Advice for Internet Subnetwork Designers

Abstract

This document provides advice to the designers of digital communication equipment, link-layer protocols and packet-switched subnetworks (collectively referred to as subnetworks) who wish to support the Internet protocols but who may be unfamiliar with Internet architecture and the implications of their design choices on the performance and efficiency of the Internet.

This document represents a consensus of the members of the IETF Performance Implications of Link Characteristics (PILC) working group.

Introduction and Overview

IP, the Internet Protocol [RFC791] is the core protocol of the Internet. IP defines a simple "connectionless" packet-switched network. The success of the Internet is largely attributed to IP’s simplicity, the "end-to-end principle" [SRC81] on which the Internet
is based, and the resulting ease of carrying IP on a wide variety of subnetworks not necessarily designed with IP in mind.

But while many subnetworks carry IP, they do not necessarily do so with maximum efficiency, minimum complexity or minimum cost. Nor do they implement certain features to efficiently support newer Internet features of increasing importance, such as multicasting and quality of service.

With the explosive growth of the Internet, IP is an increasingly large fraction of the traffic carried by the world’s telecommunications networks. It therefore makes sense to optimize both existing and new subnetwork technologies for IP as much as possible.

Optimizing a subnetwork for IP involves three complementary considerations:

1. Providing functionality sufficient to carry IP.
2. Eliminating unnecessary functions that increase cost or complexity.
3. Choosing subnetwork parameters that maximize the performance of the Internet protocols.

Because IP is so simple, consideration 2 is more of an issue than consideration 1. I.e., subnetwork designers make many more errors of commission than errors of omission. But certain enhanced Internet features, such as multicasting and quality-of-service, rely on support from the underlying subnetworks beyond that necessary to carry "traditional" unicast, best-effort IP.

A major consideration in the efficient design of any layered communication network is the appropriate layer(s) in which to implement a given function. This issue was first addressed in the seminal paper "End-to-End Arguments in System Design" [SRC81]. That paper argued that many functions can be implemented properly *only* on an end-to-end basis, i.e., at the highest protocol layers, outside the subnetwork. These functions include ensuring the reliable delivery of data and the use of cryptography to provide confidentiality and message integrity.

These functions cannot be provided solely by the concatenation of hop-by-hop services, so duplicating these functions at the lower protocol layers (i.e., within the subnetwork) can be needlessly redundant or even harmful to cost and performance.

However, partial duplication of functionality in a lower layer can *sometimes* be justified by performance, security or availability considerations. Examples include link-layer retransmission to improve the performance of an unusually lossy channel, e.g., mobile radio; link level encryption intended to thwart traffic analysis; and redundant transmission links to improve availability, increase throughput, or to guarantee performance for certain classes of traffic. Duplication of protocol function should be done only with an understanding of system level implications, including possible interactions with higher-layer mechanisms.

The architecture of the Internet was heavily influenced by the end-to-end principle, and in our view it was crucial to the Internet’s
success.

The remainder of this document discusses the various subnetwork design issues that the authors consider relevant to efficient IP support.

Maximum Transmission Units (MTUs) and IP Fragmentation

IPv4 packets (datagrams) vary in size from 20 bytes (the size of the IPv4 header alone) to a maximum of 65535 bytes. Subnetworks need not support maximum-sized (64KB) IP packets, as IP provides a scheme that breaks packets that are too large for a given subnetwork into fragments that travel as independent packets and are reassembled at the destination. The maximum packet size supported by a subnetwork is known as its Maximum Transmission Unit (MTU).

Subnetworks may, but are not required to indicate the length of each packet they carry. One example is Ethernet with the widely used DIX [DIX] (not IEEE 802.3 [IEEE8023]) header, which lacks a length field to indicate the true data length when the packet is padded to the 60 byte minimum. This is not a problem for uncompressed IP because it carries its own length field.

If optional header compression [RFC1144] [RFC2507] [RFC2508] [RFC3095] is used, however, it is required that the link framing indicate frame length as it is needed for the reconstruction of the original header.

In IP version 4 (the version now in wide use), fragmentation can occur at either the sending host or in an intermediate router, and fragments can be further fragmented at subsequent routers if necessary.

In IP version 6, fragmentation can occur only at the sending host; it cannot occur in a router.

Both IPv4 and IPv6 provide a "path MTU discovery" procedure [RFC1191] [RFC1435] [RFC1981] that allows the sending host to avoid fragmentation by discovering the minimum MTU along a given path and reducing its packet sizes accordingly. This procedure is optional in IPv4 but mandatory in IPv6 where there is no router fragmentation.

Path MTU discovery is widely deployed, but it sometimes encounters problems. Some routers fail to generate the ICMP messages that convey path MTU information to the sender, and sometimes the ICMP messages are blocked by overly restrictive firewalls. The result can be a "Path MTU Black Hole" [RFC2923] [RFC1435].

The Path MTU Discovery procedure, the persistence of path MTU black holes, and the deletion of router fragmentation in IPv6 reflects a consensus of the Internet technical community that IP fragmentation is best avoided. This requires that subnetworks support MTUs that are "reasonably" large. The smallest MTU permitted in IPv4 by [RFC791] is 68 bytes, but such a small value would clearly be inefficient. Because IPv6 omits router fragmentation, [RFC 2460] specifies a larger minimum MTU of 1280 bytes. Any subnetwork with an internal packet payload smaller than 1280 bytes MUST implement an internal fragmentation/reassembly mechanism if it is to support IPv6.

If a subnetwork cannot directly support a "reasonable" MTU with native framing mechanisms, it should internally fragment. That is, it
should transparently break IP packets into internal data elements and reassemble them at the other end of the subnetwork.

This leaves the question of what is a "reasonable" MTU. Ethernet (10 and 100 Mb/s) has a MTU of 1500 bytes, and because of its ubiquity few Internet paths have MTUs larger than this value. This severely limits the utility of larger MTUs provided by other subnetworks. But larger MTUs are increasingly desirable on high speed subnetworks to reduce the per-packet processing overhead in host computers, and implementers are encouraged to provide them even though they may not be usable when Ethernet is also in the path.

Various "tunneling" schemes, such as IP Security [RFC2406] treat IP as a subnetwork for IP. Since tunneling adds header overhead, it can trigger fragmentation even when the same physical subnetworks (e.g., Ethernet) are used on both sides of the IP router. Tunneling has made it more difficult to avoid IP fragmentation and has increased the incidence of path MTU black holes. Larger subnetwork MTUs may help to alleviate this problem.

Choosing the MTU in Slow Networks

In slow networks, the largest possible packet may take a considerable time to send. Interactive response time should not exceed the well-known human factors limit of 100 to 200 ms. This includes all sources of delay: electromagnetic propagation delay, queuing delay, and the store-and-forward time, i.e., the time to transmit a packet at link speed.

At low link speeds, store-and-forward delays can dominate total end-to-end delay, and these are in turn directly influenced by the maximum transmission unit (MTU) size. Even when an interactive packet is given a higher queuing priority, it may have to wait for a large bulk transfer packet to finish transmission. This worst-case wait can be set by an appropriate choice of MTU.

For example, if the MTU is set to 1500 bytes, then a MTU-sized packet will take about 8 milliseconds to send on a T1 (1.536 Mb/s) link. But if the link speed is 19.2kb/s, then the transmission time becomes 625 ms -- well above our 100-200ms limit. A 256-byte MTU would lower this delay to a little over 100 ms. However, care should be taken not to lower the MTU excessively, as this will increase header overhead and trigger frequent IP fragmentation (if Path MTU discovery is not in use). This is likely the case with multicast.

One way to limit delay for interactive traffic without imposing a small MTU is to give priority to this traffic and to preempt (abort) the transmission of a lower-priority packet when a higher priority packet arrives in the queue. However, the link resources used to send the aborted packet are lost, and overall throughput will decrease.

Another way is to implement a link-level multiplexing scheme that allows several packets to be in progress simultaneously, with transmission priority given to segments of higher priority IP packets. For links using the Point-To-Point Protocol (PPP) [RFC1661], multi-class multilink [RFC2686] [RFC2687] [RFC2689] provides such a facility.

ATM (asynchronous transfer mode) is another example of this technique. However, ATM is generally used on high-speed links where
the store-and-forward delays are already minimal, and it introduces significant (~9%) additional overhead due to the addition of 5-byte frame headers to each 48-byte ATM cell.

A third example of link-layer multiplexing is the Data Over Cable Service Interface Specifications. [DOCSIS1] [DOCSIS2] [DOCSIS3] DOCSIS version 1.1 introduces an internal fragmentation and reassembly procedure so that real-time voice packets can avoid undue delay when sharing the relatively slow upstream channel of a cable modem with large data packets.

To summarize, there is a fundamental tradeoff between efficiency and latency in the design of a subnetwork, and the designer should keep this in mind.

Framing on Connection-Oriented Subnetworks

IP requires that subnetworks mark the beginning and end of each variable-length, asynchronous IP packet. Some examples of links and subnetworks that do not provide this as an intrinsic feature include:

1. leased lines carrying a synchronous bit stream;
2. ISDN B-channels carrying a synchronous octet stream;
3. dialup telephone modems carrying an asynchronous octet stream;
and
4. Asynchronous Transfer Mode (ATM) networks carrying an asynchronous stream of fixed-sized "cells".

