The Internet Protocol Journal

June 2000 Volume 3, 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

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. In this issue, Geoff Huston takes a detailed look at TCP from a performance perspective and describes several enhancements to the original protocol. In a second article, Geoff will look at the challenges facing TCP in a rapidly growing and changing Internet, and describe work to further augment TCP.

Electronic mail is by far the most used of all Internet applications. The fundamental protocols for delivery and retrieval of e-mail have not changed much since the early days of the ARPANET, but as with TCP, many enhancements have been added to accommodate new uses of e-mail. Today, Internet e-mail supports international character sets, includes the ability to send file attachments, and allows roaming e-mail clients to authenticate themselves to servers. All of this has been made possible by continued development in the *Internet Engineering Task Force* (IETF). In our second article, Paul Hoffman of the Internet Mail Consortium gives an overview of Internet mail standards.

This is the second anniversary issue of *The Internet Protocol Journal* (IPJ). By now more than 10,000 people from virtually every country in the world have subscribed to the paper edition of IPJ. In order to serve our readers better, we are developing an online subscription system, which will be deployed in July 2000. With this new system you will be able to modify your mailing address as well as select your preferred delivery method for the journal. You can choose to receive IPJ on paper, or be notified via e-mail when a new issue becomes available on line. More information about this new system can be found on our Web site at **`www.cisco.com/ipj`**. We would love to hear your feedback on this system and any other aspect of IPJ. Please send your comments to **`ipj@cisco.com`**

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

TCP Performance

by Geoff Huston, Telstra

The *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP) are both IP transport-layer protocols. UDP is a lightweight protocol that allows applications to make direct use of the unreliable datagram service provided by the underlying IP service. UDP is commonly used to support applications that use simple query/response transactions, or applications that support real-time communications. TCP provides a reliable data-transfer service, and is used for both bulk data transfer and interactive data applications. TCP is the major transport protocol in use in most IP networks, and supports the transfer of over 90 percent of all traffic across the public Internet today. Given this major role for TCP, the performance of this protocol forms a significant part of the total picture of service performance for IP networks. In this article we examine TCP in further detail, looking at what makes a TCP session perform reliably and well. This article draws on material published in the *Internet Performance Survival Guide*[1].

Overview of TCP

TCP is the embodiment of reliable end-to-end transmission functionality in the overall Internet architecture. All the functionality required to take a simple base of IP datagram delivery and build upon this a control model that implements reliability, sequencing, flow control, and data streaming is embedded within TCP[2].

TCP provides a communication channel between processes on each host system. The channel is reliable, full-duplex, and streaming. To achieve this functionality, the TCP drivers break up the session data stream into discrete segments, and attach a TCP header to each segment. An IP header is attached to this TCP packet, and the composite packet is then passed to the network for delivery. This TCP header has numerous fields that are used to support the intended TCP functionality. TCP has the following functional characteristics:

- *Unicast protocol:* TCP is based on a unicast network model, and supports data exchange between precisely two parties. It does not support broadcast or multicast network models.
- *Connection state:* Rather than impose a state within the network to support the connection, TCP uses synchronized state between the two endpoints. This synchronized state is set up as part of an initial connection process, so TCP can be regarded as a connection-oriented protocol. Much of the protocol design is intended to ensure that each local state transition is communicated to, and acknowledged by, the remote party.
- *Reliable:* Reliability implies that the stream of octets passed to the TCP driver at one end of the connection will be transmitted across the network so that the stream is presented to the remote process as the same sequence of octets, in the same order as that generated by the sender.

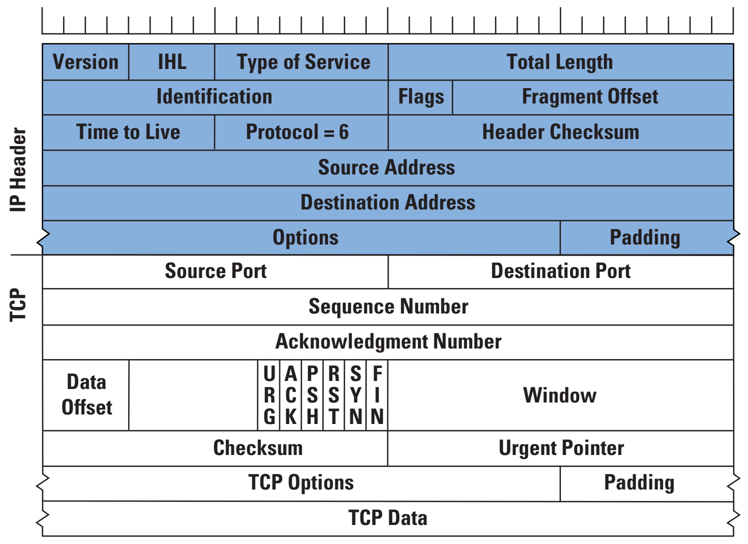
This implies that the protocol detects when segments of the data stream have been discarded by the network, reordered, duplicated, or corrupted. Where necessary, the sender will retransmit damaged segments so as to allow the receiver to reconstruct the original data stream. This implies that a TCP sender must maintain a local copy of all transmitted data until it receives an indication that the receiver has completed an accurate transfer of the data.

- *Full duplex:* TCP is a full-duplex protocol; it allows both parties to send and receive data within the context of the single TCP con–nection.
- *Streaming:* Although TCP uses a packet structure for network transmission, TCP is a true streaming protocol, and application-level network operations are not transparent. Some protocols explicitly encapsulate each application transaction; for every *write*, there must be a matching *read*. In this manner, the application-derived segmentation of the data stream into a logical record structure is preserved across the network. TCP does not preserve such an implicit structure imposed on the data stream, so that there is no pairing between *write* and *read* operations within the network protocol. For example, a TCP application may *write* three data blocks in sequence into the network connection, which may be collected by the remote reader in a single *read* operation. The size of the data blocks (segments) used in a TCP session is negotiated at the start of the session. The sender attempts to use the largest segment size it can for the data transfer, within the constraints of the maximum segment size of the receiver, the maximum segment size of the configured sender, and the maximum supportable non-fragmented packet size of the network path (path *Maximum Transmission Unit* [MTU]). The path MTU is refreshed periodically to adjust to any changes that may occur within the network while the TCP connection is active.
- *Rate adaptation:* TCP is also a rate-adaptive protocol, in that the rate of data transfer is intended to adapt to the prevailing load conditions within the network and adapt to the processing capacity of the receiver. There is no predetermined TCP data-transfer rate; if the network and the receiver both have additional available capacity, a TCP sender will attempt to inject more data into the network to take up this available space. Conversely, if there is congestion, a TCP sender will reduce its sending rate to allow the network to recover. This adaptation function attempts to achieve the highest possible data-transfer rate without triggering consistent data loss.

The TCP Protocol Header

The TCP header structure, shown in Figure 1, uses a pair of 16-bit source and destination *Port* addresses. The next field is a 32-bit *sequence number,* which identifies the sequence number of the first data octet in this packet. The sequence number does not start at an initial value of 1 for each new TCP connection; the selection of an initial value is critical, because the initial value is intended to prevent delayed data from an old connection from being incorrectly interpreted as being valid within a current connection. The sequence number is necessary to ensure that arriving packets can be ordered in the sender's original order. This field is also used within the flow-control structure to allow the association of a data packet with its corresponding acknowledgement, allowing a sender to estimate the current round-trip time across the network.

Figure 1: The TCP/IP Datagram



The *acknowledgment sequence number* is used to inform the remote end of the data that has been successfully received. The acknowledgment sequence number is actually one greater than that of the last octet correctly received at the local end of the connection. The *data offset* field indicates the number of four-octet words within the TCP header. Six single *bit flags* are used to indicate various conditions. URG is used to indicate whether the *urgent pointer* is valid. ACK is used to indicate whether the *acknowledgment* field is valid. PSH is set when the sender wants the remote application to *push* this data to the remote application. RST is used to *reset* the connection. SYN (for *synchronize)* is used within the connection startup phase, and FIN (for *finish*) is used to close the connection in an orderly fashion. The *window* field is a 16-bit count of available buffer space. It is added to the acknowledgment sequence number to indicate the highest sequence number the receiver can accept. The TCP *checksum* is applied to a synthesized header that includes the source and destination addresses from the outer IP datagram. The final field in the TCP header is the *urgent pointer,* which, when added to the sequence number, indicates the sequence number of the final octet of urgent data if the urgent flag is set.

Many options can be carried in a TCP header. Those relevant to TCP performance include:

- *Maximum-receive-segment-size option:* This option is used when the connection is being opened. It is intended to inform the remote end of the maximum segment size, measured in octets, that the sender is willing to receive on the TCP connection. This option is used only in the initial SYN packet (the initial packet exchange that opens a TCP connection). It sets both the maximum receive segment size and the maximum size of the advertised TCP window, passed to the remote end of the connection. In a robust implementation of TCP, this option should be used with path MTU discovery to establish a segment size that can be passed across the connection without fragmentation, an essential attribute of a high-performance data flow.
- *Window-scale option:* This option is intended to address the issue of the maximum window size in the face of paths that exhibit a high-delay bandwidth product. This option allows the window size advertisement to be right-shifted by the amount specified (in binary arithmetic, a right-shift corresponds to a multiplication by 2). Without this option, the maximum window size that can be advertised is 65,535 bytes (the maximum value obtainable in a 16-bit field). The limit of TCP transfer speed is effectively one window size in transit between the sender and the receiver. For high-speed, long-delay networks, this performance limitation is a significant factor, because it limits the transfer rate to at most 65,535 bytes per round-trip interval, regardless of available network capacity. Use of the window-scale option allows the TCP sender to effectively adapt to high-bandwidth, high-delay network paths, by allowing more data to be held in flight. The maximum window size with this option is 2^{30} bytes. This option is negotiated at the start of the TCP connection, and can be sent in a packet only with the SYN flag. Note that while an MTU discovery process allows optimal setting of the maximum-receive-segment-size option, no corresponding bandwidth delay product discovery allows the reliable automated setting of the window-scale option[3].
- *SACK-permitted option and SACK option:* This option alters the acknowledgment behavior of TCP. SACK is an acronym for *selective acknowledgment*. The SACK-permitted option is offered to the remote end during TCP setup as an option to an opening SYN packet. The SACK option permits selective acknowledgment of permitted data. The default TCP acknowledgment behavior is to acknowledge the highest sequence number of in-order bytes. This default behavior is prone to cause unnecessary retransmission of data, which can exacerbate a congestion condition that may have been the cause of the original packet loss. The SACK option allows the receiver to modify the acknowledgment field to describe noncontinuous blocks of received data, so that the sender can retransmit only what is missing at the receiver's end[4].

Any robust high-performance implementation of TCP should negotiate these parameters at the start of the TCP session, ensuring the following: that the session is using the largest possible IP packet size that can be carried without fragmentation, that the window sizes used in the transfer are adequate for the bandwidth-delay product of the network path, and that selective acknowledgment can be used for rapid recovery from line-error conditions or from short periods of marginally degraded network performance.

TCP Operation

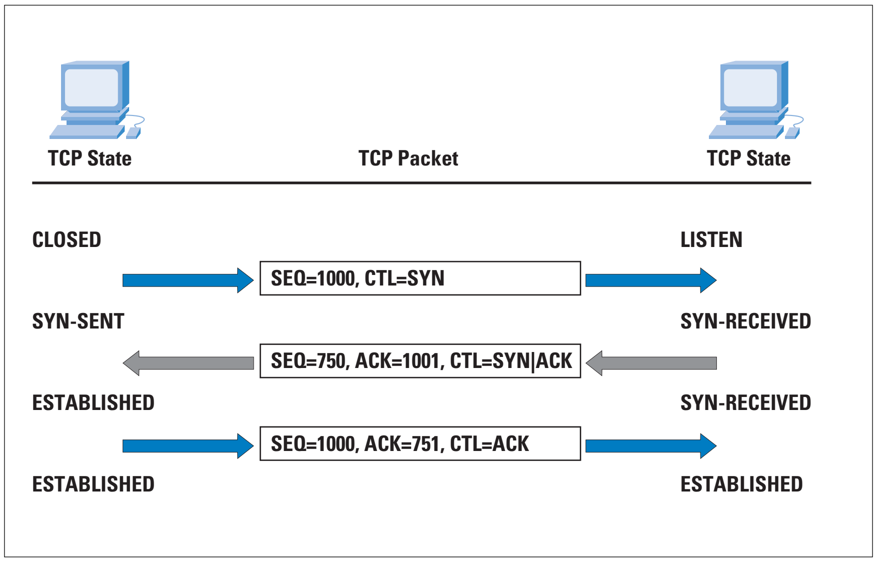
The first phase of a TCP session is establishment of the connection. This requires a *three-way handshake*, ensuring that both sides of the connection have an unambiguous understanding of the sequence number space of the remote side for this session. The operation of the connection is as follows:

- The local system sends the remote end an initial sequence number to the remote port, using a SYN packet.
- The remote system responds with an ACK of the initial sequence number and the initial sequence number of the remote end in a response SYN packet.
- The local end responds with an ACK of this remote sequence number.

