NASA/CR—2000-209785

Trade-off Analysis Report

Chris Dhas
Computer Networks and Software, Springfield, Virginia

January 2000

NASA/CR—2000-209785

Trade-off Analysis Report

Chris Dhas
Computer Networks and Software, Springfield, Virginia

Prepared under Contract NAS3–99165, Task 1

National Aeronautics and
Space Administration

Glenn Research Center

January 2000

In-Space Internet Node Technology Development Project
Trade-off Analysis Report

Table of Contents

Table of Contents

Section **Page**

List of Figures

List of Tables

1. INTRODUCTION

NASA's Glenn Research Center (GRC) defines and develops advanced technology for high priority national needs in communications technologies for application to aeronautics and space. GRC tasked Computer Networks & Software Inc. (CNS) to examine protocols and architectures for an In-Space Internet Node. CNS has developed a methodology for network reference models to support NASA's four mission areas:

- Earth Science
- Space Science
- Human Exploration and Development of Space (HEDS)
- Aerospace Technology

CNS previously developed a report which applied the methodology to three space Internet-based communications scenarios for future missions. CNS conceptualized, designed, and developed space Internet-based communications protocols and architectures for each of the independent scenarios. GRC selected for further analysis the scenario that involved unicast communications between a Low-Earth-Orbit (LEO) International Space Station (ISS) and a ground terminal Internet node via a Tracking and Data Relay Satellite (TDRS) transfer.

This report contains a tradeoff analysis on the selected scenario. The analysis examines the performance characteristics of the various protocols and architectures. The tradeoff analysis incorporates the results of a CNS developed analytical model that examined performance parameters.

2. ENVIRONMENT

The scenario analyzed involves unicast communications between a LEO ISS and a ground terminal via a TDRS transfer node. The ground terminal was assumed to be part of the terrestrial Internet infrastructure. The terrestrial infrastructure may be provided by a carrier or may be based on private network architectures. The ISS to ground terminal via TDRS scenario supports multimedia applications. The applications supported are presented in Table 1. It is assumed that multimedia traffic forms a significant portion of the traffic in this scenario.

2.1. NASA Application Types and Characteristics

The existing NASA application categories with the associated application types and their respective characteristics are shown in Table 1. The characteristics formed the basis for assessing the protocol functional requirements at each layer in the protocol stack for the applications to be supported. The applications vary from command and control to entertainment. The table shows that the majority of the applications do not require security. The response time ranges from seconds to minutes. The precedence levels range from high to low. In addition, message integrity ranges from high to low. The availability is spread over 0.999 to 0.99999.

The bandwidth requirement ranges from 64 Kbps to 100 Mbps, a wide range. The high traffic load is mainly due to multimedia and Experiment Support/Mission Payload applications. The total daily traffic is found by summing the column defined as the total bandwidth requirements. The peak hour load to be supported by the system is 15% of the total daily traffic. Therefore, the estimated peak hour load is 24.01 Mbps.

Table 1. Application Profile for Selected Architecture: ISS ↔ TDRS ↔ Ground Terminal

Application Category	Transfer Unit	Response Time	Load Factor	Total Bandwidth Requirements	Precedence	Integrity	Availability	Security	Scalability
Command and Control	Small Async Frequent	1 - 3 Sec	4	1 Mbps	High	High	99.999	A+E+C	4-10 Slow
Telemetry/Navigation	Medium Sync Continuous	1 - 3 Sec	10	5 Mbps	Medium	High	99.99	E	10-100 Moderate
Emergency/Flight Critical	Small Sync Infrequent	1-3 Sec	1	64 Kbps	High	High	99.999	NR	1-10 Slow
Experiment Support/ Mission Payload	Large Async Continuous	10-20 Min	100	100 Mbps	Medium	Medium	99.99	NR	100-1000 Moderate
System/Vehicle Maintenance	Large Async Medium	10-20 Min	10	10 Mbps	Medium	Medium	99.9	NR	10-50 Fast
Administrative	Medium Async Frequent	1-3 Min	4	4 Mbps	Low	Low	99.9	NR	1-10 Moderate

Application Category	Transfer Unit	Response Time	Load Factor	Total Bandwidth Requirements	Precedence	Integrity	Availability	Security	Scalability
Entertainment	Large Async Frequent	NA	4	40 Mbps	Low	Low	99.9	NR	1-100 Fast

3. SELECTED PROTOCOL ARCHITECTURE

The ISS to ground terminal via TDRS scenario environment consists of a ground segment and a space segment. Figure 1 shows the block diagram of the networking environment. The terrestrial Internet nodes are assumed to be part of the terrestrial Internet infrastructure. The terrestrial infrastructure may be provided by a carrier or may be based on a private network architecture. It is assumed that these nodes interface to the network using a COTS TCP/IP protocol stack. The protocol architecture of the terrestrial Internet nodes is not part of the protocol architecture analysis.