The Internet community has defined packet framing methods for all these subnetworks. The Point-To-Point Protocol (PPP) [RFC1661] is applicable to bit synchronous, octet synchronous and octet asynchronous links (i.e., examples 1-3 above). ATM has its own framing methods described in [RFC2684] [RFC2364].

At high speeds, a subnetwork should provide a framed interface capable of carrying asynchronous, variable-length IP datagrams. The maximum packet size supported by this interface is discussed above in the MTU/Fragmentation section. The subnetwork may implement this facility in any convenient manner.

IP packet boundaries need not coincide with any framing or synchronization mechanisms internal to the subnetwork. When the subnetwork implements variable sized data units, the most straightforward approach is to place exactly one IP packet into each subnetwork data unit (SNDU), and to rely on the subnetwork's existing ability to delimit SNDUs to also delimit IP packets. A good example is Ethernet. But some subnetworks have SNDUs of one or more fixed sizes, as dictated by switching, forward error correction and/or interleaving considerations. Examples of such subnetworks include ATM, with a single cell size of 48 bytes plus a 5-byte header, and IS-95 digital cellular, with two "rate sets" of four fixed frame sizes each that may be selected on 20 millisecond boundaries.

Because IP packets are of variable length, they may not necessarily fit into an integer multiple of fixed-sized SNDUs. An "adaptation layer" is needed to convert IP packets into SNDUs while marking the boundary between each IP packet in some manner.
There are several approaches to the problem. The first is to encode each IP packet into one or more SNDUs, with no SNDU containing pieces of more than one IP packet, and padding out the last SNDU of the packet as needed. Bits in a control header added to each SNDU indicate where the data segment belongs in the IP packet. If the subnetwork provides in-order, at-most-once delivery, the header can be as simple as a pair of bits to indicate whether the SNDU is the first and/or the last in the IP packet. Or only the last SNDU of the packet could be marked, as this would implicitly mark the next SNDU as the first in a new IP packet. The AAL5 (ATM Adaption Layer 5) scheme used with ATM is an example of this approach, though it adds other features, including a payload length field and a payload CRC.

In AAL5, the 1-bit per segment flag, carried in the ATM header, indicates the end of a packet. (This bit is not protected by the ATM checksum, indicating the need for an end-to-end checksum.) The packet control information (trailer) is located at the end of the segment. Placing the trailer in a fixed position may simplify hardware reassembly.

Another framing technique is to insert per-segment overhead to indicate the presence of a segment option. When present, the option carries a pointer to the end of the packet. This differs from AAL5 in that it permits another packet to follow within the same segment. MPEG-2 [EN301] [ISO13181] supports this style of fragmentation, and may utilize either padding (limiting each transport stream packet to carry only part of one packet), or to allow a second packet to start (no padding).

A third approach is to insert a special flag sequence into the data stream between each IP packet, and to pack the resulting data stream into SNDUs without regard to SNDU boundaries. The flag sequence can also pad unused space at the end of an SNDU. If the special flag appears in the user data, it is escaped to an alternate sequence (usually larger than a flag) to avoid being misinterpreted as a flag. The HDLC-based framing schemes used in PPP are all examples of this approach.

All three adaptation schemes introduce overhead; how much depends on the distribution of IP packet sizes, the size(s) of the SNDUs, and in the HDLC-like approaches, the content of the IP packet (since flags occurring in the packet must be escaped, which expands them). The designer must also weigh implementation complexity in the choice and design of an adaptation layer.

Connection-Oriented Subnetworks

IP has no notion of a "connection"; it is a purely connectionless protocol. When a connection is required by an application, it is usually provided by TCP, the Transmission Control Protocol, running atop IP on an end-to-end basis.

Connection-oriented subnetworks can be (and are) widely used to carry IP, but often with considerable complexity. Subnetworks with a few nodes can simply open a permanent connection between each pair of nodes. This is frequently done with ATM. But the number of connections increases as the square of the number of nodes, so this is clearly impractical for large subnetworks. A "shim" layer between IP and the subnetwork is therefore required to manage connections. This is one of the most common functions of a Subnetwork Dependent
Convergence Function (SNDCF) sublayer between IP and a subnetwork.

SNDCFs typically open subnetwork connections as needed when an IP packet is queued for transmission and close them after an idle timeout. There is no relation between subnetwork connections and any connections that may exist at higher layers (e.g., TCP).

Because Internet traffic is typically bursty and transaction-oriented, it is often difficult to pick an optimal idle timeout. If the timeout is too short, subnetwork connections are opened and closed rapidly, possibly over-stressing its call management system (especially if was designed for voice traffic holding times). If the timeout is too long, subnetwork connections are idle much of the time, wasting any resources dedicated to them by the subnetwork.

The ideal subnetwork for IP is connectionless. Connection-oriented networks that dedicate minimal resources to each connection (e.g., ATM) are a distant second, and connection-oriented networks that dedicate a fixed amount of capacity to each connection (e.g., the PSTN, including ISDN) are the least efficient. If such subnetworks must be used to carry IP, their call-processing systems should be capable of rapid call set-up and tear-down.

Bandwidth on Demand (BoD) Subnets

Wireless networks, including both satellite and terrestrial, may use Bandwidth on Demand (BoD). Bandwidth on demand, which is implemented at the link layer by Demand Assignment Multiple Access (DAMA) in TDMA systems, is currently one proposed mechanism to efficiently share limited spectrum resources amongst a large number of users.

The design parameters for BoD are similar to those in connection oriented subnetworks, however the implementations may be very different. In BoD, the user typically requests access to the shared channel for some duration. Access may be allocated for a period of time at a specific rate, a certain number of packets, or until the user releases the channel. Access may be coordinated through a central management entity or with a distributed algorithm amongst the users. The resource shared may be a terrestrial wireless hop, a satellite uplink, or an end-to-end satellite channel.

Long delay BoD subnets pose problems similar to connection oriented networks in anticipating traffic. While connection oriented subnets hold idle channels open expecting new data to arrive, BoD subnets request channel access based on buffer occupancy (or expected buffer occupancy) on the sending port. Poor performance will likely result if the sender does not anticipate additional traffic arriving at that port during the time it takes to grant a transmission request. It is recommended that the algorithm have the capability to extend a hold on the channel for data that has arrived after the original request was generated (this may done by piggybacking new requests on user data).

There are a wide variety of BoD protocols available and there has been relatively little comprehensive research on the interactions between the BoD mechanisms and Internet protocol performance. A tradeoff exists balancing the time a user can be allowed to hold a channel to drain port buffers with the additional imposed latency on other users who are forced to wait to get access to the channel. It is desirable to design mechanisms that constrain the BoD imposed latency variation. This will be helpful in preventing spurious
Reliability and Error Control

In the Internet architecture, the ultimate responsibility for error recovery is at the end points. The Internet may occasionally drop, corrupt, duplicate or reorder packets, and the transport protocol (e.g., TCP) or application (e.g., if UDP is used as the transport protocol) must recover from these errors on an end-to-end basis. Error recovery in the subnetwork is therefore justified only to the extent that it can enhance overall performance. It is important to recognize that a subnetwork can go too far in attempting to provide error recovery services in the Internet environment. Subnet reliability should be "lightweight", i.e., it only has to be "good enough", *not* perfect.

In this section we discuss how to analyze characteristics of a subnetwork to determine what is "good enough". The discussion below focuses on TCP, which is the most widely used transport protocol in the Internet. It is widely believed (and is a stated goal within the IETF) that non-TCP transport protocols should attempt to be "TCP-friendly" and have many of the same performance characteristics. Thus, the discussion below should be applicable even to portions of the Internet where TCP may not be the predominant protocol.

TCP vs Link-Layer Retransmission

Error recovery involves the generation and transmission of redundant information computed from user data. Depending on how much redundant information is sent and how it is generated, the receiver can use it to reliably detect transmission errors; correct up to some maximum number of transmission errors; or both. The general approach is known as Error Control Coding, or ECC.

For largely historical reasons, the use of ECC to detect transmission errors so that retransmissions (hopefully without errors) can be requested is widely known as "ARQ" (Automatic Repeat Request). ARQ has been used for many decades in computer networking protocols.

When enough ECC information is available to permit the receiver to correct transmission errors without a retransmission, the approach is known as Forward Error Correction (FEC). Due to the greater complexity of the required ECC and the need to tailor its design to the characteristics of a specific modem and channel, FEC has traditionally been implemented in special purpose hardware integral to a modem. This effectively makes it part of the physical layer.

Unlike ARQ, FEC was seldom used for telecommunications outside of deep space links until the 1990s. It is now nearly universal in telephone, cable and DSL modems, digital satellite links and digital mobile telephones. FEC is also heavily used in optical and magnetic storage where "retransmissions" are not possible.

Some systems use hybrid combinations of ARQ layered atop FEC; V.90 dialup modems with V.42bis error control are one example. Most errors are corrected by the trellis (FEC) code within the V.90 modem, and most that remain are detected and corrected by the ARQ mechanisms in V.42bis.

Work is now underway to apply FEC above the physical layer, primarily in connection with reliable multicasting [RFC3048] where conventional
ARQ mechanisms are inefficient or difficult to implement. But in this discussion we will assume that if FEC is present, it is implemented within the physical layer.