The connection is opened.

The operation of this algorithm is shown in Figure 2. The performance implication of this protocol exchange is that it takes one and a half *round-trip times* (RTTs) for the two systems to synchronize state before any data can be sent.

Figure 2: TCP Connection Handshake



After the connection has been established, the TCP protocol manages the reliable exchange of data between the two systems. The algorithms that determine the various retransmission timers have been redefined numerous times. TCP is a *sliding-window* protocol, and the general principle of flow control is based on the management of the advertised window size and the management of retransmission timeouts, attempting to optimize protocol performance within the observed delay and loss parameters of the connection. Tuning a TCP protocol stack for optimal performance over a very low-delay, high-bandwidth LAN requires different settings to obtain optimal performance over a dialup Internet connection, which in turn is different for the requirements of a high-speed wide-area network. Although TCP attempts to discover the delay bandwidth product of the connection, and attempts to automatically optimize its flow rates within the estimated parameters of the network path, some estimates will not be accurate, and the corresponding efforts by TCP to optimize behavior may not be completely successful.

Another critical aspect is that TCP is an adaptive flow-control protocol. TCP uses a basic flow-control algorithm of increasing the data-flow rate until the network signals that some form of saturation level has been reached (normally indicated by data loss). When the sender receives an indication of data loss, the TCP flow rate is reduced; when reliable transmission is reestablished, the flow rate slowly increases again.

If no reliable flow is reestablished, the flow rate backs further off to an initial probe of a single packet, and the entire adaptive flow-control process starts again.

This process has numerous results relevant to service quality. First, TCP behaves *adaptively,* rather than *predictively*. The flow-control algorithms are intended to increase the data-flow rate to fill all available network path capacity, but they are also intended to quickly back off if the available capacity changes because of interaction with other traffic, or if a dynamic change occurs in the end-to-end network path. For example, a single TCP flow across an otherwise idle network attempts to fill the network path with data, optimizing the flow rate within the available network capacity. If a second TCP flow opens up across the same path, the two flow-control algorithms will interact so that both flows will stabilize to use approximately half of the available capacity per flow. The objective of the TCP algorithms is to adapt so that the network is fully used whenever one or more data flows are present. In design, tension always exists between the efficiency of network use and the enforcement of predictable session performance. With TCP, you give up predictable throughput but gain a highly utilized, efficient network.

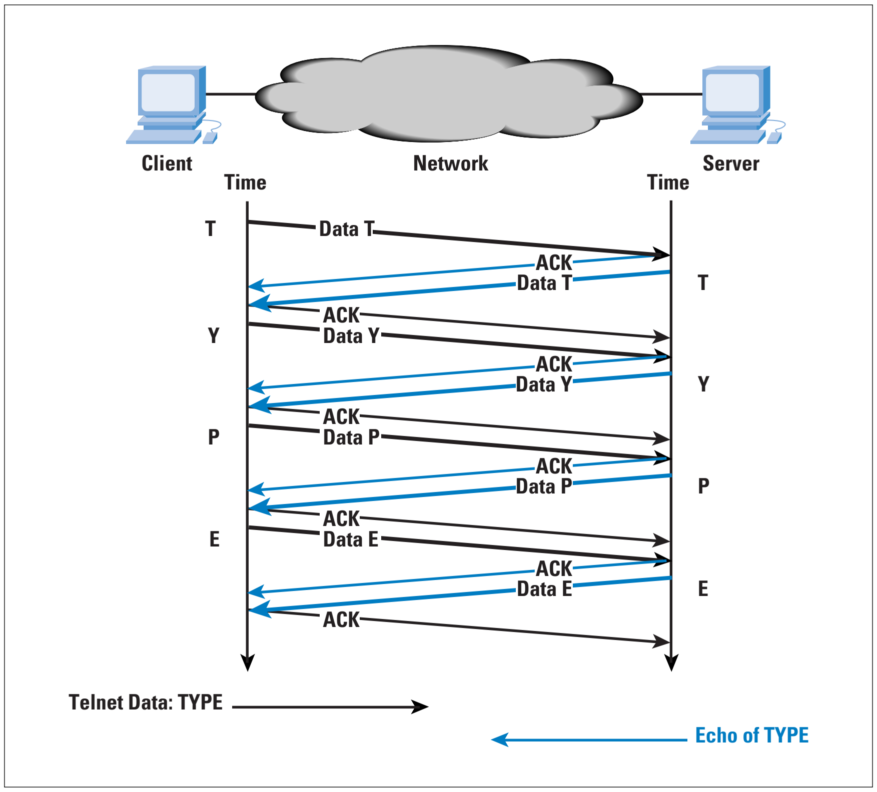
Protocol Performance

In this section we examine the transfer of data using the TCP protocol, focusing on the relationship between the protocol and performance. TCP is generally used within two distinct application areas: short-delay short data packets sent on demand, to support interactive applications such as *Telnet,* or *rlogin,* and large packet data streams supporting reliable volume data transfers, such as mail transfers, Web-page transfers, and *File Transfer Protocol* (FTP). Different protocol mechanisms come into play to support interactive applications, as distinct from short- and long-held volume transactions.

Interactive TCP

Interactive protocols are typically directed at supporting single-character interactions, where each character is carried in a single packet, as is its echo. The protocol interaction to support this is indicated in Figure 3. These 2 bytes of data generate four TCP/IP packets, or 160 bytes of protocol overhead. TCP makes some small improvement in this exchange through the use of *piggybacking,* where an ACK is carried in the same packet as the data, and *delayed acknowledgment,* where an ACK is delayed up to 200 ms before sending, to give the server application the opportunity to generate data that the ACK can piggyback. The resultant protocol exchange is indicated in Figure 4.

Figure 3: Interactive Exchange



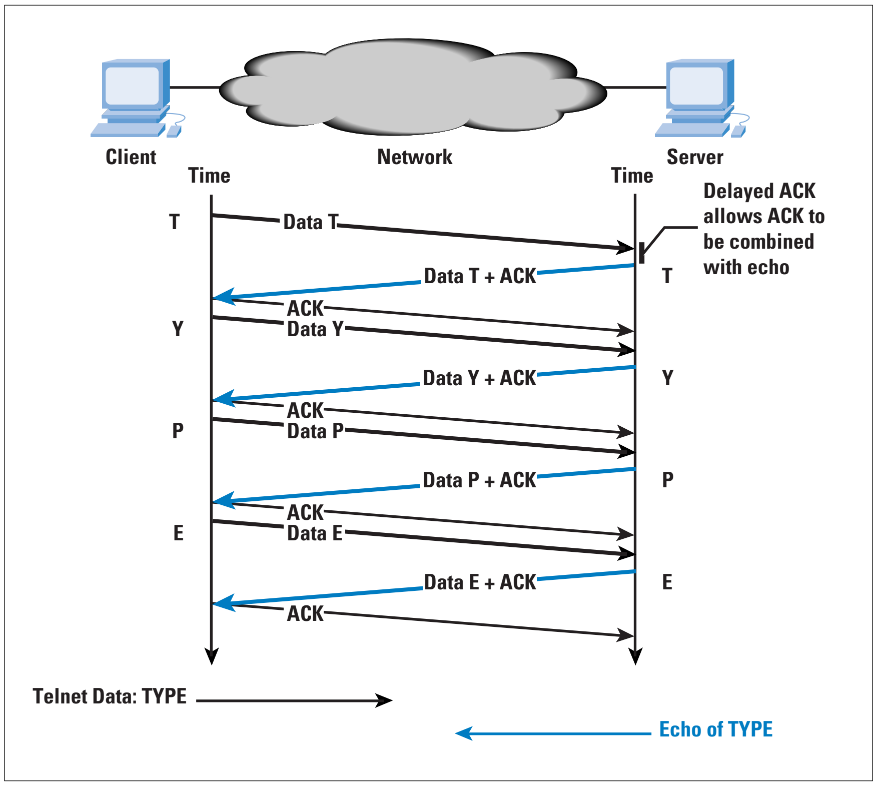


Figure 4: Interactive Exchange with Delayed ACK

For short-delay LANs, this protocol exchange offers acceptable performance. This protocol exchange for a single data character and its echo occurs within about 16 ms on an Ethernet LAN, corresponding to an interactive rate of 60 characters per second. When the network delay is increased in a WAN, these small packets can be a source of congestion load. The TCP mechanism to address this small-packet congestion was described by John Nagle in RFC 896[5]. Commonly referred to as the *Nagle Algorithm,* this mechanism inhibits a sender from transmitting any additional small segments while the TCP connection has outstanding unacknowledged small segments. On a LAN, this modification to the algorithm has a negligible effect; in contrast, on a WAN, it has a dramatic effect in reducing the number of small packets in direct correlation to the network path congestion level (as shown in Figures 5 and 6). The cost is an increase in session jitter by up to a round-trip time interval. Applications that are jitter-sensitive typically disable this control algorithm.

TCP is not a highly efficient protocol for the transmission of interactive traffic. The typical carriage efficiency of the protocol across a LAN is 2 bytes of payload and 120 bytes of protocol overhead. Across a WAN, the Nagle algorithm may improve this carriage efficiency slightly by increasing the number of bytes of payload for each payload transaction, although it will do so at the expense of increased session jitter.

