image* (MI) counting for allocating subnet numbers. This process is described in RFC 1219[3], but note that some aspects of subnet addressing have altered since this RFC was written (see RFC 1878[4]), so the description of mirror-image counting there and procedure text exclude subnet numbers that are now valid.

Using mirror-image counting is like normal counting starting from zero, except that the binary digits of the number are reversed. These numbers can be allocated as subnet numbers, starting from the most significant bit. Contrary to the example in RFC 1219, leading zeros (including the solitary zero in zero itself) should always be removed before the number is reversed.

Simplifying greatly, new subnets are allocated by incrementing the subnet number until a number is reached where a subnet of the required size can be accommodated or the subnet prefix becomes so long no subnets of the required size remain. If the prefix matches a common but shorter prefix, the subnet may be able to be allocated if we can lengthen the mask of the matching subnet prefix, freeing space from a previous allocation by reducing its maximum possible size. If the longest mask is always used when allocating subnets it is sufficient to just to skip matching prefixes. Note that the null prefix is common with all subsequent prefixes until its subnet mask is made smaller, extending the prefix.

The mask chosen is preferably the longest for the required subnet size—but can be as short as the length of the subnet prefix, because it can be adjusted later: made shorter if the subnetwork grows beyond its mask (if no later allocation has been made) or longer if a subnet sharing its prefix is allocated or increases size. The host number ignoring the subnet part must be allocated from 1.

As the number is incremented it grows from right to left, progressively enumerating subnets in smaller sizes. Since subnet numbers grow from right to left and host numbers from left to right, collision is delayed between the two. Allocating subnets in descending order of size is preferable in this procedure because it tends to reduce fragmentation of the address space.

The following table shows an example allocation using the sorted number of hosts in the example:

MI Number	Subnet Prefix	Network Size	Network Number	Prefix	Last Host Number	Max Host Number
(null)	00	64	0	/26	58	62
1	10	64	128	/26	177	190
01	010	32	64	/27	93	94
11	1100	16	192	/28	206	206
001 matches subnet prefix 00						
101 matches subnet prefix 10						
011	01100	8	96	/29	99	102

Note that the /28 and the /29 can grow simply by changing their netmask. A better allocation is possible if the third and fourth hosts in the sorted list are interchanged. In this case the three smallest networks would be able to grow without renumbering. Shortening a netmask is a much simpler operation than renumbering.

Of course in the real world, needs for subnet allocation do not conveniently arrive sorted in ascending order. If it happened that one of the two largest subnets was the fifth requiring allocation, fragmentation of the address space would require renumbering one of the three smallest networks to recover an address block of the necessary size.

Another point that may be worth mentioning is that most modern hosts and routers allow for multiple subnets to share the same physical subnet, allowing two smaller subnets to cover a range of addresses that would otherwise receive a single larger allocation. For example, a 40-host subnet can be allocated a /27 and a /28 rather than a /26.

—Andrew Friedman, Sydney, Australia
rbnsw-ipj@yahoo.com.au

Ed: Readers may wish to also peruse RFC 3531[5].

[1] P. Ferguson and H. Berkowitz, "Network Renumbering Overview: Why Would I Want It and What Is It Anyway?" RFC 2071, January 1997.

[2] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC 1918, February 1996.

[3] P. F. Tsuchiya, "On the Assignment of Subnet Numbers," RFC 1219, April 1991.

[4] T. Pummill and B. Manning, "Variable Length Subnet Table for IPv4," RFC 1878, December 1995.

[5] M. Blanchet, "A Flexible Method for Managing the Assignment of Bits of an IPv6 Address Block,"RFC 3531, April 2003.

The author responds:

Andrew is correct in stating that it is often better to try to account for future growth when assigning address space. There are many viable ways to allow for growth when allocating address spaces; hopefully, this topic will be covered more fully in a future article. I used the method in the article to illustrate how to employ the technique for working with IP addresses, rather than as an absolute best practice for allocating addresses.

—Russ White, Cisco Systems
riw@cisco.com

Dear Editor,

Russ White's article titled "Working with IP Addresses" was a nice refresher on how complicated working with IPv4 addresses has become. It should remind us all how we have gotten used to dealing with the operational expense of IPv4 address scarcity. The story about putting a frog in a pot of cold water comes to mind.

In any case, at the end of the article in the section titled "Working with IPv6 Addresses," I think the author tries too hard to fit the IPv6 address structure into the model for IPv4. Actually, it is a lot simpler.