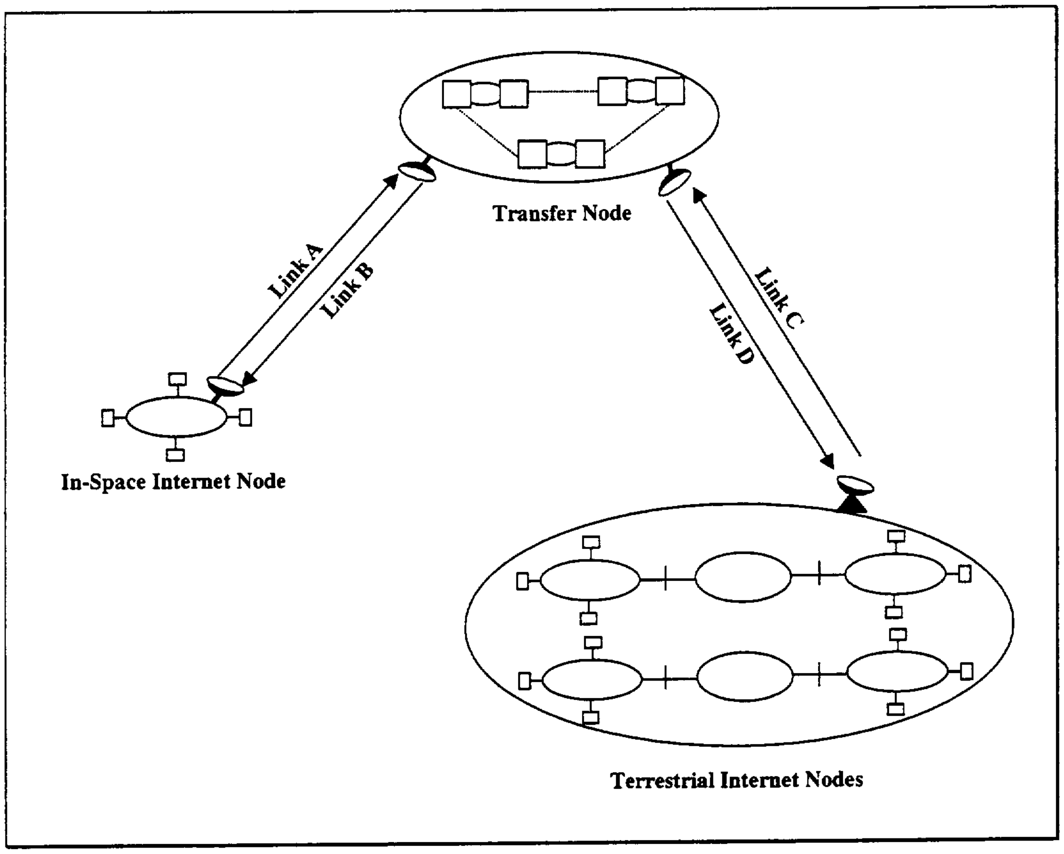

Figure 1. Selected Scenario: Space Station ↔ TDRS ↔ Ground Terminal

It is assumed that ISS consists of an intra-network that supports communications functions that meet the internal requirements of the space station. The capability of the ISS may include on-board communications and on-board data handling resources, including those with limited on-board computer and memory resources, as well as those with multiple, high-capacity on-board computers with extensive data storage. The ISS communicates with the TDRS via the paths identified as Link A and Link B. The links that support communication between the TDRS and the ground terminals are marked as Link C and Link D. The analysis in this report covers a bandwidth range of 1 to 155 Mbps.

3.1. Tracking and Data Relay Satellite System Characteristics

The Tracking and Data Relay Satellite (TDRS) system is a constellation of geostationary orbiting satellites effectively providing full-time global coverage by inter-satellite space/space links with the primary ground/space link to the White Sands Ground Terminal (WSGT) at Las Cruces, NM. The TDRS satellites are positioned at 22,300 miles (35,887 kilometers) over the equator in orbital slots approximately 130 degrees apart. The onboard TDRS transponders cover a range of bandwidths:

- TDRS Forward Link: $\leq$ 25 Mbps (K-band), $\leq$ 300 Kbps (S-band)
- TDRS Return Link: $\leq$ 300 Mbps (K-band), $\leq$ 6 Mbps (Single Access S-band), up to 5 users at $\leq$ 3 Mbps each (Advanced TDRS Multiple Access S-band)

3.2. Communication and End System Environment

The space node operations environment comprises the ground information infrastructure, with high-speed computing and communications capabilities. The space information infrastructure presents a significantly different environment from the ground systems. In space, there are relatively few end systems and networks and the communications requirements are driven by the extreme mass, power, and volume constraints, together with the delay and expense of developing space-qualified hardware.

The space communications environment is further complicated by the characteristics of the space/ground link. With rare exceptions, connectivity to a space vehicle is intermittent. The duty cycle is usually below 10% due to limited visibility from ground stations and contention among missions for scarce contact time. Limited signal strength and noise make data loss through corruption far more likely than in ground networks. Long propagation times cause terrestrial protocols to operate inefficiently or fail to function at all.

There are several existing protocols that must be supported, including standard protocols to support reliable data transfer, evolving multi-node mission configurations that require in-space network routing, and protocols that provide compatibility and interoperability with the Internet.

3.3. Transmission Facility for Selected Architecture

There are effectively four parts to the transmission facilities for this scenario (Figure 1):

- Space station to TDRS forward link (Link A)

- TDRS to space station return link (Link B)
- Terrestrial Internet nodes to TDRS forward link (Link C)
- TDRS to terrestrial Internet nodes return link (Link D)

The space station interfaces (i.e., Link A and Link B) are at the physical layer for the space/space link. Internally, the data units are interfaced to the onboard systems at the respective levels of the protocol stack. As compared to the ground segment, the space station capabilities are fixed and inflexible. Therefore, the transmission facility selection defaults to the space station transmission systems. The space station is assumed to have the necessary onboard interfaces (logical and physical) and transmission facilities to satisfy mission-specific application requirements.

The TDRS is a "bent pipe" because it functions only to relay the transmission bit stream. This is a physical layer function of interfacing the mission bit stream (transmitted data) with the transmission medium.

The ground segment transmission facilities must accommodate the ground/space link (Link C) as well as interface to the terrestrial Internet node. The terrestrial Internet node interfaces are assumed to include any protocol conversion and signal handling capabilities necessary to meet the terrestrial network requirements. Typical ground node to TDRS forward link (Link C) and TDRS to ground station return link (Link D) ground terminal transmission facilities are considered to be comprised of the antenna subsystems (S-band, Ku-band, Ka-band and C-band) and associated terminal equipment to fully support any protocol architecture requirements.

3.3.1. Operational Constraints

Ideally, the requirements identified above would be met through use of off-the-shelf technology that has been proven in ground-based systems. Unfortunately, space missions must be carried out under conditions that vary significantly from those in ground data systems. Therefore, the operational constraints encountered in space communications listed below should be taken into account in developing the conceptual protocol architecture.

- Round-trip delays much greater than those seen in ground networks.

- Intermittent connectivity, as a result of orbital position, earth rotation, and availability of ground station support.

- Variation in the format and performance characteristics of the space links used in space missions.

- Changes in the routing path from contact to contact because of the use of multiple ground stations or changes of the relative positions of multiple spacecraft.

- Noise characteristics on space links that (despite sophisticated error correction codes) produce more frequent data loss than on ground links.

- The asymmetry in the bandwidth available for the space station to TDRS link and the TDRS to ground terminal link needs to be taken into account. This asymmetry may affect the features of protocols that support end-to-end communications.

3.4. Scenario Unique Requirements

Although functionally equivalent to terrestrial networks, space communications networks often have performance and operational considerations that prevent direct use of existing commercial protocols. Protocols used in terrestrial networks assume that: connectivity is maintained; data loss due to corruption is rare; balanced bi-directional links are available; and most data loss is due to congestion. Furthermore, vendors of commercial communications products that implement these protocols use these assumptions to maximize performance and economy in this environment. This results in the treatment of retransmission, recovery, and time-outs inappropriate for space operations. For the large majority of space nodes, the space environment makes performance of these protocols unacceptable.

To meet the above constraints, protocols are required to provide interoperability across the spectrum of space nodes and between space data systems and the COTS based ground network environment. They have to provide a set of options and protocol data unit formats that can be scaled to satisfy the communication needs of both complex and simple nodes.

The protocol architecture should provide reliable stream or file transfers over existing and new link layers plus dynamic networking for multiple space node environments. Given the link characteristics and intermittent connectivity encountered over the space link, better performance is achieved by using a balance of upper-layer, confirmed, end-to-end services supported by link level error correction that avoids excessive retransmission. In addition, the architecture must be able to support multimedia applications.