Figure 5: WAN Interactive Exchange

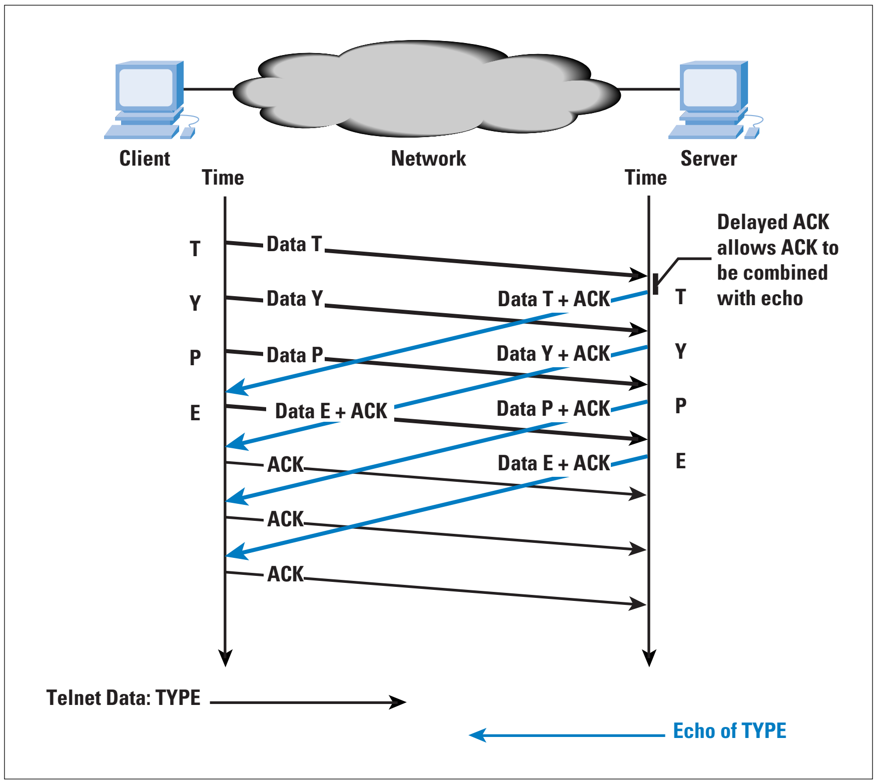
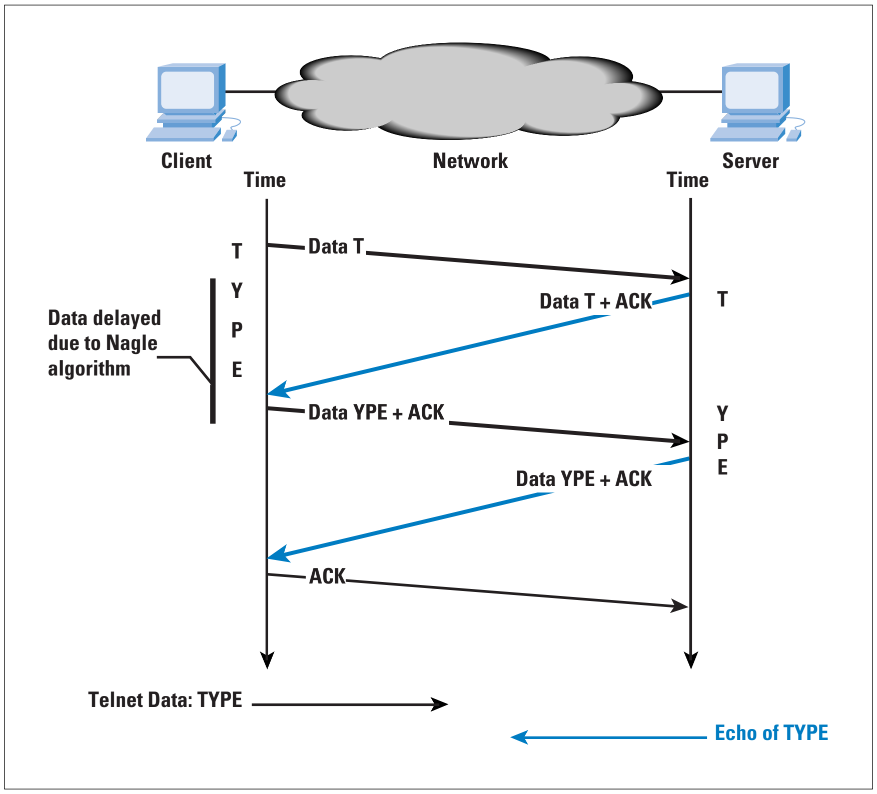


Figure 6: WAN Interactive Exchange with Nagle Algorithm



TCP Volume Transfer

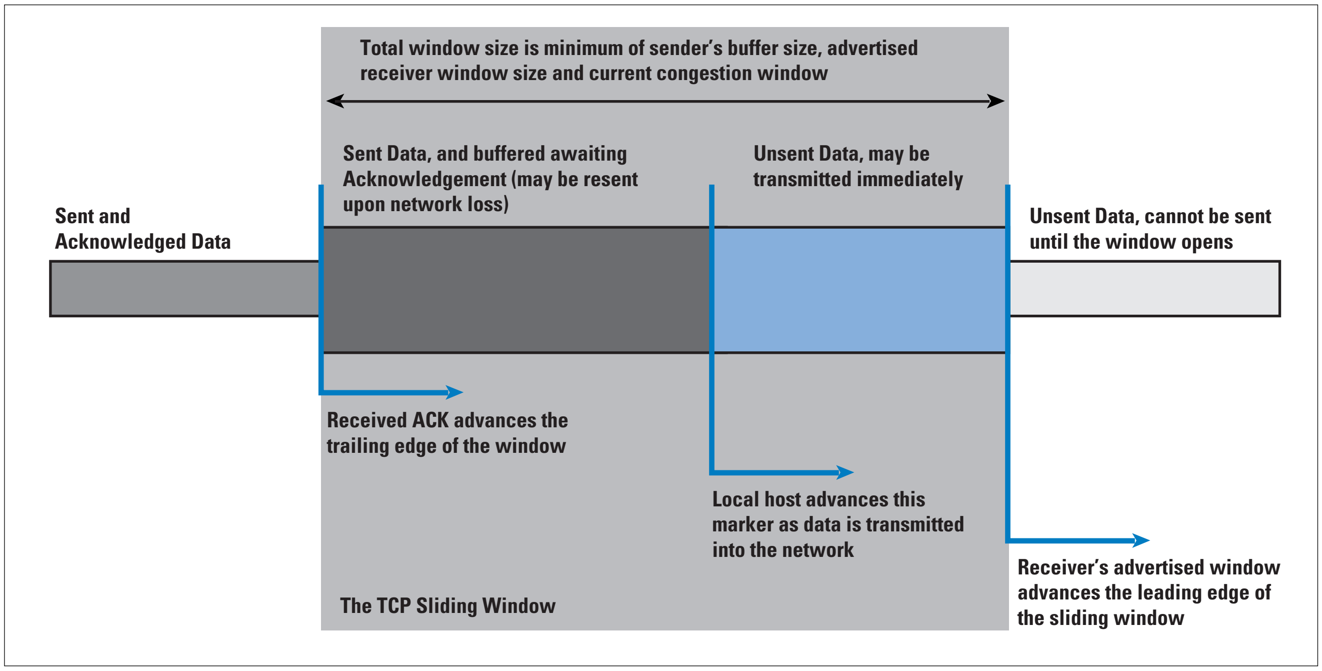
The objective for this application is to maximize the efficiency of the data transfer, implying that TCP should endeavor to locate the point of dynamic equilibrium of maximum network efficiency, where the sending data rate is maximized just prior to the onset of sustained packet loss.

Further increasing the sending rate from such a point will run the risk of generating a congestion condition within the network, with rapidly increasing packet-loss levels. This, in turn, will force the TCP protocol to retransmit the lost data, resulting in reduced data-transfer efficiency. On the other hand, attempting to completely eliminate packet-loss rates implies that the sender must reduce the sending rate of data into the network so as not to create transient congestion conditions along the path to the receiver. Such an action will, in all probability, leave the network with idle capacity, resulting in inefficient use of available network resources.

The notion of a point of equilibrium is an important one. The objective of TCP is to coordinate the actions of the sender, the network, and the receiver so that the network path has sufficient data such that the network is not idle, but it is not so overloaded that a congestion backlog builds up and data loss occurs. Maintaining this point of equilibrium requires the sender and receiver to be synchronized so that the sender passes a packet into the network at precisely the same time as the receiver removes a packet from the network. If the sender attempts to exceed this equilibrium rate, network congestion will occur. If the sender attempts to reduce its rate, the efficiency of the network will drop.

TCP uses a sliding-window protocol to support bulk data transfer (Figure 7). The receiver advertises to the sender the available buffer space at the receiver. The sender can transmit up to this amount of data before having to await a further buffer update from the receiver. The sender should have no more than this amount of data in transit in the network. The sender must also buffer sent data until it has been ACKed by the receiver. The send window is the minimum of the sender's buffer size and the advertised receiver window. Each time an ACK is received, the trailing edge of the send window is advanced. The minimum of the sender's buffer and the advertised receiver's window is used to calculate a new leading edge. If this send window encompasses unsent data, this data can be sent immediately.

Figure 7: TCP Sliding Window


Total window size is minimum of sender's buffer size, advertised receiver window size and current congestion window
Sent Data, and buffered awaiting Acknowledgement (may be resent upon network loss)
Unsent Data, may be transmitted immediately
Sent and Acknowledged Data
Unsent Data, cannot be sent until the window opens
Received ACK advances the trailing edge of the window
Local host advances this marker as data is transmitted into the network
Receiver's advertised window advances the leading edge of the sliding window
The TCP Sliding Window

The size of TCP buffers in each host is a critical limitation to performance in WANs. The protocol is capable of transferring one send window of data per round-trip interval. For example, with a send window of 4096 bytes and a transmission path with an RTT of 600 ms, a TCP session is capable of sustaining a maximum transfer rate of 48 Kbps, regardless of the bandwidth of the network path. Maximum efficiency of the transfer is obtained only if the sender is capable of completely filling the network path with data. Because the sender will have an amount of data in forward transit and an equivalent amount of data awaiting reception of an ACK signal, both the sender's buffer and the receiver's advertised window should be no smaller than the *Delay-Bandwidth Product* of the network path. That is:

$$\textit{Window size} \geq \textit{Bandwidth (bytes/sec)} \times \textit{Round-trip time (sec)}$$

The 16-bit field within the TCP header can contain values up to 65,535, imposing an upper limit on the available window size of 65,535 bytes. This imposes an upper limit on TCP performance of some 64 KB per RTT, even when both end systems have arbitrarily large send and receive buffers. This limit can be modified by the use of a window-scale option, described in RFC 1323, effectively increasing the size of the window to a 30-bit field, but transmitting only the most significant 16 bits of the value. This allows the sender and receiver to use buffer sizes that can operate efficiently at speeds that encompass most of the current very-high-speed network transmission technologies across distances of the scale of the terrestrial intercontinental cable systems.

Although the maximum window size and the RTT together determine the maximum achievable data-transfer rate, there is an additional element of flow control required for TCP. If a TCP session commenced by injecting a full window of data into the network, then there is a strong probability that much of the initial burst of data would be lost because of transient congestion, particularly if a large window is being used. Instead, TCP adopts a more conservative approach by starting with a modest amount of data that has a high probability of successful transmission, and then probing the network with increasing amounts of data for as long as the network does not show signs of congestion. When congestion is experienced, the sending rate is dropped and the probing for additional capacity is resumed.

The dynamic operation of the window is a critical component of TCP performance for volume transfer. The mechanics of the protocol involve an additional overriding modifier of the sender's window, the *congestion window,* referred to as *cwnd.* The objective of the window-management algorithm is to start transmitting at a rate that has a very low probability of packet loss, then to increase the rate (by increasing the *cwnd* size) until the sender receives an indication, through the detection of packet loss, that the rate has exceeded the available capacity of the network. The sender then immediately halves its sending rate by reducing the value of *cwnd,* and resumes a gradual increase of the sending rate. The goal is to continually modify the sending rate such that it oscillates around the true value of available network capacity. This oscillation enables a dynamic adjustment that automatically senses any increase or decrease in available capacity through the lifetime of the data flow.

The intended outcome is that of a dynamically adjusting cooperative data flow, where a combination of such flows behaves fairly, in that each flow obtains essentially a fair share of the network, and so that close to maximal use of available network resources is made. This flow-control functionality is achieved through a combination of *cwnd* value management and packet-loss and retransmission algorithms. TCP flow control has three major parts: the flow-control modes of *Slow Start* and *Congestion Avoidance,* and the response to packet loss that determines how TCP switches between these two modes of operation.

TCP Slow Start

The starting value of the *cwnd* window (the *Initial Window,* or IW) is set to that of the *Sender Maximum Segment Size* (SMSS) value. This SMSS value is based on the receiver's maximum segment size, obtained during the SYN handshake, the discovered path MTU (if used), the MTU of the sending interface, or, in the absence of other information, 536 bytes. The sender then enters a flow-control mode termed *Slow Start.*

The sender sends a single data segment, and because the window is now full, it then awaits the corresponding ACK. When the ACK is received, the sender increases its window by increasing the value of *cwnd* by the value of SMSS. This then allows the sender to transmit two segments; at that point, the congestion window is again full, and the sender must await the corresponding ACKs for these segments. This algorithm continues by increasing the value of *cwnd* (and, correspondingly, opening the size of the congestion window) by one SMSS for every ACK received that acknowledges new data.

If the receiver is sending an ACK for every packet, the effect of this algorithm is that the data rate of the sender doubles every round-trip time interval. If the receiver supports delayed ACKs, the rate of increase will be slightly lower, but nevertheless the rate will increase by a minimum of one SMSS each round-trip time. Obviously, this cannot be sustained indefinitely. Either the value of *cwnd* will exceed the advertised receive window or the sender's window, or the capacity of the network will be exceeded, in which case packets will be lost.

There is another limit to the slow-start rate increase, maintained in a variable termed *ssthresh*, or *Slow-Start Threshold*. If the value of *cwnd* increases past the value of *ssthresh,* the TCP flow-control mode is changed from Slow Start to congestion avoidance. Initially the value of *ssthresh* is set to the receiver's maximum window size. However, when congestion is noted, *ssthresh* is set to half the current window size, providing TCP with a memory of the point where the onset of network congestion may be anticipated in future.