The IPv6 address structure and textual representation was designed to avoid most of the complexities encountered in IPv4. The big differences follow:

- Addresses are represented in groups of hexadecimal digits instead of decimal digits. Hexadecimal avoids the need to convert the decimal digits to octal to find subnet boundaries. In hexadecimal there are four bits per character. This makes it easy to find the subnet boundary in an address; in many cases it is at a character boundary.
- Subnet prefix lengths are listed directly in decimal. There are no decimal subnet masks. This eliminates the need to convert decimal addresses to octets, convert the subnet masks to octets, apply the mask, and convert the result back to decimal—or to use the table and division methods described in the article.

The combination of these changes makes it much easier to work with IPv6 addresses. They are, of course, longer. The length has a few advantages besides a much larger Internet.

A byproduct of the larger address space is that most of the common subnet boundaries fall on hexadecimal digit boundaries; for example, using the example address in the article:

```
2002:FF10:9876:DD0A:9090:4896:AC56:0E01
```

The most common subnet boundary is 64 bits. The address and prefix is represented as:

```
2002:FF10:9876:DD0A:9090:4896:AC56:0E01/64
```

The subnet itself then follows:

```
2002:FF10:9876:DD0A::/64
```

The current common prefix allocated to a site is a /48. The site prefix is then:

```
2002:FF10:9876::/48
```

The current default allocation to an ISP is a /32. The ISP prefix is then:

```
2002:FF10::/32
```

These common prefix lengths can be derived directly without any need for decimal-to-octal conversions, table lookups, divisions, etc.

One of the other benefits of the larger addresses and a byproduct of IPv6 autoconfiguration is that the low-order 64 bits of an IPv6 address are reserved for the host address (called Interface Identifier in IPv6 terminology). This means that "The Hardest Subnetting Problem" described in the article is avoided completely. You can have as many hosts on a specific segment as you want in IPv6. There is no need to do this kind of calculation. This makes an initial network design trivial and, more importantly, makes later changes very easy. There is no need to redesign a subnet architecture because a few hosts need to be added to a subnet.

—Bob Hinden, Nokia
bob.hinden@nokia.com

The author responds:

Bob brings up many interesting points about IPv6, and the use of the IPv6 address space. While most IPv6 address spaces have prefix lengths that break on even octet boundaries today, we can't always count on this, for all time, so it is always good to have techniques to work with situations where the prefix length is not on an octet boundary when they do occur. As for the last problem, it is true that in all cases the subnet is the set of octets excluding the last 64 bits. But if we move the problem up one level, and ask: "What is the most efficient way to allocate out an existing /48 so customer A can get 10 subnets, customer B can get 20 subnets, etc. ?" we can see the same problem could occur at the next higher level.

—Russ White, Cisco Systems
riw@cisco.com

Corrections

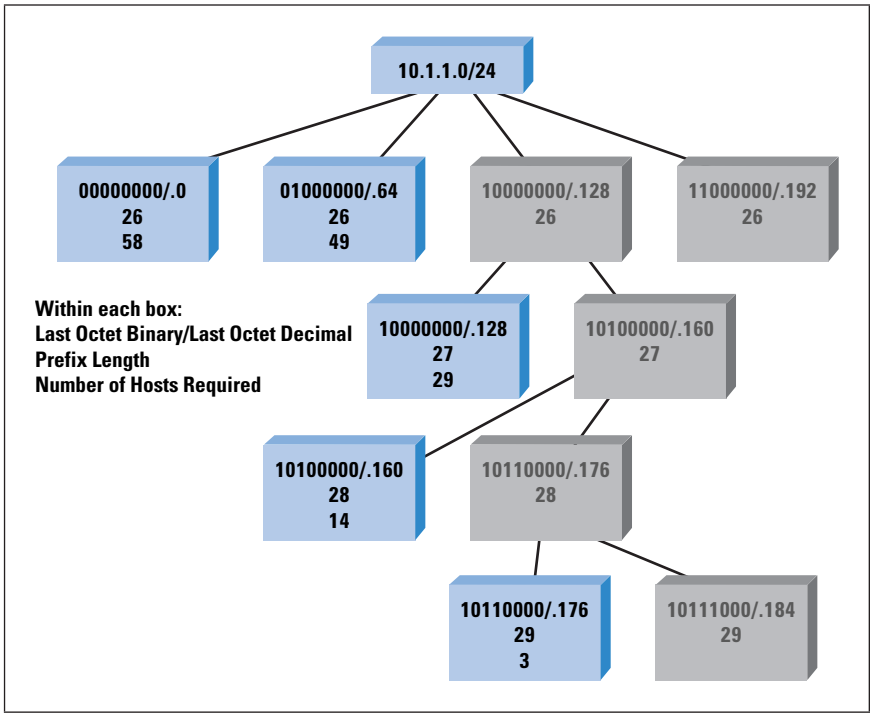
A few of our eagle-eyed readers have pointed us to some errors in IPJ, Volume 9, Number 1. The text below Figure 6 on page 29 and continuing at the top of page 30 should read as follows:

The figure shows four hosts with the addresses **10.1.0.1**, **10.1.0.2**, **10.1.0.3**, and **10.1.0.4**. Router A advertises **10.1.0.0/24**, meaning: "Any host within the address range **10.1.0.0** through **10.1.0.255** is reachable through me." Note that not all the hosts within this range exist, and that is okay—if a host within that range of addresses is reachable, it is reachable through Router A. In IP, the address that A is advertising is called a *network address*, and you can conveniently think of it as an address for the wire to which the hosts and router are attached, rather than a specific device.