3.5. Recommended Protocol Architecture

The goal of the protocol architecture is not only to support applications but also to lower the lifecycle costs by reducing development and operational costs in space communications systems. In addition, the protocol architecture should be able to extend Internet connectivity into space. The rationale for this approach is that both the data systems and the personnel (designers, operators, and users) associated with space missions are already using Internet protocols. The communications services that they need in space are similar to those they have in ground networks. The easiest, lowest risk, and most direct way to achieve this goal is to adapt the protocols that are used on the ground.

Although the Internet protocols provide an excellent basis for space communications protocol development, the space environment presents a number of constraints that are seldom encountered in the design of terrestrial data communications networks. There are physical, operational, and resource differences. The physical differences include:

- Space link delays ranging from milliseconds to hours.
- Potentially noisy space data links.
- Limited space link bandwidth.
- Variation in sub-network types from simple busses to local and wide area networks.
- Interruptions in the end-to-end data path that can vary from bits lost, and link interruptions.

The operational differences include:

- Inherently sporadic nature of contact between space and ground.
- Maximum latency requirement due to "teleoperations" activities.

The resource differences include:

- Limited onboard processing power.
- Limited onboard program memory.
- Limited onboard data buffering.
- Asymmetry in bandwidth between forward and return links.

Except for a very narrow range of operational conditions, the current off-the-shelf, Internet protocols do not satisfy the requirements encountered in the space mission environment. Therefore, the strategy is to use COTS-supported standards wherever possible; capitalize on established user interface familiarity; and minimize software development costs. This approach allows one to take advantage of the hundreds of thousands of hours of operational experience that the Internet protocols have accrued.

3.6. NASA Conceptual Protocol Architecture

The suite of NASA space protocols should provide flexibility and features that allow designers to tailor a communications protocol suite to meet specific requirements and constraints without extensive software development. It should allow specific layers or protocols within layers to be included or omitted to create an optimal profile. This calls for a layered protocol architecture. The protocol architecture is shown in Figure 2.

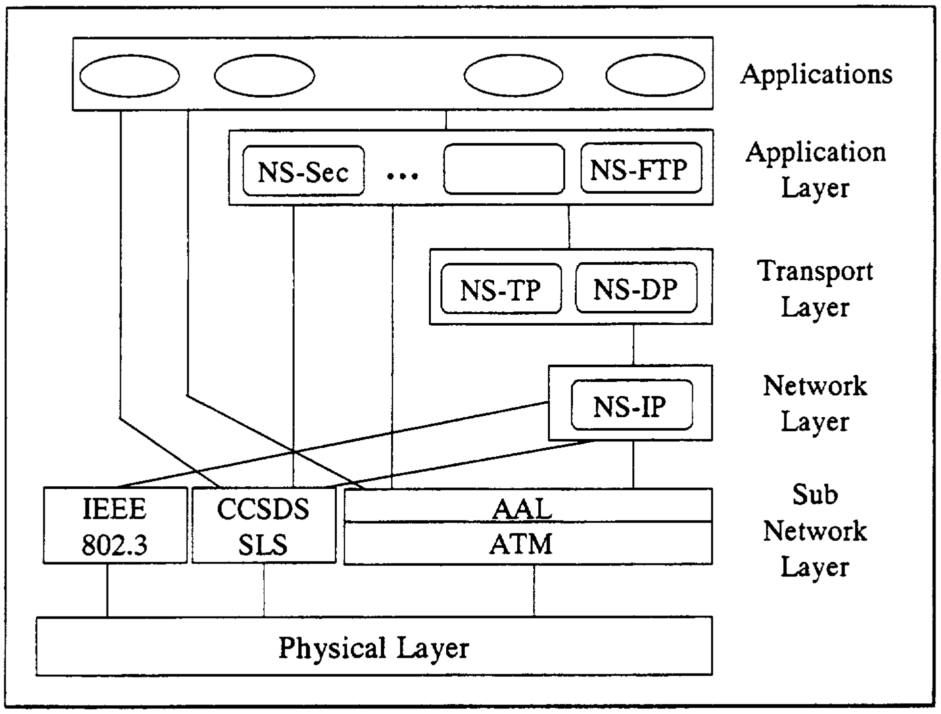

Figure 2. International Space Station ↔ TDRS ↔ Ground Terminal Architecture

Figure 2 shows the NASA Space (NS) conceptual protocol architecture. As mentioned before, this scenario supports data and multimedia applications. In addition, there is a need to provide bit efficient protocols to support these applications. The conceptual protocol architecture consists of an application layer supported by the transport layer. The application layer protocols consists of NASA Space File Transfer Protocol (NS-FTP) and the NASA Space Security Protocol (NS-Sec). This protocol architecture is very flexible. It is easy to add more application layer protocols as requirements for such protocols arise.

The transport layer supports a reliable end-to-end protocol called NS-TP. The NS-TP supports functions that are similar to TCP and ISO TP4. In order to meet the space based communications requirements, modifications need to be made to this protocol. Some of the issues are identified in the transport layer section.

The space based applications presented in Table 1 require a connectionless transport protocol. The NASA Space Datagram Protocol (NS-DP) provides support for this service. NS-DP supports a minimum capability. It is similar to the ISO Connectionless transport protocol and TCP/IP's User Datagram Protocol.

The NS transport layer interfaces to the connectionless NS-IP protocol. The functions supported by NS-IP are similar to those of ISO's CLNP and TCP/IP's IP protocols. But in order to operate efficiently in a space based communications environment, a number of enhancements are identified.

For data only applications, the connectionless NS-IP interfaces to the existing CCSDS space link subnetwork (SLS) subnetwork layer. The user has the option of interfacing to the Ethernet (IEEE 802.3) link layer protocols within the space station. To support multimedia applications in a integrated fashion, ATM has been selected as an alternate subnetwork layer. Note that the ATM technology is capable of replacing the transport and network layers.

3.6.1. Transport Layer Protocol

NS-TP provides end-to-end full and minimal reliability services. The full reliability service is provided by enhanced TCP. It supports the end-to-end data transfer of a sequence of data units with full reliability (complete, correct, in sequence, and no duplication). It uses sequence checking to assure the sequence order and avoid duplication. It incorporates acknowledgments and retransmission requests to provide completeness. This protocol closes connections without loss of data.

The minimal reliability service is provided by the NASA Space Datagram Protocol (NS-DP). It is a connectionless transport protocol and sends data in datagrams. It transfers data with minimal reliability (correct, possibly incomplete, possibly out of sequence). It does not support sequence numbering, acknowledgment of receipt, or retransmissions.

The NS-TP is based on the Transmission Control Protocol (TCP). Enhancements are identified to improve performance in the space environment. Unmodified TCP performs poorly in communications scenarios with a large bandwidth-delay product and cannot operate efficiently with the unbalanced links typical of space-ground communications. The suggested extensions to the base protocols are intended to address the performance issues.

4. PROTOCOL ARCHITECTURE RELATED ISSUES

4.1. TCP Performance Issues

This section presents the aspects of TCP that directly affect the transport layer throughput. Specifically, the focus is on a particular aspect of throughput, namely the effective transmission rate of valid data that a transport connection can achieve.