One aspect to highlight concerns the interaction of the slow-start algorithm with high-capacity long-delay networks, the so-called *Long Fat Networks* (or LFNs, pronounced "elephants"). The behavior of the slow-start algorithm is to send a single packet, await an ACK, then send two packets, and await the corresponding ACKs, and so on. The TCP activity on LFNs tends to cluster at each epoch of the round-trip time, with a quiet period that follows after the available window of data has been transmitted. The received ACKs arrive back at the sender with an inter-ACK spacing that is equivalent to the data rate of the bottleneck point on the network path. During Slow Start, the sender transmits at a rate equal to twice this bottleneck rate. The rate adaptation function that must occur within the network takes place in the router at the entrance to the bottleneck point. The sender's packets arrive at this router at twice the rate of egress from the router, and the router stores the overflow within its internal buffer. When this buffer overflows, packets will be dropped, and the slow-start phase is over. The important conclusion is that the sender will stop increasing its data rate when there is buffer exhaustion, a condition that may not be the same as reaching the true available data rate. If the router has a buffer capacity considerably less than the delay-bandwidth product of the egress circuit, the two values are certainly not the same.

In this case, the TCP slow-start algorithm will finish with a sending rate that is well below the actual available capacity. The efficient operation of TCP, particularly in LFNs, is critically reliant on adequately large buffers within the network routers.

Another aspect of Slow Start is the choice of a single segment as the initial sending window. Experimentation indicates that an initial value of up to four segments can allow for a more efficient session startup, particularly for those short-duration TCP sessions so prevalent with Web fetches[6]. Observation of Web traffic indicates an average Web data transfer of 17 segments. A slow start from one segment will take five RTT intervals to transfer this data, while using an initial value of four will reduce the transfer time to three RTT intervals. However, four segments may be too many when using low-speed links with limited buffers, so a more robust approach is to use an initial value of no more than two segments to commence Slow Start[7].

Packet Loss

Slow Start attempts to start a TCP session at a rate the network can support and then continually increase the rate. How does TCP know when to stop this increase? This slow-start rate increase stops when the congestion window exceeds the receiver's advertised window, when the rate exceeds the remembered value of the onset of congestion as recorded in *ssthresh,* or when the rate is greater than the network can sustain. Addressing the last condition, how does a TCP sender know that it is sending at a rate greater than the network can sustain? The answer is that this is shown by data packets being dropped by the network. In this case, TCP has to undertake many functions:

- The packet loss has to be detected by the sender.
- The missing data has to be retransmitted.
- The sending data rate should be adjusted to reduce the probability of further packet loss.

TCP can detect packet loss in two ways. First, if a single packet is lost within a sequence of packets, the successful delivery packets following the lost packet will cause the receiver to generate a *duplicate* ACK for each successive packet The reception of these duplicate ACKs is a signal of such packet loss. Second, if a packet is lost at the end of a sequence of sent packets, there are no following packets to generate duplicate ACKs. In this case, there are no corresponding ACKs for this packet, and the sender's retransmit timer will expire and the sender will assume packet loss.

A single duplicate ACK is not a reliable signal of packet loss. When a TCP receiver gets a data packet with an out-of-order TCP sequence value, the receiver must generate an immediate ACK of the highest in-order data byte received. This will be a duplicate of an earlier transmitted ACK. Where a single packet is lost from a sequence of packets, all subsequent packets will generate a duplicate ACK packet.

On the other hand, where a packet is rerouted with an additional incremental delay, the reordering of the packet stream at the receiver's end will generate a small number of duplicate ACKs, followed by an ACK of the entire data sequence, after the errant packet is received. The sender distinguishes between these cases by using three duplicate ACK packets as a signal of packet loss.

The third duplicate ACK triggers the sender to immediately send the segment referenced by the duplicate ACK value (*fast retransmit*) and commence a sequence termed *Fast Recovery.* In fast recovery, the value of *ssthresh* is set to half the current send window size (the send window is the amount of unacknowledged data outstanding). The congestion window, *cwnd,* is set three segments greater than *ssthresh* to allow for three segments already buffered at the receiver. If this allows additional data to be sent, then this is done. Each additional duplicate ACK inflates *cwnd* by a further segment size, allowing more data to be sent. When an ACK arrives that encompasses new data, the value of *cwnd* is set back to *ssthresh,* and TCP enters congestion-avoidance mode. Fast Recovery is intended to rapidly repair single packet loss, allowing the sender to continue to maintain the ACK-clocked data rate for new data while the packet loss repair is being undertaken. This is because there is still a sequence of ACKs arriving at the sender, so that the network is continuing to pass timing signals to the sender indicating the rate at which packets are arriving at the receiver. Only when the repair has been completed does the sender drop its window to the *ssthresh* value as part of the transition to congestion-avoidance mode[8].

The other signal of packet loss is a complete cessation of any ACK packets arriving to the sender. The sender cannot wait indefinitely for a delayed ACK, but must make the assumption at some point in time that the next unacknowledged data segment must be retransmitted. This is managed by the sender maintaining a *Retransmission Timer.* The maintenance of this timer has performance and efficiency implications. If the timer triggers too early, the sender will push duplicate data into the network unnecessarily. If the timer triggers too slowly, the sender will remain idle for too long, unnecessarily slowing down the flow of data. The TCP sender uses a timer to measure the elapsed time between sending a data segment and receiving the corresponding acknowledgment. Individual measurements of this time interval will exhibit significant variance, and implementations of TCP use a smoothing function when updating the retransmission timer of the flow with each measurement. The commonly used algorithm was originally described by Van Jacobson[9], modified so that the retransmission timer is set to the smoothed round-trip-time value, plus four times a smoothed mean deviation factor[10].

When the retransmission timer expires, the actions are similar to that of duplicate ACK packets, in that the sender must reduce its sending rate in response to congestion. The threshold value, *ssthresh,* is set to half of the current value of outstanding unacknowledged data, as in the duplicate ACK case. However, the sender cannot make any valid assumptions about the current state of the network, given that no useful information has been provided to the sender for more than one RTT interval. In this case, the sender closes the congestion window back to one segment, and restarts the flow in slow start-mode by sending a single segment. The difference from the initial slow start is that, in this case, the *ssthresh* value is set so that the sender will probe the congestion area more slowly using a linear sending rate increase when the congestion window reaches the remembered *ssthresh* value.

Congestion Avoidance

Compared to Slow Start, congestion avoidance is a more tentative probing of the network to discover the point of threshold of packet loss. Where Slow Start uses an exponential increase in the sending rate to find a first-level approximation of the loss threshold, congestion avoidance uses a linear growth function.

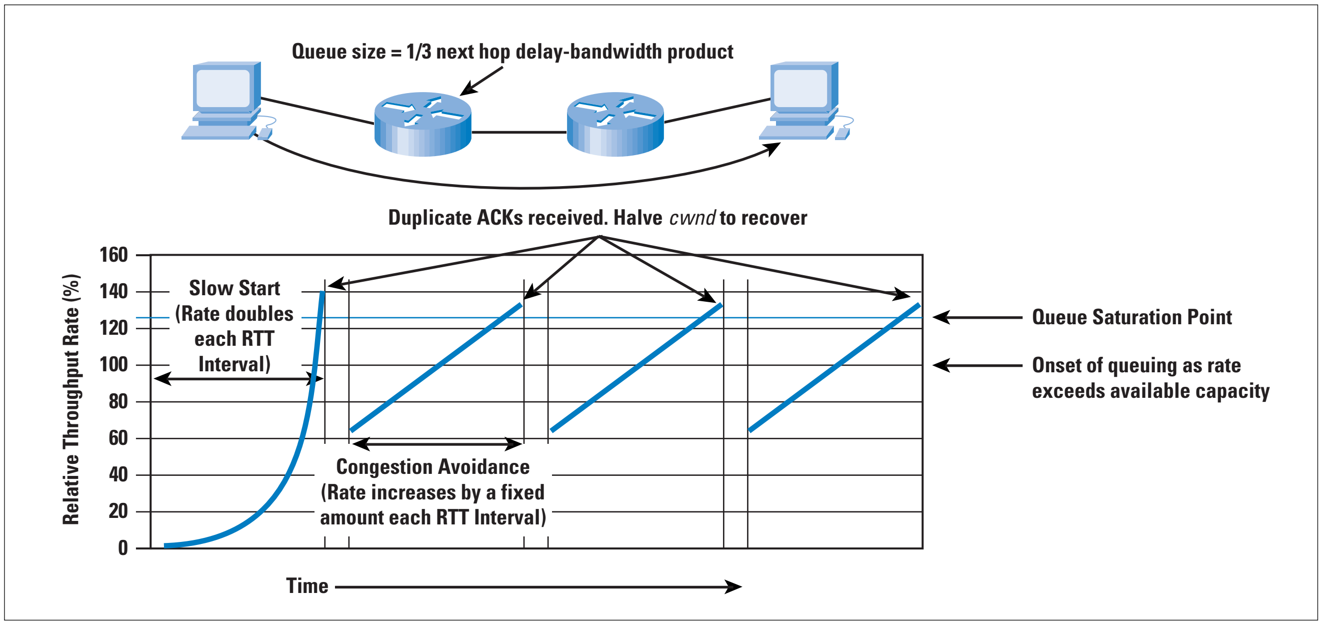
When the value of *cwnd* is greater than *ssthresh*, the sender increments the value of *cwnd* by the value $SMSS \times SMSS/cwnd$, in response to each received nonduplicate ACK[7], ensuring that the congestion window opens by one segment within each RTT time interval.

The congestion window continues to open in this fashion until packet loss occurs. If the packet loss is isolated to a single packet within a packet sequence, the resultant duplicate ACKs will trigger the sender to halve the sending rate and continue a linear growth of the congestion window from this new point, as described above in fast recovery.

The behavior of *cwnd* in an idealized configuration is shown in Figure 8, along with the corresponding data-flow rates. The overall characteristics of the TCP algorithm are an initial relatively fast scan of the network capacity to establish the approximate bounds of maximal efficiency, followed by a cyclic mode of adaptive behavior that reacts quickly to congestion, and then slowly increases the sending rate across the area of maximal transfer efficiency.

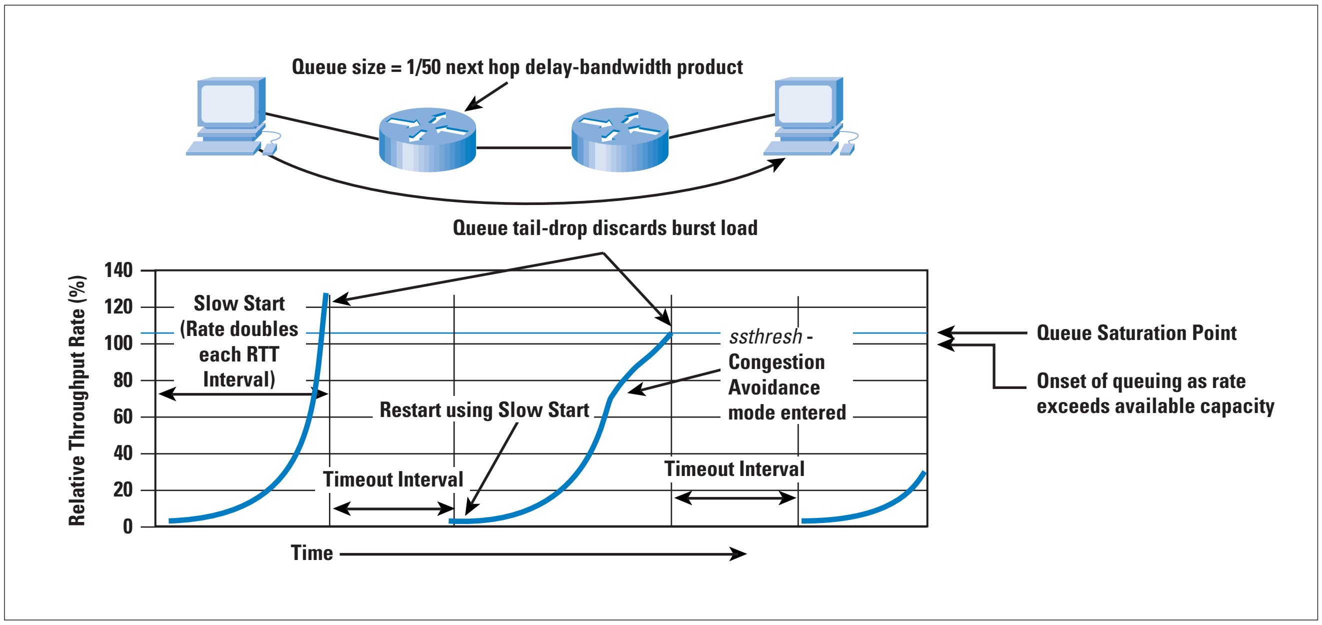
Packet loss, as signaled by the triggering of the retransmission timer, causes the sender to recommence slow-start mode, following a timeout interval. The corresponding data-flow rates are indicated in Figure 9.