Depending on the layer where it is implemented, error control can operate on an end-to-end basis or over a shorter span such as a single link. TCP is the most important example of an end-to-end protocol that uses an ARQ strategy.

Many link-layer protocols use ARQ, usually some flavor of HDLC [ISO3309]. Examples include the X.25 link layer, the AX.25 protocol used in amateur packet radio, 802.11 wireless LANs, and the reliable link layer specified in IEEE 802.2.

As explained in the introduction, only end-to-end error recovery can ensure a reliable service to the application. But some subnetworks (e.g., many wireless links) also require link-layer error recovery as a performance enhancement. For example, many cellular links have small physical frame sizes (< 100 bytes) and relatively high frame loss rates. Relying entirely on end-to-end error recovery clearly yields a performance degradation, as retransmissions across the end-to-end path take much longer to be received than when link-local retransmissions are used. Thus, link-layer error recovery can often increase end-to-end performance. As a result, link-layer and end-to-end recovery often co-exist; this can lead to the possibility of inefficient interactions between the two layers of ARQ protocols.

This inter-layer "competition" might lead to the following wasteful situation. When the link layer retransmits a packet, the link latency momentarily increases. Since TCP bases its retransmission timeout on prior measurements of end-to-end latency, including that of the link in question, this sudden increase in latency may trigger an unnecessary retransmission by TCP of a packet that the link layer is still retransmitting. Such spurious end-to-end retransmissions generate unnecessary load and reduce end-to-end throughput. One may even have multiple copies of the same packet in the same link queue at the same time. In general, one could say the competing error recovery is caused by an inner control loop (link-layer error recovery) reacting to the same signal as an outer control loop (end-to-end error recovery) without any coordination between the loops. Note that this is solely an efficiency issue; TCP continues to provide reliable end-to-end delivery over such links.

This raises the question of how persistent a link-layer sender should be in performing retransmission. We define the link-layer (LL) ARQ persistency as the maximum time that a particular link will spend trying to transfer a packet before it can be discarded. This deliberately simplified definition says nothing about maximum number of retransmissions, retransmission strategies, queue sizes, queuing disciplines, transmission delays, or the like. The reason we use the term LL ARQ persistency instead of a term such as ‘maximum link-layer packet holding time’ is that the definition closely relates to link-layer error recovery. For example, on links that implement straightforward error recovery strategies, LL ARQ persistency will often correspond to a maximum number of retransmissions permitted per link-layer frame [ARQ-DRAFT].

For link layers that do not or cannot differentiate between flows (e.g., due to network layer encryption), the LL ARQ persistency should be small. This avoids any harmful effects or performance degradation resulting from indiscriminate high persistence. A
A detailed discussion of these issues is provided in [ARQ-DRAFT].

However, when a link layer can identify individual flows and apply ARQ selectively [ARQ-DRAFT], then the link ARQ persistency should be high for a flow using reliable unicast transport protocols (e.g., TCP) and must be low for all other flows. Setting the link ARQ persistency larger than the largest link outage allows TCP to rapidly restore transmission without the need to wait for a retransmission time out. This generally improves TCP performance in the face of transient outages. However, excessively high persistence may be disadvantageous; a practical upper limit of 30–60 seconds may be desirable. Implementation of such schemes remains a research issue. (See also Section "Recovery from Subnetwork Outages").

Many subnetwork designers have opportunities to reduce the probability of packet loss, e.g., with FEC, ARQ and interleaving, at the cost of increased delay. TCP performance improves with decreasing loss but worsens with increasing end-to-end delay, so it is important to find the proper balance through analysis and simulation.

Recovery from Subnetwork Outages

Some types of subnetworks, particularly mobile radio, are subject to frequent temporary outages. For example, an active cellular data user may drive or walk into an area (such as a tunnel) that is out of range of any base station. No packets will be successfully delivered until the user returns to an area with coverage.

The Internet protocols currently provide no standard way for a subnetwork to explicitly notify an upper layer protocol (e.g., TCP) that it is experiencing an outage rather than severe congestion. Under these circumstances TCP will, after each unsuccessful retransmission, wait even longer before trying again; this is its "exponential back-off" algorithm. And TCP will not discover that the subnetwork outage has ended until its next retransmission attempt. If TCP has backed off, this may take some time. This can lead to extremely poor TCP performance over such subnetworks.

It is therefore highly desirable that a subnetwork subject to outages not silently discard packets during an outage. Ideally, it should define an interface to the next higher layer (i.e., IP) that allows it to refuse packets during an outage, and to automatically ask IP for new packets when it is again able to deliver them. If it cannot do this, then the subnetwork should hold onto at least some of the packets it accepts during an outage and attempt to deliver them when the subnetwork comes back up. When packets are discarded, IP should be notified so that the appropriate ICMP messages can be sent.

Note that it is *not* necessary to completely avoid dropping packets during an outage. The purpose of holding onto a packet during an outage, either in the subnetwork or at the IP layer, is so that its eventual delivery will implicitly notify TCP that the subnetwork is again operational. This is to enhance performance, not to ensure reliability -- a task that as discussed earlier can only be done properly on an end-to-end basis.

Only a single packet per TCP connection, including ACKs, need be held in this way to cause the TCP sender to recover from the additional losses once the flow resumes. [ARQ-DRAFT]
Because it would be a layering violation (and possibly a performance hit) for IP or a subnetwork layer to look at TCP headers (which would in any event be impossible if IPSEC [RFC2401] encryption is in use), it would be reasonable for the IP or subnetwork layers to choose, as a design parameter, some small number of packets that it will retain during an outage.

CRCs, Checksums and Error Detection

The TCP, UDP and IPv4 protocols all use the same simple 16-bit 1’s complement checksum algorithm to detect corrupted packets. The IP checksum protects only the IP header, while the TCP and UDP checksums protect both the TCP/UDP header and any user data.

These checksums are not very strong from a coding theory standpoint. But they are easy to compute in software, and various proposals to replace them with stronger checksums have failed. Yet a study [SP2000] has shown that the Internet corrupts one in 1,100 to 32,000 packets, and it is up to the end-to-end Internet checksum to detect these errors.

Most packet corruption appears to be caused by bugs and errors in host and router hardware and software. So even if every subnetwork implemented strong error detection, the end-to-end use of TCP and UDP checksums would still be necessary.

Most subnetworks implement error detection just above the physical layer. Packets corrupted in transmission are detected and discarded before delivery to the IP layer. A 16-bit cyclic redundancy check (CRC) is usually the minimum, and this is known to be considerably stronger against most kinds of errors than the 16-bit standard Internet checksum. The Point-to-Point Protocol [RFC1662] requires support of a 16-bit CRC, with a 32-bit CRC as an option. (Note that PPP is often used in conjunction with a dialup modem, which provides its own error control). Other subnetworks, including 802.3/Ethernet, AAL5/ATM, FDDI, Token Ring and PPP over SONET/SDH all use a 32-bit CRC that is considerably stronger. In addition, many subnetworks (notably dialup modems, mobile radio and satellite channels) also incorporate forward error correction, often in hardware.

Any new subnetwork designed to carry IP should therefore provide error detection at least as strong as the 32-bit CRC specified in [ISO3309]. While this will achieve a very low undetected packet error rate, it will not (and need not) achieve a very low packet loss rate as the Internet protocols are better suited to dealing with lost packets than with corrupted packets.

For link layers that can differentiate between flows, it may be appropriate to reduce the error detection level for certain flows with large numbers of small packets, such as voice flows. As such flows also benefit significantly from header compression, this should only be combined with a header compression scheme that is robust against residual bit errors [RFC3095].

Designers of complex subnetworks consisting of internal links and packet switches should consider implementing error detection on an edge-to-edge basis, i.e., at the interface to IP, either in addition to or instead of error detection at the interface to each physical link. This has the significant advantage of protecting against errors introduced anywhere in the subnetwork, not just its transmission links.
This is straightforward if the interface presented to IP by the subnetwork already includes error detection, as with PPP or Ethernet. If the subnetwork carries the PPP or Ethernet CRC without change through the subnetwork, it will automatically provide the desired edge-to-edge error detection. An existing example of such a subnetwork is an Ethernet bridge, also known as a switched hub.

IP version 6 (IPv6) has no IP header checksum. The destination host detects "important" errors in the IP header such as the delivery of the packet to the wrong destination. This is done by including the IP source and destination addresses in the computation of the checksum in the TCP or UDP header, a practice already performed in IPv4. Errors in other IPv6 header fields may go undetected; this was considered a reasonable price to pay for a considerable reduction in the processing required by each router. If desired, additional protection may be obtained for the IPv6 header by the use of the authentication and packet integrity services of the IP Security (IPSEC) protocol.

How TCP Works

One of TCP's functions is end-host based congestion control for the Internet. This is a critical part of the overall stability of the Internet, so it is important that link-layer designers understand TCP's congestion control algorithms.

TCP assumes that, at the most abstract level, the network consists of links and queues. Queues provide output-buffering on links that are momentarily oversubscribed. They smooth instantaneous traffic bursts to fit the link bandwidth.

When demand exceeds link capacity long enough to fill the queue, packets must be dropped. The traditional action of dropping the most recent packet ("tail dropping") is no longer recommended [RED93], but it is still widely practiced.