4.2. TCP Throughput Issues

The Transmission Control Protocol (TCP) is the primary transport protocol in the TCP/IP protocol suite. It implements a reliable byte stream over the unreliable datagram service provided by IP. As part of implementing the reliable service, TCP is also responsible for flow and congestion control: ensuring that data is transmitted at a rate consistent with the capacities of both the receiver and intermediate links in the network path. Since there may be multiple TCP connections active in a link, TCP is also responsible for ensuring that a link's capacity is responsibly shared among the connections using it. Therefore, the NS transport layer throughput issues are similar to those exhibited by the TCP in the TCP/IP protocol architecture.

Therefore, the major features of TCP that affect performance are examined to provide a guideline in selecting transport layer parameters. Many of these performance issues have been discovered over the past few years as link transmission speeds have increased and the so called high "bandwidth×delay" paths (paths where the product of the available path bandwidth and path delay is big) have become common.

4.3. Throughput Expectations

TCP throughput determines how fast most applications can move data across a network. Application protocols such as HTTP (the World Wide Web protocol) and the File Transfer Protocol (FTP) rely on TCP to carry their data. Thus, TCP performance directly impacts application performance.

While there are no formal TCP performance standards when sending large datagrams (to minimize the overhead of the TCP and IP headers), a TCP connection should be able to fill the available bandwidth of a path and to share the bandwidth with other users. If a link is otherwise idle, a TCP connection is expected to be able to fill it. If a link is shared with three other users, each TCP connection is expected to get a reasonable share of the bandwidth.

These expectations reflect a mix of practical concerns. When higher speed lines are used, it is generally expected that the TCP transfers will go faster. Some need faster lines to meet an increase in aggregate traffic. Others have a particular application that requires more bandwidth.

The requirement that TCP share a link effectively reflects the needs of aggregation; all users of a faster link should see improvement. The requirement that TCP fill an otherwise idle link reflects the needs of more specialized applications.

5. TRANSPORT LAYER ISSUES AND ANALYSIS

Transport layer protocols support end-to-end communication between applications using the connectivity provided by an underlying network. The transport layer in the protocol architecture corresponds with Layer 4 of the OSI Reference Model. The transport layer in the OSI architecture provides end-to-end service. It supports this service using either a reliable connection oriented protocol or a connectionless protocol. The OSI protocol architecture uses the TP4 protocol at the transport layer to provide a reliable transport connection. The TCP/IP protocol architecture uses the TCP protocol to provide the reliable transport service. Between the two connection oriented transport protocols, TCP is the most widely used. There is a wealth of research material on the TCP protocol that can be used as a model for identifying services for the transport layer protocol of the NS protocol architecture. Note that these results can be used in the design of an efficient transport protocol for the NASA environment.

5.1. Primary Source of Data Loss

Data loss due to bit-errors and to topological instability is rare in the terrestrial environment. The primary source of loss in terrestrial networks is congestion. The space and mobile communications environment are mixed-loss environments, with losses occurring due to all three causes: bit-errors, topology changes (link outages), and congestion. To treat all losses as congestion results in unnecessary reductions in the offered load. In addition, the increased round trip times in a satellite environment delay the restoration of full-rate transmission.

5.2. Transport Layer Performance

The TCP protocol was designed to operate reliably over almost any transmission medium regardless of transmission rate, delay, corruption, duplication, or reordering of segments. The introduction of fiber optics and a new generation of satellites are resulting in ever-higher transmission speeds. TCP performance problems arise when the bandwidth×delay product is large. This environment is referred to as a "long, fat pipe", and a network containing this path is an "LFN". High-capacity packet satellite channels are LFN's. For example, a DS1-speed satellite channel has a bandwidth×delay product of 10^6 bits or more. This corresponds to 100 outstanding TCP segments of 1,200 bytes each. Terrestrial fiber-optical paths will also fall into the LFN class. For example, a cross-country delay of 30 ms at a DS3 bandwidth (45 Mbps) also exceeds 10^6 bits. There are three fundamental performance problems with the current TCP over LFN paths. The future transport layer protocol (whether TCP or another) needs to address these issues.

5.2.1. TCP Transmission Window Size Limit

The purpose of the transmission window is to allow the receiving TCP to control how much data is being sent to it at any given time. The receiver advertises a window size to the sender. The window measures (in bytes) the amount of unacknowledged data that the sender can have in transit to the receiver. The distinction between the sequence numbers and the window is that sequence numbers are designed to allow the sender to keep track of the data in flight, while the window's purpose is to allow the receiver to control the rate at which it receives data.

Obviously, if a receiver advertises a small window, it is impossible for TCP to achieve high transmission rates. Plus, many implementations do not offer a very large window size (a few kilobytes is typical). However, there is a more serious problem. The standard TCP window size cannot exceed 64 KB, because the field in the TCP header used to advertise the window is only 16 bits wide. This limits the TCP effective bandwidth to 2^{16} bytes divided by the round-trip time of the path. For long delay links, such as those through satellites with a geosynchronous orbit (GEO), this limit gives a maximum data rate of just under 1 Mbps.

The TCP version 4 header uses a 16 bit field to report the receive window size to the sender. Therefore, the largest window that can be used is $2^{16} = 65$ Kbytes. To overcome this problem, the optional "Window Scale" can be used to allow windows larger than 2^{16}. This option defines an implicit scale factor, which is used to multiply the window size value found in a TCP header to obtain the true window size. As part of the changes to add timestamps to the sequence numbers, the End-To-End Research Group and IETF also enhanced TCP to negotiate a window scaling option. The option multiplies the value in the window field by a constant. The effect is that the window can only be adjusted in units of the multiplier. So if the multiplier is 4, an increase of 1 in the advertised window means the receiver is opening the window by 4 bytes.

The window size is limited by the sequence space. The window must be no larger than one half of the sequence space so that it is unambiguously clear that a byte is inside or outside the window. So the maximum multiplier permitted is 2^{14}. This means the maximum window size is 2^{30} and the maximum date rate over a GEO satellite link is approximately 15 Gbps. Given that Tbps data rates can be achieved in terrestrial fiber links, this value is small. But in the absence of a major change to the TCP header format, it is not clear how to fix the problem. Therefore, the transport layer design for the NASA Space Transport protocol should address this issue.

5.2.2. Recovery from Losses

Packet losses in an LFN can have a catastrophic effect on throughput. Until recently, properly operating TCP implementations would cause the data pipeline to drain with every packet loss, and require a slow-start action to recover. Recently, the Fast Retransmit and Fast Recovery algorithms have been introduced. Their combined effect is to recover from one packet loss per window, without draining the pipeline. However, more than one packet loss per window typically results in a retransmission timeout and the resulting pipeline drain and slow start.