Figure 8: Simulation of Single TCP Transfer


Queue size = 1/3 next hop delay-bandwidth product
Duplicate ACKs received. Halve *cwnd* to recover
160
140
120
100
80
60
40
20
0
Relative Throughput Rate (%)
Slow Start (Rate doubles each RTT Interval)
Congestion Avoidance (Rate increases by a fixed amount each RTT Interval)
Queue Saturation Point
Onset of queuing as rate exceeds available capacity
Time

The inefficiency of this mode of performance is caused by the complete cessation of any form of flow signaling from the receiver to the sender. In the absence of any information, the sender can only assume that the network is heavily congested, and so must restart its probing of the network capacity with an initial congestion window of a single segment. This leads to the performance observation that any form of packet-drop management that tends to discard the trailing end of a sequence of data packets may cause significant TCP performance degradation, because such drop behavior forces the TCP session to continually time out and restart the flow from a single segment again.

Figure 9: Simulation of TCP Transfer with Tail Drop Queue

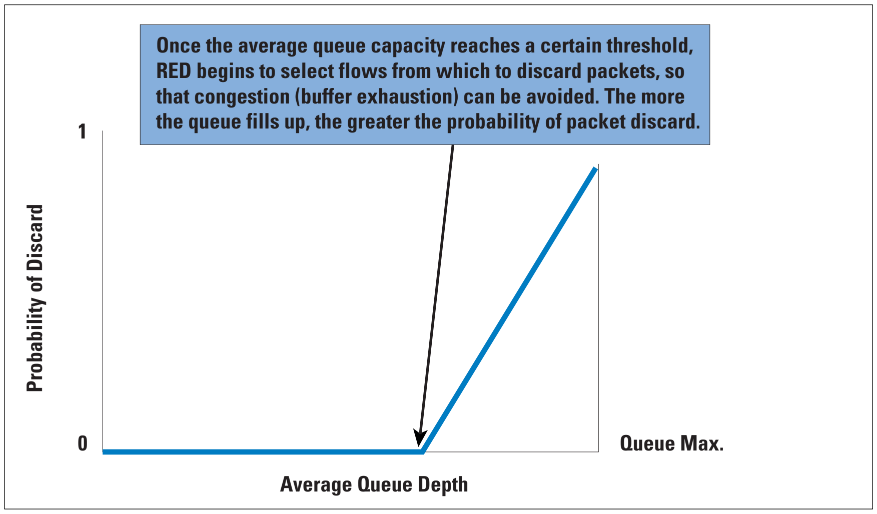

Queue size = 1/50 next hop delay-bandwidth product
Queue tail-drop discards burst load
140
120
100
80
60
40
20
0
Relative Throughput Rate (%)
Slow Start (Rate doubles each RTT Interval)
Restart using Slow Start
ssthresh - Congestion Avoidance mode entered
Timeout Interval
Timeout Interval
Queue Saturation Point
Onset of queuing as rate exceeds available capacity
Time

Assisting TCP Performance within the Network—RED and ECN

Although TCP is an end-to-end protocol, it is possible for the network to assist TCP in optimizing performance. One approach is to alter the queue behaviour of the network through the use of *Random Early Detection* (RED). RED permits a network router to discard a packet even when there is additional space in the queue. Although this may sound inefficient, the interaction between this early packet-drop behaviour and TCP is very effective.

RED uses a the weighted average queue length as the probability factor for packet drop. As the average queue length increases, the probability of a packet being dropped, rather than being queued, increases. As the queue length decreases, so does the packet-drop probability. (See Figure 10). Small packet bursts can pass through a RED filter relatively intact, while larger packet bursts will experience increasingly higher packet-discard rates. Sustained load will further increase the packet-discard rates. This implies that the TCP sessions with the largest open windows will have a higher probability of experiencing packet drop, causing a back-off in the window size.

Figure 10: RED Behavior



A major goal of RED is to avoid a situation in which all TCP flows experience congestion at the same time, all then back off and resume at the same rate, and tend to synchronize their behaviour[11,12]. With RED, the larger bursting flows experience a higher probability of packet drop, while flows with smaller burst rates can continue without undue impact. RED is also intended to reduce the incidence of complete loss of ACK signals, leading to timeout and session restart in slow-start mode. The intent is to signal the heaviest bursting TCP sessions the likelihood of pending queue saturation and tail drop before the onset of such a tail-drop congestion condition, allowing the TCP session to undertake a fast retransmit recovery under conditions of congestion avoidance. Another objective of RED is to allow the queue to operate efficiently, with the queue depth ranging across the entire queue size within a timescale of queue depth oscillation the same order as the average RTT of the traffic flows.

Behind RED is the observation that TCP sets very few assumptions about the networks over which it must operate, and that it cannot count on any consistent performance feedback signal being generated by the network. As a minimal approach, TCP uses packet loss as its performance signal, interpreting small-scale packet-loss events as peak load congestion events and extended packet loss events as a sign of more critical congestion load. RED attempts to increase the number of small-scale congestion signals, and in so doing avoid long-period sustained congestion conditions.

It is not necessary for RED to discard the randomly selected packet. The intent of RED is to signal the sender that there is the potential for queue exhaustion, and that the sender should adapt to this condition. An alternative mechanism is for the router experiencing the load to mark packets with an explicit *Congestion Experienced* (CE) bit flag, on the assumption that the sender will see and react to this flag setting in a manner comparable to its response to single packet drop[13] [14]. This mechanism, *Explicit Congestion Notification* (ECN), uses a 2-bit scheme, claiming bits 6 and 7 of the IP Version 4 *Type-of-Service* (ToS) field (or the two *Currently Unused* [CU] bits of the IP *Differentiated Services* field). Bit 6 is set by the sender to indicate that it is an ECN-capable transport system (the ECT bit). Bit 7 is the CE bit, and is set by a router when the average queue length exceeds configured threshold levels.

The ECN algorithm is that an active router will perform RED, as described. After a packet has been selected, the router may mark the CE bit of the packet if the ECT bit is set; otherwise, it will discard the selected packet. (See Figure 11).

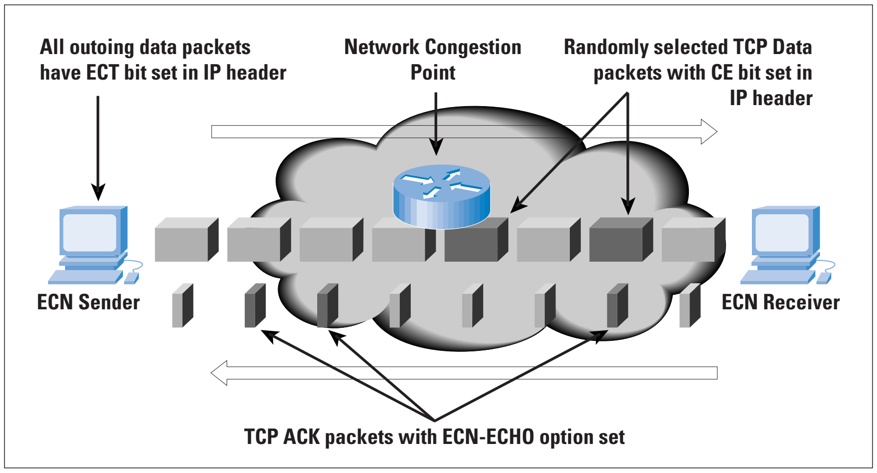

All outoing data packets have ECT bit set in IP header
Network Congestion Point
Randomly selected TCP Data packets with CE bit set in IP header
ECN Sender
ECN Receiver
TCP ACK packets with ECN-ECHO option set

Figure 11: Operation of Explicit Congestion Notification

The TCP interaction is slightly more involved. The initial TCP SYN handshake includes the addition of ECN-echo capability and *Congestion Window Reduced* (CWR) capability flags to allow each system to negotiate with its peer as to whether it will properly handle packets with the CE bit set during the data transfer. The sender sets the ECT bit in all packets sent. If the sender receives a TCP packet with the ECN-echo flag set in the TCP header, the sender will adjust its congestion window as if it had undergone fast recovery from a single lost packet.

The next sent packet will set the TCP CWR flag, to indicate to the receiver that it has reacted to the congestion. The additional caveat is that the sender will react in this way at most once every RTT interval. Further, TCP packets with the ECN-echo flag set will have no further effect on the sender within the same RTT interval. The receiver will set the ECN-echo flag in all packets when it receives a packet with the CE bit set. This will continue until it receives a packet with the CWR bit set, indicating that the sender has reacted to the congestion. The ECT flag is set only in packets that contain a data payload. TCP ACK packets that contain no data payload should be sent with the ECT bit clear.

The connection does not have to await the reception of three duplicate ACKs to detect the congestion condition. Instead, the receiver is notified of the incipient congestion condition through the explicit setting of a notification bit, which is in turn echoed back to the sender in the corresponding ACK. Simulations of ECN using a RED marking function indicate slightly superior throughput in comparison to configuring RED as a packet-discard function.

However, widespread deployment of ECN is not considered likely in the near future, at least in the context of Version 4 of IP. At this stage, there has been no explicit standardization of the field within the IPv4 header to carry this information, and the deployment base of IP is now so wide that any modifications to the semantics of fields in the IPv4 header would need to be very carefully considered to ensure that the changed field interpretation did not exercise some malformed behavior in older versions of the TCP stack or in older router software implementations.

ECN provides some level of performance improvement over a packet-drop RED scheme. With large bulk data transfers, the improvement is moderate, based on the difference between the packet retransmission and congestion-window adjustment of RED and the congestion-window adjustment of ECN. The most notable improvements indicated in ECN simulation experiments occur with short TCP transactions (commonly seen in Web transactions), where a RED packet drop of the initial data packet may cause a six-second retransmit delay. Comparatively, the ECN approach allows the transfer to proceed without this lengthy delay.

The major issue with ECN is the need to change the operation of both the routers and the TCP software stacks to accommodate the operation of ECN. While the ECN proposal is carefully constructed to allow an essentially uncoordinated introduction into the Internet without negative side effects, the effectiveness of ECN in improving overall network throughput will be apparent only after this approach has been widely adopted. As the Internet grows, its inertial mass generates a natural resistance to further technological change; therefore, it may be some years before ECN is widely adopted in both host software and Internet routing systems. RED, on the other hand, has had a more rapid introduction to the Internet, because it requires only a local modification to router behavior, and relies on existing TCP behavior to react to the packet drop.

Tuning TCP

How can the host optimize its TCP stack for optimum performance? Many recommendations can be considered. The following suggestions are a combination of those measures that have been well studied and are known to improve TCP performance, and those that appear to be highly productive areas of further research and investigation[1].