For many people, the confusing part comes next. Router B is advertising **10.1.1.0/24**, which is another network address. Router C can combine—or *aggregate*—these two advertisements into a single advertisement. Although we have just removed the correspondence between the wire and the network address, we have not changed the fundamental meaning of the advertisement itself. In other words, Router C is saying: "Any host within the range of addresses from **10.1.0.0** through **10.1.1.255** is reachable through me." There is no wire with this address space, but devices beyond Router C do not know this, so it does not matter.

Also, Figure 8 on page 32 is reproduced here in its corrected form:

Figure 8: Subnet Chart



Book Review

Wireless Networking

Wireless Networking in the Developing World: A practical guide to planning and building low-cost telecommunications infrastructure, by Rob Flickenger et al., ISBN 1-4116-7837-0, 234 pages, Limehouse Book Sprint Team, January 2006. **http://wndw.net**

To quote from the book's Website:

"This book was created by a team of individuals who each, in their own field, are actively participating in the ever-expanding Internet by pushing its reach farther than ever before. Over a period of a few months, we have produced a complete book that documents our efforts to build wireless networks in the developing world."

Even though I don't live and work in what is commonly regarded as part of the developing world, I found this to be a unique and informative book, as its practical descriptions of wireless networking have application in many environments.

Given the widespread availability of the raw materials of computers, open-source software, Wi-Fi equipment, various pieces of recycled kitchenware, scrap metal, and plastic, and a wealth of online information resources, it is possible to construct inexpensive high-speed wireless network systems almost anywhere these days. However, perhaps the most visible missing component of the overall picture, but also the most valuable, is a practical path through this wealth of information on how to construct wireless networks, and a path that is based on the recent experiences of others who have constructed cost-effective and practical wireless networks in communities in the developing world. This book sets out to meet that goal.

Organization

The book starts with a description of radio physics covering the basics of the topic. It builds upon this a description of the typical radio design trade-offs between information capacity and radio penetration, and describes the commonly encountered factors of absorption, reflection, diffraction, and interference. I found the practical approach to Fresnel zone calculation and the description of the relationship between distance and antenna height so well done that I was tempted to embark on the design of a neighborhood Wi-Fi straightaway!

The chapter on network design is somewhat of a hybrid section, covering a mix of physical layout of a wireless network and TCP/IP considerations. There were the usual summaries of IP address structure and an introduction to routing.

Study of the deployment of the *Optimized Link State Routing* (OLSR) protocol is, however, more detailed. This is a link state routing protocol that is open-source, supportable by Linux-based access points, and accommodates link quality metrics into the routing protocol metric. I found the consideration of the link budget in this section a useful practical description of the considerations that are unique to the wireless world, and the worked examples are excellent, together with some useful references to online tools. This chapter is relatively dense, and many topics are covered in a relatively short space. I suspect that an interested reader would want to drill down further before feeling confident enough to manage a service network, but some carefully chosen references to further reading are there, so that the reader can follow up this introductory material with more specialized references.

The section on antennas and transmission lines was also well-structured. I had heard of using cylindrical cans as Wi-Fi antennas, but knew little of the detail of how to actually do it. This book not only explains their design, but provides a step-by-step illustrated guide to their construction. It also provides a good description of what is involved in outdoor installation of wireless equipment. The consideration of commercial solutions as compared to the do-it-yourself approach was carefully presented, as was the section devoted to security considerations.