In addition, expanding the window size to match the capacity of an LFN results in a corresponding increase of the probability of more than one packet per window being dropped. This could have a significant effect upon the throughput of TCP over an LFN. If a congestion control mechanism based upon some form of random dropping were introduced into gateways, randomly spaced packet drops would become common, possibly increasing the probability of dropping more than one packet per window.

To generalize the Fast Retransmit/Fast Recovery mechanism to handle multiple packets dropped per window, selective acknowledgments are required. Unlike the normal cumulative acknowledgments of TCP, selective acknowledgments give the sender a complete picture of which segments are queued at the receiver and which have not yet arrived. Some evidence in favor of selective acknowledgments has been published and selective acknowledgments have been included in a number of experimental Internet protocols -- VMTP, NETBLT, RDP, and has been proposed for OSI TP4. However, in the non-LFN environment, selective acknowledgments reduce the number of packets retransmitted but do not otherwise improve performance, making their complexity of questionable value. However, selective acknowledgments are expected to become much more important in the LFN regime.

5.2.3. Round-Trip Measurement

TCP implements reliable data delivery by retransmitting segments that are not acknowledged within some retransmission timeout (RTO) interval. Accurate dynamic determination of an appropriate RTO is essential to TCP performance. RTO is determined by estimating the mean and variance of the measured round-trip time (RTT); i.e., the time interval between sending a segment and receiving an acknowledgment for it.

A new TCP option called "Timestamps" is defined and this option allows nearly every segment, including retransmissions, to be timed at negligible computational cost. This mechanism is called Round Trip Time Measurement (RTTM) to distinguish it from other uses of the Timestamps option.

5.2.4. TCP Reliability

TCP keeps track of all data in transit by assigning each byte an unique sequence number. The receiver acknowledges received data by sending an acknowledgment, which indicates that the receiver has received all data up to a particular byte number. TCP allocates its sequence numbers from a 32-bit wraparound sequence space. To ensure that a given sequence number uniquely identifies a particular byte, TCP requires that no two bytes with the same sequence number be active in the network at the same time. But, the IP datagram carrying the TCP segment was assumed to live for up to two minutes. Thus, when TCP sends a byte in an IP datagram, the sequence number of that byte cannot be reused for two minutes. Unfortunately, a 32-bit sequence space spread over two minutes gives a maximum data rate of only 286 Mbps.

High transfer rate enters into TCP performance through the bandwidth×delay product. However, high transfer rate can affect reliability of data transfer in the transport layer by violating the assumptions behind the duplicate detection and sequencing. Error may result from an accidental reuse of TCP sequence numbers in data segments. Suppose that an "old duplicate segment"; e.g., a duplicate data segment that was delayed in Internet queues is delivered to the receiver at the wrong moment such that its sequence numbers fall somewhere within the current window. There would be no checksum failure to warn of the error, and the result could be an undetected corruption of the data. Arrival of an old duplicate ACK segment at the transmitter could also cause serious problems. It is likely to lock up the connection so that no further progress can be made, forcing a reset on the connection.

Reliability in the transport layer is based on the bound on the lifetime of a segment ("Maximum Segment Lifetime" or MSL). An MSL is generally required by any reliable transport protocol, since every sequence number field must be finite, and therefore any sequence number may eventually be reused. In the Internet protocol suite, the MSL is bound to the IP-layer mechanism, the "Time-to-Live" or TTL field.

Duplication of sequence numbers might happen in either of two ways: Sequence number wrap-around on the current connection and an earlier incarnation of the connection. A TCP sequence number contains 32 bits. At a high enough transfer rate, the 32-bit sequence space may be "wrapped" (cycled) within the time that a segment is delayed in queues.

In the earlier incarnation of the connection scenario, suppose that a connection terminates either by a proper close sequence or due to a host crash and the same connection (i.e., using the same pair of sockets) is immediately reopened. A delayed segment from the terminated connection could fall within the current window for the new incarnation and be accepted as valid.

Duplicates from earlier incarnations are avoided by enforcing the current fixed MSL of the TCP. However, avoiding the reuse of sequence numbers within the same connection requires an MSL bound that depends upon the transfer rate, and at high enough rates, a new mechanism is required.

More specifically, if the maximum effective bandwidth at which TCP is able to transmit over a particular path is B bytes per second, then the following constraint must be satisfied for error-free operation: $2^{31} / B > \text{MSL (secs)}$ (V. Jacobson). Table 2 shows the value for *Twrap* = $2^{31}/B$ in seconds, for various line speeds.

Table 2. Twrap for Various Line Speeds

Link Speed in Bits/sec	Bytes/sec	Twrap in secs
56 K	7 K	3×10^5 (~ 3.6 days)
1.5 M	190 K	10^4 (~ 3 hours)
10 M	1.25 M	1700 (~30 mins)
45 M	5.6 M	380
100 M	12.5 M	170
1 G	125 M	17

It is clear that wrap-around of the sequence space is not a problem for 56 Kbps packet switching or even 10 Mbps Ethernets. On the other hand, at DS3 and FDDI speeds, *Twrap* is comparable to the 2 minute MSL assumed by the TCP specification. Moving towards gigabit speeds, *Twrap* becomes too small for reliable enforcement by the Internet TTL mechanism.

The 16-bit window field of TCP limits the effective bandwidth B to 2^{16}/RTT, where RTT is the round-trip time in seconds. If the RTT is large enough, this limits B to a value that meets the constraint for a large MSL value. For example, consider a transcontinental backbone with an RTT of 60 ms. With the bandwidth×delay product limited to 64 KB by the TCP window size, B is then limited to 1.1 Mbytes/sec, no matter how fast the theoretical transfer rate of the path. This corresponds to cycling the sequence number space in *Twrap* = 2,000 secs, which is safe in today's Internet.

The issue to be addressed here is not the larger window but rather the higher bandwidth. For example, consider a (very large) FDDI LAN with a diameter of 10 km. Using the speed of light, we can compute the RTT across the ring as $(2\times10^4)/(3\times10^8) = 67$ microseconds, and the bandwidth×delay product is then 833 bytes. A TCP connection across this LAN using a window of only 833 bytes will run at the full 100 Mbps and can wrap the sequence space in about 3 minutes, very close to the MSL of TCP. Thus, high speed alone can cause a reliability problem with sequence number wrap-around, even without extended windows.

The IP mechanism for MSL enforcement is loosely defined and even more loosely implemented in the Internet. Therefore, it is unwise to depend upon active enforcement of MSL for TCP connections. It is unrealistic to imagine setting MSL's smaller than the current values (i.e., 120 seconds specified for TCP).

A possible solution for the problem of cycling the sequence space would be to increase the size of the TCP sequence number field. For example, the sequence number field (and also the acknowledgment field) could be expanded to 64 bits. This could be done either by changing the TCP header or by means of an additional option. But this affects the bandwidth efficiency that is important in a satellite environment.