TCP uses sequence numbering and acknowledgments (ACKs) on an end-to-end basis to provide reliable, sequenced, once-only delivery. TCP ACKs are cumulative, i.e., each implicitly ACKs every segment received so far. If a packet is lost, the cumulative ACK will cease to advance.

Since the most common cause of packet loss is congestion, TCP treats packet loss as an Internet congestion indicator. This happens automatically, and the subnetwork need not know anything about IP or TCP. It simply drops packets whenever it must, though some packet-dropping strategies (e.g., RED) are more fair to competing flows than others.

TCP recovers from packet losses in two different ways. The most important is the retransmission timeout. If an ACK fails to arrive after a certain period of time, TCP retransmits the oldest unacked packet. Taking this as a hint that the network is congested, TCP waits for the retransmission to be ACKed before it continues, and it gradually increases the number of packets in flight as long as a timeout does not occur again.

A retransmission timeout can impose a significant performance penalty, as the sender is idle during the timeout interval and restarts with a congestion window of 1 following the timeout. To
allow faster recovery from the occasional lost packet in a bulk transfer, an alternate scheme known as "fast recovery" was introduced [RFC2581] [RFC2582] [RFC2914] [TCPF98].

Fast recovery relies on the fact that when a single packet is lost in a bulk transfer, the receiver continues to return ACKs to subsequent data packets that do not actually acknowledge any newly-received data. These are known as "duplicate acknowledgments" or "dupacks". The sending TCP can use dupacks as a hint that a packet has been lost and retransmit it without waiting for a timeout. Dupacks effectively constitute a negative acknowledgment (NAK) for the packet sequence number in the acknowledgment field. TCP waits until a certain number of dupacks (currently 3) are seen prior to assuming a loss has occurred; this helps avoid an unnecessary retransmission during out-of-sequence delivery. Recent proposals have been made to lower the dupack threshold to 2.

A new technique called "Explicit Congestion Notification" (ECN) allows routers to directly signal congestion to hosts without dropping packets. This is done by setting a bit in the IP header. Since this is currently an optional behavior (and, longer term, there will always be the possibility of congestion in portions of the network which don’t support ECN), the lack of an ECN bit MUST NEVER be interpreted as a lack of congestion. Thus, for the foreseeable future, TCP MUST interpret a lost packet as a signal of congestion.

The TCP "congestion avoidance" [RFC2581] algorithm maintains a congestion window (cwnd) controlling the amount of data TCP may have in flight at any moment. Reducing cwnd reduces the overall bandwidth obtained by the connection; similarly, raising cwnd increases the performance, up to the limit of the available bandwidth.

TCP probes for available network bandwidth by setting cwnd at one packet and then increasing it by one packet for each ACK returned from the receiver. This is TCP’s "slow start" mechanism. When a packet loss is detected (or congestion is signaled by other mechanisms), cwnd is reset to one and the slow start process is repeated until cwnd reaches one half of its previous setting before the reset. Cwnd continues to increase past this point, but at a much slower rate than before. If no further losses occur, cwnd will ultimately reach the window size advertised by the receiver.

This is an "Additive Increase, Multiplicative Decrease" (AIMD) algorithm. The steep decrease of cwnd in response to congestion provides for network stability; the AIMD algorithm also provides for fairness between long running TCP connections sharing the same path.

TCP Performance Characteristics

Caveat

Here we present the current "state-of-the-art" understanding of TCP performance. This analysis attempts to characterize the performance of TCP connections over links of varying characteristics.

Link designers may wish to use the techniques in this section to predict what performance TCP/IP may achieve over a new link-layer design. Such analysis is encouraged. Because this is relatively new analysis, and the theory is based on single stream TCP connections under "ideal" conditions, it should be recognized that the results of such analysis may be different than actual performance in the
Internet. That being said, we have done the best we can to provide information which will help designers get an accurate picture of the capabilities and limitations of TCP under various conditions.

The Formulae

The performance of TCP’s AIMD Congestion Avoidance algorithm has been extensively analyzed. The current best formula for the performance of the specific algorithms used by Reno TCP is given by Padhye, et al [PFTK98]. This formula is:

\[
\text{BW} = \frac{\text{MSS}}{\text{RTT}\sqrt{1.33p} + \text{RTO}p[1+32p^2]\min[1,3\sqrt{.75p}]}
\]

where

- \( \text{BW} \) is the maximum throughput achievable
- \( \text{MSS} \) is the segment size being used by the connection
- \( \text{RTT} \) is the end-to-end round trip time of the TCP connection
- \( \text{RTO} \) is the packet timeout (based on RTT)
- \( p \) is the packet loss rate for the path
  (i.e. .01 if there is 1% packet loss)

Note that the speed of the links making up the Internet path does not explicitly appear in this formula. Attempting to send faster than the slowest link in the path causes the queue to grow at the transmitter driving the bottleneck. This increases the RTT, which in turn reduces the achievable throughput.

This is currently considered to be the best approximate formula for Reno TCP performance. A further simplification to this formula is generally made by assuming that RTO is approximately 5*RTT.

TCP is constantly being improved. A simpler formula, which gives an upper bound on the performance of any AIMD algorithm which is likely to be implemented in TCP in the future, was derived by Ott, et al [MSMO97][OKM96].

\[
\text{BW} = \frac{\text{MSS}}{\text{RTT}\sqrt{p}}
\]

where \( C = 0.93 \).

Assumptions

Both formulae assume that the TCP Receiver Window is not limiting the performance of the connection. Because receiver window is entirely determined by end-hosts, we assume that hosts will maximize the announced receiver window to maximize their network performance.

Both of these formulae allow BW to become infinite if there is no loss. This is because an Internet path will drop packets at bottleneck queues if the load is too high. Thus, a completely lossless TCP/IP network can never occur (unless the network is being underutilized).

The RTT used is the arithmetic average, including queuing delays.

The formulae are for a single TCP connection. If a path carries many
TCP connections, each will follow the formulae above independently.

The formulae assume long running TCP connections. For connections which are extremely short (<10 packets) and don’t lose any packets, performance is driven by the TCP slow start algorithm. For connections of medium length, where on average only a few segments are lost, single connection performance will actually be slightly better than given by the formulae above.

The difference between the simple and complex formulae above is that the complex formula includes the effects of TCP retransmission timeouts. For very low levels of packet loss (significantly less than 1%), timeouts are unlikely to occur, and the formulae lead to very similar results. At higher packet losses (1% and above), the complex formula gives a more accurate estimate of performance (which will always be significantly lower than the result from the simple formula).

Note that these formulae break down as p approaches 100%.

Analysis of Link-Layer Effects on TCP Performance

Consider the following example:

A designer invents a new wireless link layer which, on average, loses 1% of IP packets. The link layer supports packets of up to 1040 bytes, and has a one-way delay of 20 msec.

If this link layer were used in the Internet, on a path that otherwise had a round trip of of 80 msec, you could compute an upper bound on the performance as follows:

For MSS, use 1000 bytes to exclude the 40 bytes of TCP/IP headers.

For RTT, use 120 msec (80 msec for the Internet part, plus 20 msec each way for the new wireless link).

For p, use .01. For C, assume 1.

The simple formula gives:

\[ BW = \frac{1000 \times 8 \text{ bits}}{0.120 \text{ sec} \times \sqrt{0.01}} = 666 \text{ kbit/sec} \]

The more complex formula gives:

\[ BW = 402.9 \text{ kbit/sec} \]

If this were a 2 Mb/s wireless LAN, the designers might be somewhat disappointed.

Some observations on performance:

1. We have assumed that the packet losses on the link layer are interpreted as congestion by TCP. This is a "fact of life" that must be accepted.

2. The equations for TCP performance are all expressed in terms of packet loss, but many link-layer designers think in terms of bit-error rate. *If* channel bit errors are independent, then the probability of a packet being corrupted is:
\[ p = 1 - (1 - \text{BER})^{[\text{PACKET} \_\text{SIZE} \times 8]} \]

Here we assume \text{PACKET} \_\text{SIZE} is in bytes. It includes the user data and all headers (TCP, IP and subnetwork). If the inequality

\[ \text{BER} \times [\text{PACKET} \_\text{SIZE} \times 8] \ll 1 \]

holds, the packet loss probability \( p \) can be approximated by:

\[ p = \text{BER} \times [\text{PACKET} \_\text{SIZE} \times 8] \]

These equations can be used to apply BER to the performance equations above.

Note that \text{PACKET} \_\text{SIZE} can vary from one packet to the next. Small packets (such as TCP acks) generally have a smaller probability of packet error than, say, a TCP packet carrying one MSS (maximum segment size) of user data. A flow of small TCP acks can be expected to be slightly more reliable than a stream of larger TCP data segments.

It bears repeating that the above analysis assumes that bit errors are statistically independent. Because this is not true for many real links, our computation of \( p \) is actually an upper bound, not the exact probability of packet loss.

There are many reasons why bit errors are not independent on real links. Many radio links are affected by propagation fading or by interference that lasts over many bit times.

Also, links with Forward Error Correction (FEC) generally have very non-uniform bit error distributions that depend on the type of FEC, but in general the uncorrected errors tend to occur in bursts even when channel symbol errors are independent. In all such cases our computation of \( p \) from BER can only place an upper limit on the packet loss rate.