- *Use a good TCP protocol stack:* Many of the performance pathologies that exist in the network today are not necessarily the by-product of oversubscribed networks and consequent congestion. Many of these performance pathologies exist because of poor implementations of TCP flow-control algorithms; inadequate buffers within the receiver; poor (or no) use of path-MTU discovery; no support for fast-retransmit flow recovery, no use of window scaling and SACK, imprecise use of protocol-required timers, and very coarse-grained timers. It is unclear whether network ingress-imposed Quality-of-Service (QoS) structures will adequately compensate for such implementation deficiencies. The conclusion is that attempting to address the symptoms is not the same as curing the disease. A good protocol stack can produce even better results in the right environment.
- *Implement a TCP Selective Acknowledgment (SACK) mechanism:* SACK, combined with a selective repeat-transmission policy, can help overcome the limitation that traditional TCP experiences when a sender can learn only about a single lost packet per RTT.
- *Implement larger buffers with TCP window-scaling options:* The TCP flow algorithm attempts to work at a data rate that is the minimum of the delay-bandwidth product of the end-to-end network path and the available buffer space of the sender. Larger buffers at the sender and the receiver assist the sender in adapting more efficiently to a wider diversity of network paths by permitting a larger volume of traffic to be placed in flight across the end-to-end path.
- *Support TCP ECN negotiation:* ECN enables the host to be explicitly informed of conditions relating to the onset of congestion without having to infer such a condition from the reserve stream of ACK packets from the receiver. The host can react to such a condition promptly and effectively with a data flow-control response without having to invoke packet retransmission.
- *Use a higher initial TCP slow-start rate than the current 1 MSS (Maximum Segment Size) per RTT.* A size that seems feasible is an initial burst of 2 MSS segments. The assumption is that there will be adequate queuing capability to manage this initial packet burst; the provision to back off the send window to 1 MSS segment should remain intact to allow stable operation if the initial choice was too large for the path. A robust initial choice is two segments, although simulations have indicated that four initial segments is also highly effective in many situations.

- *Use a host platform that has sufficient processor and memory capacity to drive the network*. The highest-quality service network and optimally provisioned access circuits cannot compensate for a host system that does not have sufficient capacity to drive the service load. This is a condition that can be observed in large or very popular public Web servers, where the peak application load on the server drives the platform into a state of memory and processor exhaustion, even though the network itself has adequate resources to manage the traffic load.

All these actions have one thing in common: They can be deployed incrementally at the edge of the network and can be deployed individually. This allows end systems to obtain superior performance even in the absence of the network provider tuning the network's service response with various internal QoS mechanisms.

Conclusion

TCP is not a predictive protocol. It is an adaptive protocol that attempts to operate the network at the point of greatest efficiency. Tuning TCP is not a case of making TCP pass more packets into the network. Tuning TCP involves recognizing how TCP senses current network load conditions, working through the inevitable compromise between making TCP highly sensitive to transient network conditions, and making TCP resilient to what can be regarded as noise signals.

If the performance of end-to-end TCP is the perceived problem, the most effective answer is not necessarily to add QoS service differentiation into the network. Often, the greatest performance improvement can be made by upgrading the way that hosts and the network interact through the appropriate configuration of the host TCP stacks.

In the next article on this topic, we will examine how TCP is facing new challenges with increasing use of wireless, short-lived connections, and bandwidth-limited mobile devices, as well as the continuing effort for improved TCP performance. We'll look at a number of proposals to change the standard actions of TCP to meet these various requirements and how they would interact with the existing TCP protocol.

References

[1] Huston, G., *Internet Performance Survival Guide: QoS Strategies for Multiservice Networks,* ISBN 0471-378089, John Wiley & Sons, January 2000.

[2] Postel, J., "Transmission Control Protocol," RFC 793, September 1981.

[3] Jacobson, V., Braden, R., and Borman, D., "TCP Extensions for High Performance," RFC 1323, May 1992.

[4] Mathis, M., Madavi, J., Floyd, S., and Romanow, A., "TCP Selective Acknowledgement Options," RFC 2018, October 1996.

[5] Nagle, J., "Congestion Control in IP/TCP Internetworks," RFC 896, January 1984.

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

[7] Allman, M., Paxson, V., and Stevens, W., "TCP Congestion Control," RFC 2581, April 1999.

[8] Stevens, W. R., *TCP/IP Illustrated, Volume 1,* Addison-Wesley, 1994.

[9] Jacobson V., "Congestion Avoidance and Control," ACM *Computer Communication Review,* Vol. 18, No. 4, August 1988.

[10] Jacobson, V., "Berkeley TCP Evolution from 4.3-Tahoe to 4.3, Reno," Proceedings of the 18th Internet Engineering Task Force, University of British Colombia, Vancouver, BC, September 1990.

[11] Floyd, S., and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM *Transactions on Networking,* Vol. 1, No. 4, August 1993.

[12] Braden, R. et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, April 1998.

[13] Floyd, S., "TCP and Explicit Congestion Notification," ACM *Computer Communication Review,* Vol. 24, No. 5, October 1994.

[14] Ramakrishnan, K., and Floyd, S., "A Proposal to Add Explicit Congestion Notification (ECN) to IP," RFC 2481, January 1999.

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 the past decade, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. Huston is currently the Chief Technologist in the Internet area for Telstra. He is also an active member of the IETF, and is the chair of the Internet Society Board of Trustees. He is author of *The ISP Survival Guide,* ISBN 0-471-31499-4, *Internet Performance Survival Guide: QoS Strategies for Multiservice Networks,* ISBN 0471-378089, and coauthor of *Quality of Service: Delivering QoS on the Internet and in Corporate Networks,* ISBN 0-471-24358-2, a collaboration with Paul Ferguson. All three books are published by John Wiley & Sons. E-mail: `gih@telstra.net`

Overview of Internet Mail Standards

by Paul Hoffman, Internet Mail Consortium

People who are new to the Internet often think it is equivalent to "the Web" since that's what they have heard about most in the media. After a few weeks of using their new Internet account, they tend to say the Internet is "e-mail and the Web," in that order.

Business users have an even higher regard for e-mail. According to the American Management Association, most business people say that e-mail has surpassed the telephone in importance for business communication. While many companies believe that their Web site will be very important in a few years, their e-mail system is already extremely critical to them today.

Because mail is one of the oldest services on the Internet, the protocols used to move mail around are more stable and mature than those used for newer services. The flip side of this is that some of the protocols that are used to move the billions of pieces of mail a day are somewhat arcane and even quaint. The *Internet Engineering Task Force* (IETF) motto "if it isn't broken, don't fix it!" has prevented people from redesigning Internet mail. Instead, numerous extensions and enhancements have been added to the original set of mail standards, as we shall see later.

Historically, there have been many other mail systems, such as BITNET, Fidonet, MAPI, cc:Mail, and so on. Of course, users of these systems still exist, and there is quite an active market for systems that act as gateways between Internet mail and other systems. However, this article only covers the tried-and-true Internet mail system.

The Internet Mail Model

Many Internet protocols are simple client/server systems with a single message payload format. Mostly due to history, Internet mail doesn't have this luxury. Figure 1 shows the main protocols and formats used to move Internet mail.

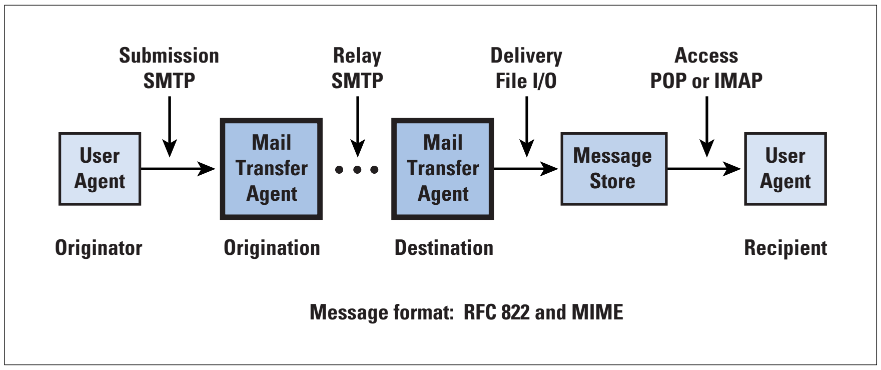


Figure 1: Internet Mail Architecture

A typical mail transaction goes from left to right in the figure. A *Mail User Agent* (MUA), which is most often run by a human but could be controlled by a program, submits a message using the *Simple Mail Transfer Protocol* (SMTP) to the initiating host. That host looks up the IP address associated with the destination host computer and sends the message to the destination host using SMTP. The destination host receives the message and writes it into the local message store (which almost always is a file or database on a hard drive).

The recipient MUA checks the mail store periodically and, if there is mail, retrieves it. Again, the recipient is often a human but might be an automated program such as an order entry system that is controlled by e-mail. The protocols that checks for and retrieves mail are usually the *Post Office Protocol* (POP) or *Internet Message Access Protocol* (IMAP), but it could also be any number of proprietary systems.

E-mail messages have a format that is quite easy to understand, so much so that many other protocols have adopted very similar formats. The message consists of ASCII text *headers* followed by one ASCII (or possibly binary) message *body*. The header format is defined in RFC 822[1], thus the headers are called "RFC 822 headers" or just "822 headers." A simple message body is a single string of text; a complex body uses the *Multipurpose Internet Mail Extensions* (MIME) message format.

Moving Between Hosts: SMTP

Early host-to-host mail delivery was done using file transfer protocols. Since such methods offer little flexibility and requires knowledge of user names and file structures of the remote system, a more general purpose delivery mechanism evolved. The resulting protocol, SMTP was defined in 1982 and has proven to work effectively in the face of orders of magnitude increase in the size of the network.

Even though SMTP moves mail between two host computers, it is a client/server protocol. The host that initiates the contact always acts as a *client,* and the host that was contacted is the *server*. (There are a few rarely-used exceptions to this rule.) The client has a variety of text-based commands that it can give, and the server replies with short responses. The server in the relationship never gives commands on its own, so it is up to the client to ask enough questions, and to carefully watch the server's responses, to know how best to interact with the server.

When a host wants to send mail somewhere on the Internet, it determines where the mail should go and initiates contact with the target server. Thus, the sender is always the SMTP client, and the hosts that are listening for SMTP traffic are always servers. In reality, most SMTP server software can act both as clients and servers; MUAs almost always only participate in SMTP as clients.

The sending host uses the *Domain Name System* (DNS) to determine the IP address of the target host, contacts that host using TCP port 25, and uses SMTP to deliver the message. Sometimes, as we shall see later, this IP lookup involves a level of indirection,—the target host may use a different host to receive mail on its behalf.

SMTP client commands consist of a keyword, possibly followed by command arguments. The server's response always starts with a three-digit number that is a status indicator, which is possibly followed by additional information.

Most SMTP interactions follow a typical set of steps, shown in Figure 2. The initiator who has mail to send (the client) is on the left and the host that is receiving the mail (the server) is on the right. The client first opens a TCP connection on port 25 on the the server. Next, the client and server exchange greetings (the HELO command and response). The client then prepares the server to receive the message by telling the server who the message is from and who it is to; the server gives a positive acknowledgment to each of these commands. The client then asks if the server is ready for the body of the message and, when the server says yes, sends the message as a stream of lines that is followed by a single period on a line by itself. After the server says that it has received the message fully, the client says good-bye and closes the connection.

Figure 2: A Typical SMTP Exchange

Client Action	*Server Response*
<connects to the server using TCP>	
	`220 example.com WhizzyMail server version 2.32`
`HELO somecompany.com`	
	`250-example.com says howdy`
`MAIL FROM:<chrisj@somecompany.com>`	
	`250 OK`
	`354 Start mail input; end with <CRLF>.<CRLF>`
`Text of message...`	
`...more text of message...`	
`.`	
	`250 OK`
`QUIT`	
	`221 example.com Service finished`
<disconnects from the server>	

Submission and Relay

After a message is created, the creator uses SMTP to submit the message to one of two places: a local mail-forwarding host (such as the mail server at the sender's *Internet Service Provider* (ISP) or corporate IS services) or the mail server that the DNS says is definitive for the recipient. The former is typically used by Internet users who do not have persistent network connections; the latter is more common on systems with network connections that are always available.

Messages may be forwarded hop-by-hop from the sending host, via intermediary hosts, to the recipient. This is called "relaying." In many cases, a message will go through more than two relays, for instance when the recipient's network is configured to accept all incoming messages on one machine that later relays messages to individual departmental hosts. Note that submission and relay uses the same SMTP commands described above (a recent change to this scheme is described near the end of this article).

The last host in the chain makes the message available to the recipient. This is done by moving the message to the message store, which usually means "write the message out on disk." There are, of course, many ways to write something on disk; some hosts write out each message as a separate file, some concatenate the message at the end of a file, while others write the message into a database.