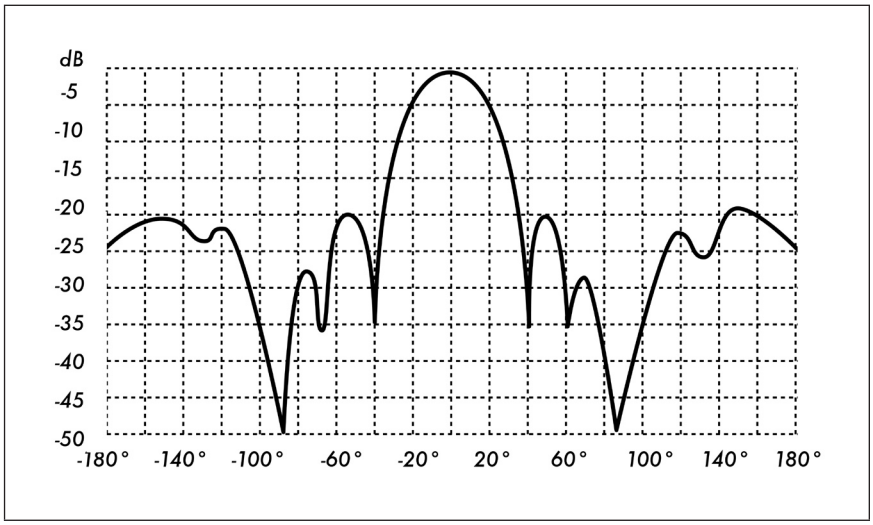
Aside from the technical considerations, the book also has some very interesting case studies of wireless networking projects, and was careful to include both success and failure stories. The issues in the developing world about combining technical capability with practical business solutions for communities that can be financially self-sustaining are indeed challenging, as the case studies show. They provide not only useful information about related experiences in setting up such network services, but also show how such projects can be assessed in a constructive manner.

Thoughtfully Written

Having spent some time working in this area myself as part of the ISOC *Developing Countries Workshop* training team, I have developed an appreciation of what constitutes truly useful and valuable training material, and this book is perhaps the best example I've seen yet. It is practical, helpful, technically accurate, and relatively complete in terms of coverage of material. Where the book does not dive into fine detail it provides useful references for further reading. The book is thoughtfully written in a simple non-nonsense style and does not hide behind technical jargon. Above all, it is material that can instill confidence that these networks can readily be built and operated by people like you and me.

I certainly would not call myself an expert after reading this book, but the next time a radio technician arrives in the office and starts talking about radiation patterns, front-to-back ratios, and the relative merits of omnis and yagis, at least I'll have an idea of what he is talking about. Even better, I might even be able to show him my own modest efforts in do-it-yourself Wi-Fi networking by then!

Rectangular plot of a Yagi Radiation Pattern from Chapter 4 of the book



Publishing Model

This is not a conventional technical book in the sense that it does not come with a conventional technical book price tag. The book is published in a manner as to be readily available in the developing world, so an online publication model has been used here. The PDF is freely available under a *Creative Commons* Attribution-Share-Alike 2.5 license at **http://wndw.net**, and they have managed to squeeze all 254 pages into an impressively small 1.92-MB file. You can find related resources and ways that you can assist in this project at **http://wndw.net**.

—Geoff Huston, APNIC
gih@apnic.net

Read Any Good Books Lately?

Then why not share your thoughts with the readers of IPJ? We accept reviews of new titles, as well as some of the "networking classics." In some cases, we may be able to get a publisher to send you a book for review if you don't have access to it. Contact us at **ipj@cisco.com** for more information.

Fragments

Internet Governance

The *World Summit on the Information Society* (WSIS) was held in two phases. The first phase took place in Geneva in December 2003, and the second phase took place in Tunis in November 2005. The so-called "WSIS Outcome Documents" are now available at:

http://www.itu.int/wsis/promotional/outcome.pdf

The follow-on to WSIS is called the *Internet Governance Forum* (IGF). The forum will hold its first meeting in Athens, Greece October 30th to November 2nd, 2006. For more information visit:

http://www.intgovforum.org/

The *Internet Society* (ISOC) played an active part in the WSIS process. You will find background information here:

http://www.isoc.org/isoc/conferences/wsis/index.shtml

DNS Root Name Servers Explained

Daniel Karrenberg of RIPE NCC has written two "Member Briefings" on the subject of DNS root servers that can be found on the ISOC Website:

http://www.isoc.org/briefings/019/
http://www.isoc.org/briefings/020/

Internationalized Domain Names

Internationalized Domain Names (IDNs) are, according to the ICANN Website, "...domain names represented by local language characters. Such domain names could contain letters or characters from non-ASCII scripts (for example, Arabic or Chinese). Many efforts are ongoing in the Internet community to make domain names available in character sets other than ASCII." ICANN has established an information area on its Website with links to more information about IDNs. See:

http://icann.org/topics/idn/

The ISP Column

Geoff Huston is well known to readers of this journal. He also hosts *The ISP Column* that can be found here:

http://www.isoc.org/pubs/isp/index.shtml

The Internet Protocol Journal is published quarterly by the Chief Technology Office, Cisco Systems, Inc.
www.cisco.com
Tel: +1 408 526-4000
E-mail: ipj@cisco.com

The Internet Protocol Journal, Cisco Systems
170 West Tasman Drive, M/S SJ-7/3
San Jose, CA 95134-1706
USA

ADDRESS SERVICE REQUESTED

PRSRT STD
U.S. Postage
PAID
PERMIT No. 5187
SAN JOSE, CA