If the distribution of errors under the FEC scheme is known, one could apply the same type of analysis as above, using the correct distribution function for the BER. It is more likely in these FEC cases, however, that empirical methods will need to be used to determine the actual packet loss rate.

3. Note that the packet size plays an important role. If the subnetwork loss characteristics are such that large packets have the same probability of loss as smaller packets, then larger packets will yield improved performance.

4. We have chosen a specific RTT that might occur on a wide-area Internet path within the USA. It is important to recognize that RTTs vary considerably in the Internet.

For example, RTTs are typically less than 10 msec in a wired LAN environment. International connections may have RTTs of 200 msec or more. Modems and other low-capacity links can add considerable delay due to their long packet transmission times.

Links over geostationary repeater satellites have one-way speed-of-light delays of around 250ms: 125ms up to the satellite and 125ms down. The RTT of an end-to-end TCP connection that includes such a link can be expected to be greater than 250ms.
Queues on heavily-congested links may back up, increasing RTTs. Finally, virtual private networks (VPNs) and other forms of encryption and tunneling can add significant end-to-end delay to network connections.

Quality-of-Service (QoS) considerations

It is generally recognized that specific service guarantees are needed to support real-time multimedia, toll quality telephony and other performance critical applications. The provision of such Quality of Service guarantees in the Internet is an active area of research and standardization. The IETF has not converged on a single service model, set of services or single mechanism that will offer useful guarantees to applications and be scalable to the Internet. Indeed, the IETF does not have a single definition of Quality of Service. [RFC2990] represents the present understanding of the challenges in architecting QoS for the Internet.

There are presently two architectural approaches to providing mechanisms for QoS support in the Internet.

IP Integrated Services (Intserv) [RFC1633] provides fine-grained service guarantees to individual flows. Flows are identified by a flow specification (flowspec), which creates a stateful association between individual packets by matching fields in the packet header. Bandwidth is reserved for the flow, and appropriate traffic conditioning and scheduling is installed in routers along the path. The ReSerVation Protocol (RSVP) [RFC2205, RFC2210] usually, but not necessarily, is used to install the flow QoS state. Intserv defines two services, in addition to the Default (best effort) service.

-- Guaranteed Service (GS) [RFC 2212] offers hard upper bounds on delay to flows that conform to a traffic specification (TSpec). It uses a fluid flow model to relate the TSpec and reserved bandwidth (RSpec) to variable delay. Non-conforming packets are forwarded on a best-effort basis.

-- Controlled Load Service (CLS) [RFC2211] offers delay and packet loss equivalent to that of an unloaded network to flows that conform to a TSpec, but no hard bounds. Non-conforming packets are forwarded on a best-effort basis.

Intserv requires installation of state information in every participating router. Performance guarantees cannot be made unless this state is present in every router along the path. This, along with RSVP processing and the need for usage-based accounting, is believed to have scalability problems, particularly in the core of the Internet [RFC2208].

IP Differentiated Services (Diffserv) [RFC2475] provides a "toolkit" offering coarse-grained controls to aggregates of flows. Diffserv in itself does NOT provide QoS guarantees, but can be used to construct services with QoS guarantees across a Diffserv domain. Diffserv attempts to address the scaling issues associated with Intserv by requiring state awareness only at the edge of a Diffserv domain. At the edge, packets are classified into flows, and the flows are conditioned (marked, policed or shaped) to a traffic conditioning specification (TCS). A Diffserv Codepoint (DSCP), identifying a per-hop behavior (PHB), is set in each packet header. The DSCP is carried in the DS-field, subsuming six bits of the former TOS byte of
the IP header [RFC2474]. The PHB denotes the forwarding behavior to be applied to the packet in each node in the Diffserv domain.

Although there is a "recommended" DSCP associated with each PHB, the mappings from DSCPs to PHBs are defined by the DS-domain. In fact, there can be several DSCPs associated with the same PHB. Diffserv presently defines three PHBs.

The class selector PHB [RFC2474] replaces the IP precedence field of the former TOS byte. It offers relative forwarding priorities.

The Expedited Forwarding (EF) PHB [RFC2598] guarantees that packets will have a well-defined minimum departure rate which, if not exceeded, ensures that the associated queues are short or empty. EF is intended to support services that offer tightly-bounded loss, delay and delay jitter.

The Assured Forwarding (AF) PHB group [RFC2597] offers different levels of forwarding assurance for each aggregated flow of packets. Each AF group is independently allocated forwarding resources. Packets are marked with one of three drop precedences; those with the highest drop precedence are dropped with lower probability than those marked with the lowest drop precedence. DSCPs are recommended for four independent AF groups, although a DS domain can have more or fewer AF groups.

Ongoing work in the IETF is addressing ways to support Intserv with Diffserv. There is some belief (e.g. as expressed in [RFC 2990]) that such an approach will allow individual flows to receive service guarantees and scale to the global Internet.

The QoS guarantees that can be offered by the IP layer are a product of two factors:

-- the concatenation of the QoS guarantees offered by the subnets along the path of a flow. This implies that a subnet may wish to offer multiple services (with different QoS guarantees) to the IP layer, which can then determine which flows use which subnet service. Or, to put it another way, forwarding behavior in the subnet needs to be 'clued' by the forwarding behavior (service or PHB) at the IP layer, and

-- the operation of a set of cooperating mechanisms, such as bandwidth reservation and admission control, policy management, traffic classification, traffic conditioning (marking, policing and/or shaping), selective discard, queuing and scheduling. Note that support for QoS in subnets may require similar mechanisms, especially when these subnets are general topology subnets (e.g., ATM, frame relay or MPLS) or shared media subnets.

Many subnetwork designers face inherent tradeoffs between delay, throughput, reliability and cost. Other subnetworks have parameters that manage bandwidth, internal connection state, and the like. Therefore, the following subnetwork capabilities may be desirable, although some might be trivial or moot if the subnet is a simple point-to-point link.

- The subnetwork should have the ability to reserve bandwidth for a connection or flow and schedule packets accordingly.

- Bandwidth reservations should be based on a one- or two-token bucket model, depending on whether the service is intended to
support constant-rate or bursty traffic.

- If a connection or flow does not use its reserved bandwidth at a given time, the unused bandwidth should be available for other flows.

- Packets in excess of a connection or flow’s agreed rate should be forwarded as best effort or discarded, depending on the service offered by the subnet to the IP layer.

- If a subnet contains error control mechanisms (retransmission and/or FEC), it should be possible for the IP layer to influence the inherent tradeoffs between uncorrected errors, packet losses and delay. These capabilities at the subnet/IP layer service boundary correspond to selection of particular error control mechanisms within the subnetwork.

- The subnet layer should know, and be able to inform the IP layer, how much fixed delay and delay jitter it offers for a flow or connection. If the Intserv model is used, the delay jitter component may best be expressed in terms of the TSpec/RSpec model described in [RFC2212].

- Support of the Diffserv class selectors [RFC2474] suggests that the subnet might consider mechanisms that support priorities.

Fairness vs Performance

Subnetwork designers should be aware of the tradeoffs between fairness and efficiency inherent in many transmission scheduling algorithms. For example, many local area networks use contention protocols to resolve access to a shared transmission channel. These protocols represent overhead. Limiting the amount of data that a station may transmit per contention cycle helps assure each station of timely access to the channel, but it also increases contention overhead per unit of data sent.

In some mobile radio networks, capacity is limited by interference, which in turn depends on average transmitter power. Some receivers may require considerably more transmitter power (generating more interference and consuming more channel capacity) than others.

In each case, the scheduling algorithm designer must balance competing objectives: providing a fair share of capacity to each station while maximizing the total capacity of the network.

Delay Characteristics

TCP bases its retransmission timeout (RTO) on measurements of the round trip delay experienced by previous packets. This allows TCP to adapt automatically to the very wide range of delays found on the Internet. The recommended algorithms are described in [RFC2988].

These algorithms model the delay along an Internet path as a normally-distributed random variable with slowly-varying mean and standard deviation. TCP estimates these two parameters by exponentially smoothing individual delay measurements, and it sets the RTO to the estimated mean delay plus some fixed number of standard deviations. (The algorithm actually uses mean deviation as
an approximation to standard deviation, as it is easier to compute.)

The goal is to compute a RTO that is small enough to detect and recover from packet losses while minimizing unnecessary ("spurious") retransmissions when packets are unexpectedly delayed but not lost. Although these goals conflict, the algorithm works well when the delay variance along the Internet path is low, or the packet loss rate is low.

If the path delay variance is high, TCP sets a RTO that is much larger than the mean of the measured delays. But if the packet loss rate is low, the large RTO is of little consequence, as timeouts occur only rarely. Conversely, if the path delay variance is low, then TCP recovers quickly from lost packets; again, the algorithm works well.

But when delay variance and the packet loss rate are both high, these algorithms perform poorly, especially when the mean delay is also high.

Because TCP uses returning acknowledgments as a "clock" to time the transmission of additional data, excessively high delays (even if the delay variance is low) also affects TCP’s ability to fully utilize a high speed transmission pipe. It also slows down the recovery of lost packets even when delay variance is small.

Subnetwork designers should therefore minimize all three parameters (delay, delay variance and packet loss) as much as possible.