To fix this problem, the Internet End-to-End Research Group devised a set of TCP options and algorithms to extend the sequence space. These changes were adopted by the Internet Engineering Task Force (IETF) and are now part of the TCP standard. The option is a timestamp option, which concatenates a timestamp to the 32-bit sequence number. Comparing timestamps using an algorithm called PAWS makes it possible to distinguish between two identical sequence numbers sent less than two minutes apart.

Depending on the actual granularity of the timestamp (the IETF recommends between 1-second and 1 millisecond), this extension is sufficient for link speeds of between 8 Gbps and 8 Tbps (terabits per second).

5.2.5. Slow Start

When a TCP connection starts up, the TCP specification requires the connection to be conservative and assume that the available bandwidth to the receiver is small. TCP is supposed to use an algorithm called "slow start" to probe the path to learn how much bandwidth is available. The slow start algorithm is quite simple and based on data sent per round trip. At the start, the sending TCP sends one segment (datagram) and waits for an acknowledgment. When it gets the acknowledgment, it sends two segments. Many TCP layers acknowledge every other segment they receive, so the slow start algorithm effectively sends 50 percent more data every round trip. It continues this process (sending 50 percent more data each round trip) until a segment is lost. This loss is interpreted as indicating congestion and the connection scales back to a more conservative approach for probing bandwidth for the rest of the connection.

There are two problems with the slow start algorithm on high-speed networks. First, the probing algorithm can take a long time to get up to speed. The time required to get up to speed is $RTT(1 + \log_{1.5}(D \times B/L))$ where: RTT is the round-trip time, D×B is the delay-bandwidth product and L is the average segment length. If we are trying to fill a pipe with a single TCP connection, then D×B should be the product of the bandwidth available to the connection and the round-trip time.

An important point is that as the bandwidth goes up or the round-trip time increases or both the startup time can be quite long. For instance, on a Gbps GEO satellite link with a 0.5 second round-trip time, it takes 29 round-trip times or 14.5 seconds to finish startup. If the link is otherwise idle, during that period most of the link bandwidth will not be used.

Even worse is that in many cases the entire transfer will be completed before the slow start algorithm has finished. The user will never experience the full link bandwidth. All the transfer time will be spent in slow start. Currently, IETF is in the early stages of considering a change to allow TCP connections to transmit more than one segment at the beginning of the initial slow start. If there is capacity in the path, this change will reduce the slow start by up to three round-trip times. This change mostly benefits shorter transfers that never get out of slow start.

The second problem is interpreting loss as indicating congestion. TCP has no easy way to distinguish losses due to transmission errors from losses due to congestion, so it makes the conservative assumption that all losses are due to congestion. Given the loss of a TCP segment early in the slow start process, TCP will then set its initial estimate of the available bandwidth too low. Since the probing algorithm becomes linear rather than exponential after the initial estimate is set, the time to get to full transmission rate can be very long. On a gigabit GEO link, it could be several hours. Therefore, there is a strong incentive to use separate protocols to identify congestion and transmission errors in a satellite environment.

5.2.6. Congestion Avoidance

Jacobson describes a congestion avoidance algorithm, which is similar to the slow start algorithm. Essentially, the sending TCP maintains a congestion window, an estimate of the actual available bandwidth of the path to the receiver. This estimate is set initially by the slow start at the start of the connection. Then, the estimate is varied up and down during the life of the connection based on indications of congestion. In general, congestion is indicated by loss of one or more datagrams.

The basic estimation algorithm is as follows. Every round trip, the sending TCP increases its estimate of the available bandwidth by one maximum-sized segment. Whenever the sender either finds a segment was lost or receives an indication from the network (e.g., an ICMP Source Quench) that congestion exists, the sender halves its estimate of the available bandwidth. The sender then resumes the one segment per round-trip probing algorithm. Like the slow start algorithm, the major issue with this algorithm is that over high-delay-bandwidth links a datagram lost to transmission error will trigger a low estimate of the available bandwidth, and the linear probing algorithm will take a longer time to recover.

Another issue is that the rate of improvement under congestion avoidance is a function of the delay-bandwidth product. Basically, congestion avoidance allows a sender to increase its window by one segment for every round-trip time's worth of data sent. In other words, congestion avoidance increases the transmission rate by $L/D \times B$ each round trip.

5.2.7. Selective Acknowledgments

Recently the IETF has approved an extension to TCP called Selective Acknowledgments (SACKs). SACKs make it possible for TCP to acknowledge data received out of order. SACKs have two major benefits. First, they improve the efficiency of TCP retransmissions by reducing the retransmission period. Historically, TCP has used a retransmission algorithm that emulates selective-repeat ARQ using the information provided by in-order acknowledgments. This algorithm works, but takes roughly one round-trip time per lost segment to recover. SACKs allow a TCP to retransmit multiple missing segments in a round trip. Second and more importantly, work by Mathis and Mahdavi has shown that with SACKs a TCP can better evaluate the available path bandwidth in a period of successive losses and avoid doing a slow start.

It is important to keep in mind that the various TCP mechanisms are interrelated, especially when applied to problems of high performance. If the sequence space and window size is not large enough, no improvement to congestion windows will help since TCP cannot go fast enough anyway. Also, if the receiver chooses a small window size, it takes precedence over the congestion window and can limit throughput. More broadly, tinkering with TCP algorithms tends to show odd interrelations. For instance, the individual TCP Vegas performance improvements were shown to work only when applied together. Applying only some of the changes actually degrades performance. There are known TCP anomalies where the congestion window gets misestimated, causing the estimation algorithm to briefly thrash before converging on a congestion window.

5.3. Satellites and Transport Layer Protocol Throughput

Significant advances have been made in area of satellite technology in terms of link speed and throughput. There is a lot of interest to support high speed traffic and Internet access using satellite technology. Therefore, it is important to understand the limitations of existing transport protocols when used in a satellite environment. Specifically, understanding the problem of achieving high throughput over satellite links is important. There is a need to implement the extensions to the TCP sequence space and window size and the relationship between slow start and performance over satellite links and some possible solutions. Currently, satellites offer a range of channel bandwidths from the very small to the very large (TDRS and ACTS satellites). They also have a range of delays, from relatively small delays of low earth orbit (LEO) satellites to the much larger delays of GEO satellites.

5.4. Performance Issues in Satellite Environment

Many of the issues related to TCP/IP performance were due to the presence of high-delay-bandwidth paths. All but the slowest satellite links are by definition high-delay-bandwidth paths because the transmission delays between the satellite and the earth's surface are large. Support of large windows and PAWS are essential for TCP to perform well at higher speeds and at speeds past about 100 Mbps.

In addition, the initial slow start period can be quite long and involve large quantities of data. Between 8 and 21 megabytes of data are sent over a satellite link during slow start at 155 Mbps. Even at 1.5 Mbps, a GEO link must carry nearly 200 KB before slow start ends. Therefore, satellite links look slow and inefficient for the average data transmission. Interestingly enough, long-distance terrestrial links will also look slow. Their delays are comparable to those of LEO links.