Mail Addresses and MX Records

The initiating host's first job is to determine where a message is supposed to go, that is, how to contact the recipient's host. SMTP is a hop-by-hop protocol, meaning that a sending host does not know the true destination host for a message: it only knows the designated recipient host. Of course, this might be the recipient's final host, or it might be a host that will pass the message along further.

The domain name in mail addresses do not necessarily correspond to hosts on the Internet. For example, there is no host whose domain name is **`imc.org`**. When determining where to send a message, the initiating host first looks in the DNS for a *Mail Exchange* (MX) record that matches the domain name in the recipient's mail address. If there is no MX record, the initiating host looks for a DNS A record that matches the domain name. If there is no MX record or A record, the message cannot be delivered.

Many people find MX records to be somewhat tricky. Part of the confusion comes from the fact that an SMTP host is supposed to look up MX records before they look for A records; there are very few protocols that don't rely on A records. Another confusing aspect is that MX records may have wildcards in them. For instance, if a message is being sent to **`someone@eng.example.com`**, there may be no MX record for **`eng.example.com`**, but there may be one for **`*.example.com`**. Wildcard MX records tell the sending host that any message for a domain name that matches the wildcard specification should be sent to the named host.

Modern Mail Extensions

All protocols must evolve, and SMTP has improved over the years. Early mail implementors realized that the initial set of SMTP commands would have to expand. Since the SMTP client gives all commands in an exchange, the client determines which SMTP commands a server will be able to handle. The *SMTP Service Extension*s (ESMTP), defined in RFC 1869[2], is a small change to SMTP that allows an SMTP server to list the commands it knows at the beginning of an SMTP session.

The bootstrapping process for ESMTP is quite simple. Instead of starting with the "HELO" command, an ESMTP server starts with the "EHLO" command. If the SMTP host indicates that it has no idea what "EHLO" means, the client knows that the server doesn't understand ESMTP, and therefore doesn't understand any SMTP extensions. On the other hand, if the server does understand the "EHLO" greeting, the host responds with the entire list of SMTP extensions that the client is allowed to use during the session.

There have been over a dozen extensions to SMTP that are on standards track in the IETF, and many more have been proposed. However, most modern SMTP servers have only implemented a few of these.

Probably the most publicized SMTP extension in the past few years has been the *SMTP Service Extension for Authentication* (AUTH) for authenticating the SMTP client to the server. The AUTH extension, described in RFC 2554[3], allows roaming users to submit mail from outside their local networks without forcing the servers to accept mail from just anyone. This new method, which is now starting to appear in both mail clients and servers, will reduce the hassle faced by many roaming users as they move from ISP to ISP.

Another significant SMTP extension that has become widely implemented is *Delivery Status Notifications,* or DSNs defined in RFC 1891[4]. These are similar to return receipts in postal mail, but with some significant differences. DSNs are issued by SMTP servers, not end users. Thus, the meaning of a DSN is interpreted as “the message was received by this SMTP host,” not “the message was received by the intended recipient.”

Retrieving Mail

After the final SMTP server has received a message and written it into the message store, the recipient needs to be able to access the message. In the early days of Internet mail, the message store was nothing more than a text file on disk, and mail was read by reading the text file. In fact, many people still read their mail this way, albeit using somewhat more modern tools.

If the recipient is not directly logged into the host computer that has the message store, reading the disk file can be difficult. To alleviate this problem, the *Post Office Protocol* (POP), described in RFC 1939[5] introduced a client/server model for an MUA to get mail from the message store and store it on the local computer. The vast majority of mail users today use POP to retrieve their mail.

POP looks like many Internet protocols. The client connects to the server, logs in using a user name and password, checks if it has any messages waiting for it, then asks for the messages one by one. The client has the option of leaving messages that it has read on the server or deleting them after they have been retrieved.

Modern Mail Access with IMAP

Although POP works well for many people, it has its drawbacks. The mail client cannot preview a message to see whether or not it wants to download it. The client has only one mailbox which has no hierarchical structure. In most POP systems, leaving all your mail on the server makes retrieving new mail quite slow. To get around these problems, the mail community developed the *Internet Message Access Protocol* (IMAP), described in RFC 2060[6].

IMAP is significantly more powerful than POP. IMAP clients give the user much more control over their mail, such as letting them keep some of their mail locally while leaving other mail on the server. IMAP even allows for mailboxes that are shared among users, such as group announcements lists. It also gives mail administrators many more opportunities to support novice users by keeping their mail in a central location. Most modern mail clients support IMAP, and IMAP servers are available from many vendors.

It should be noted that, although IMAP is considered much more useful than POP and is widely available, it has had very little adoption in the ISP market (it has been accepted much more readily in the enterprise mail market). The reasons for this are not clear. Many ISPs say they do not want to incur the costs and responsibilities of storing users' mail, even if this gives them greater ability to administer the mail. It is not clear what, if anything, will shift ISPs away from POP to IMAP.

Access Through Web Browsers

The ubiquity of the Web has introduced a new method for getting mail that has become surprisingly popular: the use of the *HyperText Transfer Protocol* (HTTP). Web access to e-mail lets users read their mail without a POP or IMAP client. Of course, this offers many fewer features than POP or IMAP; for instance, you can't easily store messages after reading them and getting file attachments in your mail takes many more steps. However, the big advantage of this method is that Web browsers are almost everywhere these days, and there are many situations where you don't care about being able to store your mail on your local computer.

Giving users Web browser access to their mail quickly became a commodity market. Now, almost every portal offers such services. In fact, many corporations and ISP also offer this service because it is a fairly easy add-on to POP and IMAP servers. As more and more users want to access their e-mail from small devices such as cellular phones, it is likely that these devices will include Web-like mail interfaces.

Client Extensions

Both POP and IMAP are extensible, and developers have proposed many extensions for both protocols, although most work is being done on IMAP. Because of the slow adoption of IMAP by ISPs (who could make its advantages much more visible), it's not clear when these will appear in clients and servers, even though many of them add interesting functionality that is wanted by both users and administrators.

There are many client extensions that don't rely on either POP or IMAP, however. One of the most popular is *Message Disposition Notifications* (MDNs), which are quite similar to postal return receipts. Unlike DSNs, which say that a particular message got to one of the servers in an SMTP chain, MDNs are truly end-to-end, and are returned by recipients when they open their mail.

Some people find MDNs intrusive ("why should he know when I read this?"), and they aren't particularly reliable because not all mail clients (most notably Web browser readers) support them. However, they are a good example of what end users are seeing in terms of extensions that add desired functions to the Internet mail system.

The Format of Mail Messages

SMTP, POP, and (to a great extent) IMAP ignore the contents of a message. SMTP uses its own control information to find the recipient of a message; POP and IMAP retrieve messages based on user account names, which may or may not correspond to the address in a message. In users minds, however, the contents of the messages they read are almost always much more important than the way that the message got to them.

Mail messages consist of two parts: the *headers* and the *body*. The headers come first, followed by a blank line, followed by the body, as shown in Figure 3. The basic structure of messages has remained unchanged since it was defined in RFC 822. Originally, the headers were designed to look like inter-office memos, and also to contain control and debugging information; today, some parts of the headers are considered to be as important as the body of the message.

Figure 3: A Typical E-mail Message

```
Received: from mail.somecompany.com ([198.81.17.2])
  by mail.example.com (8.8.8/8.8.5) with ESMTP id VAA17989
  for <althea@example.com>; Wed, 9 Dec 1998 21:07:44 -0800 (PST)
From: jerry@somecompany.com
Message-ID: <823227a3.366f53ea@somecompany.com>
Date: Wed, 9 Dec 1998 23:54:02 EST
To: Althea Cassidy <althea@example.com>
Mime-Version: 1.0
Subject: I'm outta here
Content-type: text/plain; charset=US-ASCII
Content-transfer-encoding: 7bit

Sorry to make this so brief, but I've got a train to catch.
I'll meet you at the jubilee.
      --J
```

Message Headers

Because they were designed to be functional, message headers have a very straight-forward design. Each header has a single token, followed by a colon, followed by the parameters and options of the header. Headers usually consist of a single line, but you can create multi-line headers by starting the continuation lines with blanks.

There are dozens of common headers, and dozens more that are rarely used. Almost all mail users are familiar with "To:", "From:", "Subject:", and "Date:", and they may have seen additional common headers such as "Cc:" and "Received:". Depending on the interface of the MUA, users typically see some of these headers after they have retrieved a message with POP or IMAP but before they have "opened" the message to see the message body.

Basic and Advanced Message Bodies

Originally, the body of mail messages consisted of plain ASCII text. This was sufficient for the inventors of e-mail, who spoke mostly English and had access to other information transfer mechanisms such as FTP to move binary data around. Of course, such restrictions would not last.

Probably the biggest advance in Internet mail in the past ten years is in the format of mail messages, not in their transport. In the early 1990s, Internet mail went from being text-only to allowing the transfer of non-text messages and parts of messages. MIME, described in RFCs 2045–2047[7, 8, 9], revolutionized the usefulness of Internet mail by allowing senders to include files with messages, to use styled text, to give their messages useful structure, and to provide the first interoperable support for international e-mail.

Unfortunately, the term "attachments" became associated with MIME even though it is much more powerful than just allowing files to be attached to a message. The majority of MIME-enabled messages today don't contain any attachments: instead, they use MIME's capability of labeling the type of a single message body part. MIME labeling can tell the receiving client the format of the message (for instance, an HTML message) and, if it is a text message, the type of characters in the message.

Another great feature of MIME is that it allows messages to have structure. For instance, Figure 4 shows a message with two representations of the same information: text and HTML. A mail client that cannot display HTML can skip that part of the message and just display the plain text. This allows message content to gradually migrate towards new technology. In the near future, it is likely that similar logic will be used for messages that contain XML, HTML, and plain text.

Figure 4: A Multipart MIME message

```
From: jerry@somecompany.com
Message-ID: <828d83ffzwd.47r7c2dxsa@somecompany.com>
Date: Wed, 10 Dec 1998 03:24:00 EST
To: Althea Cassidy <althea@example.com>
Subject: What you should know
MIME-Version: 1.0
Content-Type: multipart/alternative;
   boundary=ad8ekd2ddr9332dc3df332

--ad8ekd2ddr9332dc3df332
Content-Type: text/plain; charset=us-ascii

Important stuff.
Blah blah blah.

--ad8ekd2ddr9332dc3df332
Content-Type: text/html

<html><head><title>Important stuff</title></head><body>
<h1>Important stuff.</h1>
<p><b>Blah blah blah.</b>
</body></html>

--ad8ekd2ddr9332dc3df332--
```

Because of the capability to structure messages, MIME can be used for multimedia and unified messaging. A single mail message can contain one or more movies, sound files, text files in a variety of formats, binary files such as word processing documents, calendar events, fax images, and so on. The MIME structure tells the recipient software which parts of the message contain particular types of data, as well the relationship between the parts (such as "this part contains three different alternative sound formats").

With the explosion of the popularity of the Web, users have come to expect that the content they read will look like Web pages. Most users don't understand that a "Web page" that "contains" graphics in fact isn't a single entity but is really a page of HTML that has links to other pages that contain individual images. They expect to be able to receive mail messages that look just like the things they see on the Web. The MHTML protocol (described in RFC 2557[10]) describes how to structure MIME messages that contain both HTML parts and images so that they appear together in mail clients exactly like they appear in Web browsers.

MIME enables a plethora of other uses for e-mail. For example, secure e-mail using S/MIME and PGP uses MIME to structure the messages so that the cryptographic control information is separate from the message itself. For instance, in a digitally-signed message, the signature information (which is unreadable to the human recipient) is in a different part of the structure than the human-readable content. You can even have layers of encryption and signatures, all structured through MIME.

Internationalization of E-mail

You can use character sets other than ASCII in both the headers and body of Internet e-mail messages. Using different character sets in text bodies requires the use of the "charset" parameter in the "Content-type:" header, as described in RFC 2046[8]. You can also use character sets in message headers with the methods described in RFC 2047[9].

The Future of E-mail

E-mail is incredibly popular with Internet users, but it is far from finished. The next billion new e-mail users will most likely be much less technically savvy than today's Internet users, and they will come to the Internet with very different expectations. In order to give these users a more pleasant experience, the Internet mail industry will have to add many new features and make mail clients easier.