In many subnetworks, these parameters are inherently in conflict. For example, on a mobile radio channel the subnetwork designer can use retransmission (ARQ) and/or forward error correction (FEC) to trade off delay, delay variance and packet loss in an effort to improve TCP performance. For example, while ARQ increases delay variance, FEC does not. However, FEC (especially when combined with interleaving) often increases mean delay even on good channels where ARQ would not increase either the delay or the delay variance.

The tradeoffs among these error control mechanisms and their interactions with TCP can be quite complex, and they are the subject of much ongoing research. We therefore recommend that subnetwork designers provide as much flexibility as possible in the implementation of these mechanisms, and to provide access to them as discussed above in the section on Quality of Service.

Bandwidth Asymmetries

Some subnetworks may provide asymmetric bandwidth and the Internet protocol suite will generally still work fine. However, there is a case when such a scenario reduces TCP performance. Since TCP data segments are ‘clocked’ out by returning acknowledgments TCP senders are limited by the rate at which ACKs can be returned [BPK98]. Therefore, when the ratio of the bandwidth of the subnetwork carrying the data to the bandwidth of the subnetwork carrying the acknowledgments is too large, the slow return of of the ACKs directly impacts performance. Since ACKs are generally smaller than data segments, TCP can tolerate some asymmetry, but as a general rule designers of subnetworks should avoid large differences in the incoming and outgoing bandwidth.

One way to cope with asymmetric subnetworks is to increase the size
of the data segments as much as possible. This allows more data to be sent per ACK, mitigating the slow flow of ACKs. Using the delayed acknowledgment mechanism [Bra89] that reduces the number of ACKs transmitted by the receiver by roughly half can also improve performance by reducing the congestion on the ACK channel. These mechanisms should be employed in asymmetric networks.

Several researchers have introduced strategies for coping with bandwidth asymmetry. These mechanisms generally attempt to reduce the number of ACKs being transmitted over the low bandwidth channel by limiting the ACK frequency or filtering out ACKs at an intermediate router [BPK98]. While these solutions mitigate the performance problems caused by asymmetric subnetworks, they do have some cost. Therefore, as suggested above, bandwidth asymmetry should be minimized in subnetwork designs.

Buffering, flow & congestion control

Many subnets include multiple links with varying traffic demands and possibly different transmission speeds. At each link there must be a queuing system, including buffering, scheduling and a capability to discard excess subnet packets. These queues may also be part of a subnet flow control or congestion control scheme.

For the purpose of this discussion, we talk about packets without regard to whether they refer to a complete IP datagram or a subnetwork packet. At each queue, a packet experiences a delay that depends on competing traffic and the scheduling discipline, and is subjected to a local discarding policy.

Some subnets may have flow or congestion control mechanisms in addition to packet dropping. Such mechanisms can operate on components in the subnet layer, such as schedulers, shapers or discarders, and can affect the operation of IP forwarders at the edges of the subnet. However, with the exception of RFC2481 explicit congestion notification (discussed below), IP has no way to pass explicit congestion or flow control signals to TCP.

TCP traffic, especially aggregated TCP traffic, is bursty. As a result, instantaneous queue depths can vary dramatically, even in nominally stable networks. For optimal performance, packets should be dropped in a controlled fashion, not just when buffer space is unavailable. How much buffer space should be supplied is still a matter of debate, but as a rule of thumb, each node should have enough buffering to hold one bandwidth*delay product’s worth of data for each TCP connection sharing the link.

This is often difficult to estimate, since it depends on parameters beyond the subnetwork’s control or knowledge. Internet nodes generally do not implement admission control policies, and cannot limit the number of TCP connections that use them. In general, it is wise to err in favor of too much buffering rather than too little. It may also be useful for subnets to incorporate mechanisms that measure propagation delays to assist in buffer sizing calculations.

There is a rough consensus in the research community that active queue management is important to improving fairness, link utilization and throughput [RFC2309]. Although there are questions and concerns about the effectiveness of active queue management (e.g., see [MBDL99]), it is widely considered an improvement over tail-drop discard policies.
One form of active queue management is the Random Early Detection (RED) algorithm [RED93], actually a family of related algorithms. In one version of RED, an exponentially-weighted moving average of the queue depth is maintained:

When this average queue depth is between a maximum threshold \( \text{max\_th} \), and a minimum threshold \( \text{min\_th} \), packets are dropped with a probability which is proportional to the amount by which the average queue depth exceeds \( \text{min\_th} \).

When this average queue depth is equal to \( \text{max\_th} \), the drop probability is equal to a configurable parameter \( \text{max\_p} \).

When this average queue depth is greater than \( \text{max\_th} \), packets are always dropped. Numerous variants on RED appear in the literature, and there are other active queue management algorithms which claim various advantages over RED [MG01].

With an active queue management algorithm, dropped packets become a feedback signal to trigger more appropriate congestion behavior by the TCPs in the end hosts. Randomization of dropping tends to break up the observed tendency of TCP windows belonging to different TCP connections to become synchronized by correlated drops, and it also imposes a degree of fairness on those connections that properly implement TCP congestion avoidance. Another important property of active queue management algorithms is that they attempt to keep average queue depths short while accommodating large short term bursts.

Since TCP neither knows nor cares whether congestive packet loss occurs at the IP layer or in a subnet, it may be advisable for subnets that perform queuing and discarding to consider implementing some form of active queue management. This is especially true if large aggregates of TCP connections are likely to share the same queue. However, active queue management may be less effective in the case of many queues carrying smaller aggregates of TCP connections, e.g., in an ATM switch that implements per-VC queuing.

Note that the performance of active queue management algorithms is highly sensitive to settings of configurable parameters, and also to factors such as RTT [MBB00] [FB00].

Some subnets, most notably ATM, perform segmentation and reassembly at the subnetwork edges. Care should be taken here in designing discard policies. If the subnet discards a fragment of an IP packet, then the remaining fragments become an unproductive load on the subnet that can markedly degrade end-to-end performance [RF95]. Subnetworks should therefore attempt to discard these extra fragments whenever one of them must be discarded. If the IP packet has already been partially forwarded when discarding becomes necessary, then every remaining fragment except the one marking the end of the IP packet should also be discarded. For ATM subnets, this specifically means using Early Packet Discard and Partial Packet Discard [ATMFTM].

Some subnets might include flow control mechanisms that effectively require that the rate of traffic flows be shaped as they enter the subnet. One example of such a subnet mechanism is in the ATM Available Bit rate (ABR) service category [ATMFTM]. Such flow control mechanisms have the effect of making the subnet nearly lossless by pushing congestion into the IP routers at the edges of
the subnet. In such a case, adequate buffering and discard policies are needed in these routers to deal with a subnet that appears to have varying bandwidth. Whether there is benefit in this kind of flow control is controversial; there are numerous simulation and analytical studies that go both ways. It appears that some of the issues that lead to such different results include sensitivity to ABR parameters, use of binary rather than explicit rate feedback, use (or not) of per-VC queuing, and the specific ATM switch algorithms selected for the study. Anecdotally, some large networks have used IP over ABR to carry TCP traffic, have claimed it to be successful, but have published no results.

Another possible approach to flow control in the subnet would be to work with TCP Explicit Congestion Notification (ECN) semantics [RFC2481] [RFC801]. Routers at the edges of the subnet, rather than shaping, would set the ECN bit in those IP packets that are received in subnet packets that have an ECN indication. Nodes in the subnet would need to implement an active queue management protocol that marks subnet packets instead of dropping them.

ECN is currently a proposed standard, but it is not yet widely deployed.

Compression

Application data compression is a function that can usually be omitted in the subnetwork. The endpoints typically have more CPU and memory resources to run a compression algorithm and a better understanding of what is being compressed. End-to-end compression benefits every network element in the path, while subnetwork-layer compression, by definition, benefits only a single subnetwork.

Data presented to the subnetwork layer may already be in compressed format (e.g., a JPEG file), compressed at the application layer (e.g., the optional "gzip", "compress", and "deflate" compression in HTTP/1.1 [RFC2616]), or compressed at the IP layer (the IP Payload Compression Protocol [RFC2393] supports DEFLATE [RFC2394] and LZO [RFC2395]). In any of these cases, compression in the subnetwork is of no benefit.

The subnetwork may also process data that has been encrypted by the application (OpenPGP [RFC2440] or S/MIME [RFCs-2630-2634]), just above TCP (SSL, TLS [RFC2246]), or just above IP (IPSEC ESP [RFC2406]). Ciphers generate random-looking bit streams lacking any patterns that can be exploited by a compression algorithm.

However, much data is still transmitted uncompressed over the Internet, so subnetwork compression may be beneficial. Any subnetwork compression algorithm must not expand uncompressible data, e.g., data that has already been compressed or encrypted.

We make a stronger recommendation that subnetworks operating at low speed or with small MTUs compress IP and transport-level headers (TCP and UDP) using several header compression schemes developed within the IETF. An uncompressed 40-byte TCP/IP header takes about 33 milliseconds to send at 9600 bps. "VJ" TCP/IP header compression [RFC1144] compresses most headers to 3-5 bytes, reducing transmission time to several milliseconds. This is especially beneficial for small, latency-sensitive packets in interactive sessions.