Furthermore, short data transfers will never achieve full link rate. In many cases, a gigabyte file transfer or larger is probably required to ensure throughput figures are not heavily influenced by slow start. Obviously some sort of solution to reduce the slow start transient would be desirable. One obvious solution is to dispense with slow start and just start sending as fast as one can until data is dropped, and then slow down. This approach is known to be disastrous. Indeed, slow start was invented in an environment in which TCP implementations behaved this way and were driving the Internet into congestion collapse. As one example of how this scheme goes wrong, consider a Gbps capable TCP launching several hundred megabits of data over a path that turns out to have only 9.6 Kbps of bandwidth. There is a significant bandwidth mismatch that cause segments to be discarded or suffer long queuing delays. One of the issues is that the sending TCP has no idea when it starts sending and how much bandwidth a particular transmission path has. In the absence of this knowledge, a TCP should be conservative. And slow start is conservative – it starts by sending just one segment in the first round trip.

In order to avoid TCP spending its time in slow start, TCP needs more information about the path. One nice aspect of this problem is that it is not specific to satellites. Terrestrial lines need a solution too. Therefore, a general solution that would work in both satellites and terrestrial environment is needed. If the transport protocol had more information about the path, it could presumably skip at least some of the slow start process by starting the slow start at a somewhat higher rate than one segment. But actually learning the properties of the path is hard. At present, IP does not keep path bandwidth information so TCP cannot ask the network about path properties. While there are ways to estimate path bandwidth dynamically (such as packet-pair), the estimates can easily be distorted in the presence of cross traffic.

Another idea for getting around slow start is a practice known as "TCP spoofing". The idea calls for a router near the satellite link to send back acknowledgments for the TP data to give the sender the illusion of a short delay path. The router then suppresses acknowledgments returning from the receiver, and takes responsibility for retransmitting any segments lost downstream of the router. There are a number of issues associated with this scheme. First, the router must do a considerable amount of work after it sends an acknowledgment. It must buffer the data segment because the original sender is now free to discard its copy. If the segment gets lost between the router and the receiver, the router has to take full responsibility for retransmitting it.

One side effect of this behavior is that if a queue builds up, it is likely to be a queue of TCP segments that the router is holding for possible retransmission. Unlike IP datagrams, this data cannot be deleted until the router gets the relevant acknowledgments from the receiver. Second, spoofing requires symmetric paths. The data and the acknowledgments must flow along the same path through the router. However in a satellite environment, asymmetric paths are quite common. Third, spoofing is vulnerable to unexpected failures. If a path changes or the router crashes, data may be lost. Data may even be lost after the sender has finished sending and, based on the router's acknowledgments, reported data successfully transferred. Fourth, it doesn't work if the data in the IP datagram is encrypted because the router will be unable to read the TCP header.

5.5. Error Rates for Satellite Paths

Experience suggests that satellite paths have higher error rates than terrestrial lines. In some cases, the error rates are as high as 1 in 10^{-5}. Higher error rates matter for two reasons. First, they cause errors in datagrams, which will have to be retransmitted. Second, as noted above TCP typically interprets loss as a sign of congestion and goes back into a modified version of slow start. Clearly there is a need to either reduce the error rate to a level acceptable to TCP or find a way to let TCP know that the datagram loss is due to transmission errors, not congestion.

What is an acceptable link error rate in a TCP/IP environment that will offer a certain level of throughput? There is no hard and fast answer to this problem. Partridge and Shepard suggest ways to find a relationship between TCP's natural frequency of congestion avoidance starts to the link error rate. Then, pick an error rate that is substantially less than congestion avoidance frequency.

As an alternative to or in conjunction with reducing satellite error rates, one might wish to make TCP more intelligent about handling transmission errors. There are basically two approaches:

either TCP can explicitly be told that link errors are occurring, or TCP can infer that link errors are occurring. The Satellite Communications Protocol Specification (SCPS) has looked at this technique as a possible approach. One general challenge in explicit notification is that TCP and IP rarely know that transmission errors have occurred because transmission layers discard the erred frames without passing them to TCP and IP.

Having TCP infer which errors are due to transmission error rather than congestion also presents challenges. One has to find a way for TCP to distinguish congestion from transmission errors reliably, using only information provided by TCP acknowledgments. The algorithm should never make a mistake because a failure to respond to congestion can exacerbate network congestion.

5.6. Closed Form Queuing Model For Transport Layer Latency

TCP provides a reliable transport service between two processes running on different hosts. An important characteristic of TCP is that it includes a congestion control mechanism. TCP congestion control prevents any one TCP connection from monopolizing the bandwidth in the path along which the connection's segments travel. The TCP protocol uses implicit feedback information from the network to regulate each connection's transmission rate.

TCP congestion control aims to give each connection traversing a congested link an equal share of the bandwidth. Consider a connection k, that passes through N links with link k having transmission rate CK and supporting a total of JK TCP connections. Suppose all the TCP connections are transporting big files and that there is no UDP traffic along the path of N links. If the bandwidth is equitably shared among the TCP connections on each link, then connection k gets allocated a rate of CK/JK on the k^{th} link. However, a connection's end-to-end average rate cannot exceed the minimum allocated bandwidth (i.e., the bottleneck bandwidth) along the path. Thus the "truly fair" end-to-end transmission rate for connection k is:

$$Ck = \min\{C1/J1,\ldots,CN/JN\}$$

The TCP protocol aims to give the connection k the end-to-end rate Ck. The way a TCP connection controls its transmission rate is by limiting the number of segments it sends in a round trip time (RTT). RTT is the elapsed time between when TCP sends a segment into the network until it receives an acknowledgement for the segment. The round-trip time includes end-to-end propagation delay, queuing delays and processing delays in the routers and end systems. Assuming that a connection transmits w segments of size M every RTT seconds, then the connection's transmission rate is (w x M) / RTT. TCP modifies its transmission rate (up or down) by simply modifying w (up or down). In general, a TCP connection starts with a small value of w and probes for spare bandwidth by increasing w. The TCP connection continues to increase w until a segment loss occurs (detected by a timeout or duplicate acknowledgements). This signals to the sender that there is very likely an overly congested link between sender and receiver. When a loss occurs, the TCP connection reduces w to a "safe level" and then probes again for more bandwidth by slowly increasing w.

5.6.1. Overview of TCP Congestion Control

The source and destination sides of a TCP connection consists of a receive buffer, a send buffer, and several variables (LastByteRead, RcvWin, etc.). In addition, congestion control mechanisms on each side of the connection keep track of two additional variables: the congestion window and the threshold. The congestion window (CongWin) imposes an additional constraint how much traffic a source can send into a connection. Specifically, the amount of unacknowledged traffic that a host can send into a connection oriented transport connection may not exceed the minimum of CongWin and RcvWin; i.e., (LastByteSent - LastByteAcked) ≤ min{CongWin, RcvWin}.