The number of ISPs is also increasing, although not as fast as the number of Internet users. Since e-mail is such an integral part of the service that an ISP offers, mail server software will also have to become easier to administer. Internet mail server vendors are working on such enhancements as a way of gaining a competitive advantage.

The most major change that users will see in the next few years are more highly enabled MUAs. These clients will be all-in-one messaging centers that will handle faxes, voice messages, paging, calendar and event management, and probably some sort of instant messaging. In this way, traditional mail will be only one part of what the user sees when they go to their messaging client.

The importance of Internet fax should not be underestimated. The recent standards for Internet fax, defined in RFCs 2301–2306[11, 12, 13, 14, 15, 16] specify how faxes go through Internet mail. Although there have been a raft of proprietary real-time fax proposals, fax vendors have rallied around faxes in e-mail as an easy way to transition from fax over phone lines. Comparing the high cost of sending international faxes to the near-zero cost of sending e-mail, many companies are quickly moving towards the new standards.

Other mail-enabled services are becoming standardized as well. For example, calendaring over Internet mail is nearing completion. This will allow users to coordinate schedules for meetings, even with people who are not online. E-mail fall-back for phone conversations that were not completed is also being researched.

The e-mail world five and ten years from now will not necessarily look completely different from the way it looks today. Certainly, there will be many more enriched text and multimedia messages being composed by end users. Mailing lists will grow and the mail on them will be more like Web pages than today's text messages. Many people predict that the face of e-mail will change radically if e-mail becomes the "universal inbox" for voicemail, faxes, and other types of communication. Many companies are discovering that regular newsletters sent through e-mail are more effective than expecting users to come to a web site regularly, and it is likely that there will be an increase in the number of publications that are delivered as e-mail.[18]

There is still plenty of room for additions to Internet mail that resemble today's non-Internet services. For instance, users are already clamoring for features such as true message tracking, which is currently available from many package delivery services. Better security is clearly desired, although there seems to be major impediments caused by the need for trusted certificates before we can see wide deployment of secure mail. More problematic features such as message rescinding also have been proposed.

Forces outside the Internet mail world will also change how Internet mail works. For instance, the rapid increase in wireless users will change the way that large messages are handled by message stores. As more users start reading their mail from more than one system, IMAP may become more popular. At the same time, users will expect to be able to move their configuration information with them from machine to machine, probably using protocols such as the *Application Configuration Access Protocol* (ACAP) defined in RFC 2244[17].

There are plenty of opportunities in the Internet mail market. The only significant dark cloud is the possibility that increasing unsolicited e-mail—so called "spam"—might scare away users. To date, the technical solutions for battling spam have been limited, and they probably won't scale well if the amount of spam increases by an order of magnitude. On the bright side, it appears that most legitimate marketers have been scared away from spam and are focusing on opt-in e-mail marketing. This could be a boon for ISPs who specialize in bringing interested e-mail users and potential advertisers together.[19]

In such an environment, mail with rich media and lots of convenience could become the place where many users want to spend much of their time. To get there, we need to build on today's well-established mail protocols and to be creative in the kinds of features we add to both the transport and display of e-mail. Fortunately, we don't need to do much with SMTP, IMAP, and MIME in order to bring these new capabilities to the burgeoning numbers of new users waiting to get on the Internet.

References

[1] Crocker, D., "Standard for the format of ARPA Internet text messages," RFC 822, August 1982.

[2] Klensin, J., Freed, N., Rose, M., Stefferud, E., Crocker, D., "SMTP Service Extensions," RFC 1865, November 1995.

[3] Myers, J., "SMTP Service Extension for Authentication," RFC 2554, March 1999.

[4] Moore, K., "SMTP Service Extension for Delivery Status Notifications," RFC 1891, January 1996.

[5] Myers, J. and Rose, M., "Post Office Protocol—Version 3," RFC 1939, May 1996.

[6] Crispin, M., "Internet Message Access Protocol—Version 4rev1," RFC 2060, December 1996.

[7] Freed, N. and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045, November 1996.

[8] Freed, N. and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part Two: Media," RFC 2046, November 1996.

[9] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text," RFC 2047, November 1996.

[10] Palme, J., Hopmann, A., Shelness, N., "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)," RFC 2557, March 1999.

[11] McIntyre, L., Zilles, S., Buckley, R., Venable, D., Parsons, G., Rafferty, J., "File Format for Internet Fax," RFC 2301, March 1998.

[12] Parsons, G., Rafferty, J., Zilles, S., "Tag Image File Format (TIFF)—image/tiff MIME Sub-type Registration," RFC 2302, March 1998.

[13] Allocchio, C., "Minimal PSTN address format in Internet Mail," RFC 2303, March 1998.

[14] Allocchio, C., "Minimal FAX address format in Internet Mail," RFC 2304, March 1998.

[15] Toyoda, K., Ohno, H., Murai, J., Wing, D., "A Simple Mode of Facsimile Using Internet Mail," RFC 2305, March 1998.

[16] Parsons, G., Rafferty, J., "Tag Image File Format (TIFF)–F Profile for Facsimile," RFC 2306, March 1998.

[17] Newman, C., Myers J. G., "ACAP—Application Configuration Access Protocol," RFC 2244, November 1997.

[18] *Poor Richard's E-mail Publishing,* by Chris Pirillo, ISBN 0966103254, Top Floor Publishing, 1999.

[19] *Internet Messaging: From the Desktop to the Enterprise,* by Marshall T. Rose and David Strom, ISBN 0-13-978610-4, Prentice Hall PTR, 1998.

[20] *Essential Email Standards: RFCs and Protocols Made Practical,* by Pete Loshin, ISBN 0-471-34597-0, Wiley, 1999.

[21] *Internet Email Protocols: A Developer's Guide,* by Kevin Johnson, ISBN 0-201-43288-9, Addison-Wesley, 1999.

PAUL HOFFMAN is the director of the Internet Mail Consortium (**`http://www.imc.org/`**), which is the trade association for Internet mail software vendors and service providers. He is the editor of many recent mail standards, as well as Internet-related books such as *Netscape For Dummies*. He has been active on the Internet for twenty years. E-mail: **`phoffman@imc.org`**

Book Review

Introduction to Data Communications and Networking

Introduction to Data Communications and Networking, Behrouz Farouzan, ISBN 0-256-23044-7, WCB/McGraw-Hill, 1998.

As personal computers have proliferated the landscape over the years, they have become the domain of an increasing number of nontechnical end users. Two things assisted in this transformation. The realization of their value as a productivity tool became apparent, as well as their ability to become more user friendly to the masses. Networks, and networking, have followed a similar path. The investment in creating a networked environment in the past may have been a burden—in both time and added complexity—to all but the largest corporations. However, as the world becomes more "wired," the presence of networks has become commonplace in nearly every work environment, not to mention the movement into private residences. The need to become familiar with concepts and terms as they relate to data communications and networks has become an important part of the technological landscape. *Introduction to Data Communications and Networking* assists the novice in grasping these concepts, as well as serving as a refresher to the more experienced audience.

Organization

The preface explains the ways this book can be useful. The textbook portion is helpful. Multiple choice as well as discussion questions are provided within each chapter, although all the answers are not. In addition, some of the questions asked do not always seem to be posed in the context of the chapter just covered. However, it does turn out to be a rather small inconvenience. The requisite appendices are included as well—such as ASCII and EBCDIC codes, and various representations of numbers. However, two areas that usually receive only fleeting recognition—Fourier analysis and Huffman coding—are covered. Not being an engineer, I'm not sure that I now understand these concepts, but at least now I know why.

Although the areas covered in this book are covered in many introductory network books, this one takes nothing for granted. A good portion of the more experienced readers will know that Layers 2–6 of the OSI model have headers, only Layer 2 will include a trailer. Details such as these are easily forgotten. Introducing concepts in meaningful, practical ways is another positive attribute of this book. One great example is how the author describes the difference between analog and digital. Hands of a traditional, or analog, clock do not jump from minute to minute or hour to hour. The notion of time advancing seems to be a smooth transition, much like an analog signal is a continuous wave form that changes smoothly over time. Digital (as in the case of a digital clock), on the other hand, indicates discrete units of time—usually whole hours and minutes—and can have only limited numbers of defined values. In Chapter 4, analog and digital signals are detailed and explained with clarity and excellent examples are given as well.

In fact, the only subject matter I had difficulty deciphering concerned material presented in Chapter 5. The concepts of polar, unipolar, and bipolar encoding seemed straightforward enough, but digital-to-analog and analog-to-analog encoding will definitely have to be revisited. Amplitude and phase shifting keys may or may not be revisited. In fact, it was at this point that I realized that the material was moving to a different, more difficult, level.

Although the preface states that the first eight chapters are essential for readers being introduced to networking concepts, I found that chapters 5–8 went into a level of depth that would be particularly daunting for an introductory discussion.

Summary

I don't remember exactly how I was introduced to this book—whether I read about it in a journal or it was recommended by a friend—but the book got favorable reviews wherever I inquired about it. It is a practical addition to your bookshelf, regardless of your level of comfort with networks and voice/data communications.

The book is relevant and practical for the professional who has been working in the field for a few years. It is also useful as a textbook for use in the classroom. However, I do not believe that all the information can be adequately covered in a semester, as the author suggests. I believe one of the reasons I enjoyed this book was because of the way it explained ideas and concepts that were never used in any class I had ever taken. I recall promises of receiving a good, comprehensive background in these areas, yet years later I continue to struggle with some of the same concepts I've encountered in classes before. I found myself continually searching for a source that would provide me the information in a comprehensive, understandable fashion. I believe I have finally found it.

—*Steve Barsamian, Cisco Systems*
`sbarsam@cisco.com`

Would You Like to Review a Book for IPJ?

We receive numerous books on computer networking from all the major publishers. If you've got a specific book you are interested in reviewing, please contact us and we will make sure a copy is mailed to you. The book is yours to keep if you send us a review. We accept reviews of new titles, as well as some of the "networking classics." Contact us at **`ipj@cisco.com`** for more information.

Fragments

New Top-Level Domains Are Coming

For several years, there have been proposals to introduce new *generic top-level domains* (gTLDs) into the Internet *Domain Name System* (DNS). Although the introduction of gTLDs raises several issues that are of concern to various members of the Internet community, significant progress has been made recently toward achieving a consensus solution. The *Internet Corporation for Assigned Names and Numbers* (ICANN) Board of Directors is expected to consider adopting a policy to introduce new gTLDs at its meeting in July 2000. The *Names Council* has recommended to the ICANN Board that: "...a limited number of new top-level domains be introduced initially and that the future introduction of additional top-level domains be done only after careful evaluation of the initial introduction."

ICANN Announces CPR Institute as New Dispute Resolution Provider

ICANN recently announced that the *CPR Institute for Dispute Resolution* has been designated an approved provider under their *Uniform Dispute Resolution Policy* (UDRP) for domain name disputes. CPR, an alliance of 500 general counsel of global corporations and partners of major law firms, is the fourth dispute resolution provider to be designated by ICANN to handle domain disputes, joining the *National Arbitration Forum*, the *Disputes.org/eResolution Consortium*, and the *World Intellectual Property Organization*. The UDRP establishes a streamlined, economical process administered by neutral arbitration companies to provide a quick and cheap alternative to litigation. The procedure applies to cases that meet all three of the following criteria: The domain name must be identical or confusingly similar to a name in which the complaining party has trademark rights (either through a registered trademark or a common-law trademark); The domain name holder must have no legitimate right or interest in the name; The domain name must have been registered and used in bad faith.

In its first few months of operation, the UDRP has proven to be a very popular means of quickly resolving trademark/domain name disputes. To date, 691 proceedings have been commenced under the policy involving 1022 domain names. Of those proceedings, 348 have already been resolved. For additional information on UDRP, see **`http://www.icann.org/udrp/udrp.htm`**

The Internet Protocol Journal
Ole J. Jacobsen, Editor and Publisher

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

Cisco Systems

The Internet Protocol Journal, Cisco Systems
170 West Tasman Drive, M/S SJ-10/5
San Jose, CA 95134-1706
USA

ADDRESS SERVICE REQUESTED

Bulk Rate Mail
U.S. Postage
PAID
Cisco Systems, Inc.