Similarly, RTP compression schemes such as CRTP [RFC2508] and ROHC
RFC3095] compress most IP/UDP/RTP headers to one to four bytes. The resulting savings are especially significant when audio packets are kept small to minimize store-and-forward latency.

Designers should consider the effect of the subnetwork error rate on the performance of header compression. TCP ordinarily recovers from lost packets by retransmitting only those packets that were actually lost; packets arriving correctly after a packet loss are kept on a resequencing queue and do not need to be retransmitted. In VJ TCP/IP [RFC1144] header compression, however, the receiver cannot explicitly notify a sender about data corruption and subsequent loss of synchronization between compressor and decompressor. It relies instead on TCP retransmission to re-synchronize the decompressor. After a packet is lost, the decompressor must discard every subsequent packet, even if the subnetwork makes no further errors, until the sending TCP retransmits to re-synchronize the decompressor. This effect can substantially magnify the effect of subnetwork packet losses if the sending TCP window is large, as it will often be on a path with a large bandwidth*delay product.

Alternative header compression schemes such as those described in [RFC2507] include an explicit request for retransmission of an uncompressed packet to allow decompressor resynchronization without waiting for a TCP retransmission. However, these schemes are not yet in widespread use.

Both TCP header compression schemes do not compress widely-used TCP options such as selective acknowledgements (SACK). Both fail to compress TCP traffic that makes use of explicit congestion notification (ECN). Work is under way in the IETF ROHC WG to address these shortcomings in a ROHC header compression scheme for TCP. [RFC3095] [RFC3096]

The subnetwork error rate also is important for RTP header compression. CRTP uses delta encoding, so a packet loss on the link causes uncertainty about the subsequent packets, which often must be discarded until the decompressor has notified the compressor and the compressor has sent re-synchronizing information. This typically takes slightly more than a round-trip time. For links that combine significant error rates with latencies that require multiple packets to be in flight at a time, this leads to significant error propagation, i.e. subsequent losses caused by an initial loss.

For links that are both high-latency (multiple packets in flight from a typical RTP stream) and error-prone, RTP ROHC provides a more robust way of RTP header compression, at a cost of higher complexity at the compressor and decompressor. Within a talk spurt, only extended losses of (depending on the mode chosen) 12 to 64 packets typically cause error propagation.

Packet Reordering

The Internet architecture does not guarantee that packets will arrive in the same order in which they were originally transmitted, and transport protocols like TCP must take this into account.

But reordering does come at a cost with TCP as it is currently defined. Because TCP returns a cumulative acknowledgment (ACK) indicating the last in-order segment that has arrived, out-of-order segments cause a TCP receiver to transmit a duplicate acknowledgment.
When the TCP sender notices three duplicate acknowledgments it assumes that a segment was dropped by the network and uses the fast retransmit algorithm [Jac90] [APS99] to resend the segment. In addition, the congestion window is reduced by half, effectively halving TCP’s sending rate. If a subnetwork badly re-orders segments such that three duplicate ACKs are generated, the TCP sender needlessly reduces the congestion window and performance suffers.

Packet reordering does frequently occur in parts of the Internet, and it seems to be difficult or impossible to eliminate [BPS99]. For this reason, research has begun into improving TCP’s behavior in the face of packet reordering.

[BPS99] cites reasons why it may even be undesirable to eliminate reordering. There are situations where average packet latency can be reduced, link efficiency can be increased, and/or reliability can be improved if reordering is permitted. Examples include certain high speed switches within the Internet backbone and the parallel links used over many Internet paths for load splitting and redundancy.

This suggests that subnetwork implementers should try to avoid packet reordering whenever possible, but not if doing so compromises efficiency, impairs reliability or increases average packet delay.

Note that every header compression scheme currently standardized for the Internet requires in-order packet delivery on the link between compressor and decompressor. PPP is frequently used to carry compressed TCP/IP packets; since it was originally designed for point-to-point and dialup links it is assumed to provide in-order delivery. For this reason, subnetwork implementers who provide PPP interfaces to VPNs and other, more complex subnetworks must also maintain in-order delivery of PPP frames.

Mobility

Internet users are increasingly mobile. Not only are many Internet nodes laptop computers, but pocket organizers and mobile embedded systems are also becoming nodes on the Internet. These nodes may connect to many different access points on the Internet over time, and they expect this to be largely transparent to their activities. Except when they are not connected to the Internet at all, and for performance differences when they are connected, they expect that everything will "just work" regardless of their current Internet attachment point or local subnetwork technology.

Changing a host’s Internet attachment point involves one or more of the following steps.

First, if use of the local subnetwork is restricted, the user’s credentials must be verified and access granted. There are many ways to do this. A trivial example would be an "Internet cafe" that grants physical access to the subnetwork for a fee. Subnetworks may implement technical access controls of their own; one example is IEEE 802.11 Wireless Equivalent Privacy [IEEE80211]. And it is common practice for both cellular telephone and Internet service providers (ISPs) to agree to serve each others users; RADIUS [RFC2865] is the standard means for ISPs to exchange authorization information.

Second, the host may have to be reconfigured with IP parameters appropriate for the local subnetwork. This usually includes setting an IP address, default router, and domain name system (DNS) servers.
On multiple-access networks, the Dynamic Host Configuration Protocol (DHCP) [RFC2131] is almost universally used for this purpose. On PPP links, these functions are performed by the IP Control Protocol (IPCP) [RFC1332].

Third, traffic destined for the mobile host must be routed to its current location. This function is the most common meaning of the term "Internet mobility".

Internet mobility can be provided at any of several layers in the Internet protocol stack, and there is ongoing debate as to which are the most appropriate and efficient. Mobility is already an feature of certain application layer protocols; the Post Office Protocol (POP) [RFC1939] and the Internet Message Access Protocol (IMAP) [RFC2060] were created specifically to provide mobility in the receipt of electronic mail.

Mobility can also be provided at the IP layer [RFC2002]. This mechanism provides greater transparency, viz., IP addresses that remain fixed as the nodes move, but at the cost of potentially significant network overhead and increased delay because of the non-optimum network routing and tunneling involved.

Some subnetworks may provide internal mobility, transparent to IP, as a feature of their own internal routing mechanisms. To the extent that these simplify routing at the IP layer, reduce the need for mechanisms like Mobile IP, or exploit mechanisms unique to the subnetwork, this is generally desirable. This is especially true when the subnetwork covers a relatively small geographic area and the users move rapidly between the attachment points within that area. Examples of internal mobility schemes include Ethernet switching and intra-system handoff in cellular telephony.

However, if the subnetwork is physically large and connects to other parts of the Internet at multiple geographic points, care should be taken to optimize the wide-area routing of packets between nodes on the external Internet and nodes on the subnet. This is generally done with "nearest exit" routing strategies. Because a given subnetwork may be unaware of the actual physical location of a destination on another subnetwork, it simply routes packets bound for the other subnetwork to the nearest gateway between the two. This implies some awareness of IP addressing and routing within the subnetwork. The subnetwork may wish to use IP routing internally for wide area routing and restrict subnetwork-specific routing to constrained geographic areas where the effects of suboptimal routing are minimized.

Multicasting

The Internet model includes "multicasting", where IP packets are sent to all the members of a multicast group [RFC1112] [RFC2236]. IP routers organize each multicast group into a spanning tree, and they route multicast packets by making a copy for each output interface that includes at least one downstream member of the multicast group.

Multicasting is considerably more efficient when a subnetwork explicitly supports it. For example, a router relaying a multicast packet onto an Ethernet subnet need send only one copy, no matter how many members of the multicast group are connected to the segment. Without native Ethernet multicast support, the router would have to transmit a separate copy of every multicast packet to every member of
the multicast group on the segment.

Subnetworks using shared channels (e.g., radio LANs, Ethernets, etc) are especially suitable for native multicasting, and their designers should make every effort to support it. This involves designating a section of the subnetwork’s own address space for multicasting and designing receivers to accept packets addressed to some number of multicast addresses in addition to the unicast packets specifically addressed to them. How many multicast addresses are supported depends on the requirements of the associated host or router; at least several dozen will meet most current needs.

On low-speed networks this address recognition function may be readily implemented in host software, but on high speed networks it should be implemented in subnetwork hardware. This hardware need not be complete; for example, many Ethernet interfaces implement a "hashing" function that passes all of the multicast (and unicast) traffic to which the associated host subscribes, plus some small fraction of multicast traffic to which the host does not subscribe. Host software then only has to discard the relatively few unwanted packets that make it past the hardware filter.

**Broadcasting and Discovery**

Link layers fall into two categories: point-to-point and shared. A point-to-point link has exactly two endpoint components (hosts or gateways); a shared link has more than two, either on an inherently broadcast media (e.g., Ethernet, radio) or on a switching layer hidden from the network layer (switched Ethernet, Myrinet [MYR], ATM).

Several Internet protocols which make use of link-layer broadcast capabilities, including link-layer address lookup (ARP), auto-configuration (RARP, BOOTP, DHCP), and routing (RIP). These protocols require broadcast-capable links. Shared links SHOULD support native, link-layer subnet broadcast.

The lack of broadcast can impede the performance of these protocols, or in some cases render them inoperable. ARP-like link address lookup can be provided by a centralized database but at the expense of potentially higher response latency and the need for nodes to have explicit knowledge of the ARP server address.

Other protocols, such as DHCP, cannot function at all without a subnetwork broadcast mechanism.

**Routing**

Many subnetworks provide their own internal routing mechanisms. Since routing is the major function of the Internet layer, the question naturally arises as to the proper division of function between routing at the Internet layer and routing in the subnet.

In general, routing in a subnetwork and at IP is more complementary than competitive. Routing algorithms often have difficulty scaling to very large networks, and a division of labor between IP and a large subnetwork can often make the routing problem more tractable for both.

Some subnetworks have special features that allow the use of more effective or responsive routing mechanisms that cannot be implemented
in IP because of its need for generality. One example is the self-
learning bridge algorithm widely used in Ethernet networks. Another
is the "handoff" mechanism in cellular telephone networks,
particularly the "soft handoff" scheme in IS-95 CDMA.

On the other hand, routing optimality can suffer when a subnetwork’s
routing architecture hides internal structure that an IP router could
have used to make more efficient decisions. Such situations occur
most often when the subnetwork covers a large geographic area and
includes links of widely varying capacities, but presents itself to
IP as a single, fully-connected network with uniform metrics between
border nodes.

The subnetwork designer who decides to implement internal routing
should also consider whether a custom routing algorithm is warranted,
or if an existing Internet routing algorithm or protocol may suffice.
Protocols and routing algorithms can be notoriously subtle, complex
and difficult to implement correctly. Much work can be avoided if an
existing protocol or off-the-shelf product can be readily used.

Security Considerations

Security has become a high priority in the design and operation of
the Internet. The Internet is vast, and countless organizations and
individuals own and operate its various components. A consensus has
emerged for what might be called a "security placement principle": a
security mechanism is most effective when it is placed as close as
possible to, and under the direct control of the owner of, the asset
that it protects.

The most important conclusion that follows from this principle is
that end-to-end security (e.g., confidentiality, integrity and access
control) cannot be ensured with subnetwork security mechanisms. Not
only are end-to-end security mechanisms much more closely associated
with the end-user assets they protect, they are also much more
comprehensive. For example, end-to-end security mechanisms cover gaps
that can appear when otherwise good subnetwork mechanisms are
concatenated. This is an important application of the end-to-end
principle [SRC81].

Several security mechanisms that can be used end-to-end have already
been deployed in the Internet and are enjoying increasing use. The
most important are the Secure Sockets Layer (SSL) [SSL2] [SSL3] and
TLS [RFC2246] primarily used to protect web commerce; Pretty Good
Privacy (PGP) [RFC1991], primarily used to protect and authenticate
email and software distributions; the Secure Shell (SSH) [SSH], used
for secure remote access and file transfer; and IPSEC [RFC2401], a
general purpose encryption and authentication mechanism that sits
just above IP and can be used by any IP application. (IPSEC can
actually be used either on an end-to-end basis or between security
gateways that do not include either or both end systems.)

Nonetheless, end-to-end security mechanisms are not used as widely as
might be desired. However, the group could not reach consensus on
whether subnetwork designers should be actively encouraged to
implement mechanisms to protect user data.

The majority of the working group held that subnetwork security
mechanisms, especially when weak or incorrectly implemented [BGW],
may actually be counterproductive. The argument is that subnetwork
security mechanisms can lull end users into a false sense of
security, diminish the incentive to deploy effective end-to-end mechanisms, and encourage "risky" uses of the Internet that would not be made if users understood the inherent limits of subnetwork security mechanisms.

The other point of view encourages subnetwork security on the principle that it is better than the default situation, which all too often is no security at all. Users of especially vulnerable subnets (such as consumers who have wireless home networks and/or shared media Internet access) often have control over at most one endpoint -- usually a client -- and therefore cannot enforce the use of end-to-end mechanisms. However, subnet security can be entirely adequate for protecting low-valued assets against the most likely threats. In any event, subnet mechanisms do not preclude the use of end-to-end mechanisms, which are typically used to protect high valued assets. This viewpoint recognizes that many security policies implicitly assume that the entire end-to-end path is composed of a series of concatenated links that are nominally physically secured. That is, these policies assume that all endpoints of all links are trusted and that access to the physical media by attackers is difficult. To meet the assumptions of such policies, explicit mechanisms are needed for links (especially shared media links) that lack physical protection. This, for example, is the rationale that underlies Wired Equivalent Privacy (WEP) in the IEEE 802.11 wireless LAN standard, and the Baseline Privacy Interface in the DOCSIS data over cable television networks standards.

We therefore recommend that subnetwork designers who choose to implement security mechanisms to protect user data be as candid as possible with the details of such security mechanisms and the inherent limits of even the most secure mechanisms when implemented in a subnetwork rather than on an end-to-end basis.

In keeping with the "placement principle", a clear consensus exists for another subnetwork security role: the protection of the subnetwork itself. Possible threats to subnetwork assets include theft of service and denial of service; shared media subnets tend to be especially vulnerable to such attacks. In some cases, mechanisms that protect subnet assets can also improve (but NOT ensure) end-to-end security.

Another potential role for subnetwork security is to protect users against traffic analysis, i.e., identifying the communicating parties and determining their communication patterns and volumes even when their actual contents are protected by strong end-to-end security mechanisms. Lower-layer security can be more effective against traffic analysis due to its inherent ability to aggregate the communications of multiple parties sharing the same physical facilities while obscuring higher layer protocol information that indicates specific end points, such as IP addresses and TCP/UDP port numbers.

However, traffic analysis is a notoriously subtle and difficult threat to understand and defeat, far more so than threats to confidentiality and integrity. We therefore urge extreme care in the design of subnetwork security mechanisms specifically intended to thwart traffic analysis.

Subnetwork designers must keep in mind that design and implementation for security is difficult [Schneier2] [Schneier3]. [Schneier1] describes protocols and algorithms which are considered well
understood and believed to be sound.

Poor design process, subtle design errors and flawed implementation can result in gaping vulnerabilities. In recent years, a number of subnet standards have had problems exposed. The following are examples of the mistakes that have been made:

1. Use of weak and untested algorithms [Crypto9912] [BGW]. For a variety of reasons, algorithms were chosen which had subtle flaws that made them vulnerable to a variety of attacks.

2. Use of 'security by obscurity' [Schneier4] [Crypto9912]. One common mistake is to assume that keeping cryptographic algorithms secret makes them more secure. This is intuitive, but wrong. Full public disclosure early in the design process attracts peer review by knowledgeable cryptographers. Exposure of flaws by this review far outweighs any imagined benefit from forcing attackers to reverse engineer security algorithms.

3. Inclusion of trapdoors [Schneier4] [Crypto9912]. Trapdoors are flaws surreptitiously left in an algorithm to allow it to be broken. This might be done to recover lost keys or to permit surreptitious access by governmental agencies. Trapdoors can be discovered and exploited by malicious attackers.

4. Sending passwords or other identifying information as clear text. For many years, analog cellular telephones could be cloned and used to steal service. The cloners merely eavesdropped on the registration protocols that exchanged everything in clear text.

5. Keys which are common to all systems on a subnet [BGW].

6. Incorrect use of a sound mechanism. For example [BGW], one subnet standard includes an initialization vector which is poorly designed and poorly specified. A determined attacker can easily recover multiple ciphertexts encrypted with the same key stream and perform statistical attacks to decipher them.

7. Identifying information sent in clear text that can be resolved to an individual, identifiable device. This creates a vulnerability to attacks targeted to that device (or its owner).

8. Inability to renew and revoke shared secret information.

9. Insufficient key length [Blaze96].

10. Failure to address "man-in-the-middle" attacks, e.g., with mutual authentication.

This list is by no means comprehensive. Design problems are difficult to avoid, but expert review is generally invaluable in avoiding problems.

In addition, well-designed security protocols can be compromised by implementation defects. Examples of such defects include use of predictable pseudo-random numbers [RFC1750], vulnerability to buffer overflow attacks due to unsafe use of certain I/O system calls [WFBA2000], and inadvertent exposure of secret data.

References
References of the form RFCnnnn are Internet Request for Comments (RFC) documents available online at www.rfc-editor.org.


ecn-00.txt, draft-ietf-tsvwg-ecn-tunnels-00.txt, and draft-ietf-ipsec-ecn-02.txt. The IESG approved this document to go to Proposed Standard on June 12, 2001.


[RFC1750] D. Eastlake, S. Crocker, J. Schiller, "Randomness Recommendations for Security", December 1994 (this will be obsoleted soon... check before submitting to RFC editor)


[ Schneier3] Schneier, Bruce "Why Cryptography is Harder Than it Looks", www.counterpane.com


Authors’ Addresses:

Phil Karn, editor (karn@qualcomm.com)
Carsten Bormann (cabo@tzi.org)
Gorry Fairhurst (gorry@erg.abdn.ac.uk)
Aaron Falk (afalk@panamsat.com)
Dan Grossman (dan@dma.isg.mot.com)
Reiner Ludwig (Reiner.Ludwig@ericsson.com)
Jamshid Mahdavi (mahdavi@novell.com)
Saverio Mascolo (mascolo@poliba.it)
Gabriel Montenegro (Gabriel.Montenegro@eng.sun.com)
Marie-Jose Montpetit (marie@teledesic.com)
Joe Touch (touch@isi.edu)
Lloyd Wood (lwood@cisco.com)