This threshold is a variable that controls how CongWin grows. To see how the congestion window evolves throughout the lifetime of a TCP connection, assume that the TCP receive buffer is large enough that the receive window can be ignored. Therefore, the amount of unacknowledged traffic that a source can send into a TCP connection is solely limited by CongWin. If a sender has a huge file to send to a destination once a TCP connection is established between the two end systems, the application process at the source writes bytes to the source's TCP send buffer. The transport layer takes a data segment of size M (e.g., 536 bytes), encapsulates each data segment within a TCP segment, and hands the segments into the network layer. The TCP congestion window regulates the times at which the segments are sent into the network.

Initially, the congestion window is equal to one M. Therefore, TCP sends the first segment into the network and waits for an acknowledgement. If this segment is acknowledged before its timeout, the sender increases the congestion window by one M and sends out two maximum-size segments. If these segments are acknowledged before their timeouts, the sender increases the congestion window by one M for each of the acknowledged segments, giving a congestion window of four M, and sends out four maximum-sized segments. This procedure continues as long as:

- The congestion window is below the threshold and
- Acknowledgements arrive before their corresponding timeouts.

During this phase of the congestion control procedure, the congestion window increases exponentially. (The congestion window is initialized to one M. After one RTT this window is increased to two segments. After three round-trip times the window is increased to four segments. After four round-trip times the window is increased to eight segments, etc.) This phase of the algorithm is called slow start because it begins with a small congestion window equal to one M. (The transmission rate of the connection starts slowly but has rapid acceleration.)

When a timeout occurs, the threshold is set to half the current congestion window and the congestion window is reset to one M. The source then increases the congestion window exponentially using the slow start procedure until the congestion window reaches the threshold. Once the congestion window reaches the threshold, the congestion window increases linearly rather than exponentially. In particular, TCP increases its congestion window by one each round-trip time. For example, if w is the current value of the congestion window and w is greater than the threshold, then each time an acknowledgement arrives TCP replaces w with w + the lower floor of 1/w. This increases the congestion window by one in each RTT for which an entire window's worth of acknowledgements arrives. This phase of the algorithm is called congestion avoidance. The congestion avoidance phase continues as long as the acknowledgements arrive before their corresponding timeouts.

Therefore, when the congestion window is below the threshold, the congestion window grows exponentially. When the congestion window is above the threshold, the congestion window grows linearly. Whenever there is a timeout, the threshold is set to one half of the current congestion window and the congestion window is then set to one. This congestion control algorithm is due to V. Jacobson.

The TCP congestion control algorithm just described is often referred to as Tahoe. One problem with the Tahoe algorithm is that when a segment is lost the sender side of the application may have to wait a long period of time for the timeout. For this reason, a variant of Tahoe, called Reno, is implemented by most operating systems. Like Tahoe, Reno sets its congestion window to one segment upon the expiration of a timer. However, Reno also includes a mechanism that sometimes triggers the retransmission of a dropped segment before the occurrence of the corresponding timeout. This mechanism called fast retransmission, requires a small change in the sender side of the Tahoe algorithm and no change at the receiver side.

To see how fast retransmission works, consider sending a file from host A to host B over a TCP connection. Recall that every time a segment arrives at host B, host B issues an acknowledgement, even if host B has already acknowledged the segment. Consider one of the segments in the middle of the file. If the segment is either dropped or arrives out of order, host B sends a duplicate acknowledgement for the preceding segment. If host A receives several identical accumulative acknowledgements for the preceding segment, then host A starts to suspect that the segment was dropped. The fast retransmission heuristic assumes that the packet was dropped when it receives the same accumulative acknowledgement three times, that is, when it receives a triple acknowledgement. When a triple acknowledgement occurs, host A resends the segment, even if the timer for the segment has not yet expired. Reno also employs a fast recovery mechanism, which essentially cancels the slow start phase after a fast retransmission.

Most TCP implementations currently use the Reno algorithm. There is another algorithm in the literature (the Vegas algorithm) which can improve Reno's performance. Whereas Tahoe and Reno react to congestion (i.e., to overflowing router buffers), Vegas attempts to avoid congestion while maintaining good throughput. The basic idea of Vegas is to

- Detect congestion in the routers between source and destination before packet loss occurs, and

- Lower the rate linearly when this imminent packet loss is detected.

The imminent packet loss is predicted by observing the round-trip times -- the longer the round-trip times of the packets, the greater the congestion in the routers. Vegas is not part of the most popular TCP implementations.

5.6.2. TCP and Fairness

TCP congestion control strives to give all the ongoing client-server applications sharing a network link an equal share of the bandwidth. For example, suppose there is a bottleneck link of rate C bps that supports 10 ongoing applications. Ideally, each of the 10 ongoing applications would get an average throughput of C/10 bps. If all of the applications were to run over TCP and if every application were to use just one TCP connection, then TCP congestion control would nearly achieve the goal of fairness. But in practice, these two conditions are typically not met and client-server applications obtain very unequal portions of link bandwidth.

Many network applications run over TCP rather than UDP because they want to make use of TCP's reliable transport service. But when an application chooses TCP, it not only gets reliable data transfer but also TCP congestion control. TCP congestion control regulates an application's transmission rate. But many multimedia applications do not want their transmission rate to be throttled, even if the network is very congested. Therefore, these applications typically run over UDP. Audio and video applications prefer to send to the network at a constant rate and occasionally lose packets, rather than reduce their rates to "fair" levels at times of congestion and not lose any packets. From the perspective of TCP, the multimedia applications running over UDP are not being fair -- they do not cooperate with the other connections and adjust their transmission rates appropriately. A major challenge in the upcoming years will be to develop congestion control mechanisms that allow the resources to be shared fairly.

Even if one could force UDP traffic to behave fairly, the fairness problem would still not be completely solved. This is because there is no mechanism available to prevent an application running over TCP from using multiple parallel connections.

5.6.3. Dynamic Performance of TCP

Consider sending a large file over a TCP connection. In such an environment the connection is in the slow-start phase for a relatively short period of time because the connection grows out of the phase exponentially fast. Therefore, the slow-start phase can be ignored and the congestion window grows linearly, gets reduced in half when loss occurs, grows linearly, gets reduced in half when loss occurs, etc. This gives rise to the well-documented saw-tooth behavior of TCP.

Given the saw-tooth behavior, how is the average throughput of a TCP connection calculated? During a particular round-trip interval, the rate at which TCP sends data is a function of the congestion window and the current RTT: when the window size is $w \times M$ and the current round-trip time is RTT, then:

$$\text{TCP connections transmission rate} = (w \times M)/\text{RTT}.$$

During the congestion avoidance phase, TCP probes for additional bandwidth by increasing w by one each RTT until loss occurs. Denote by W the value of w at which loss occurs. Assuming that the RTT and W are approximately constant over the duration of the connection, then:

$$(W \times M)/(2 \times RTT) \leq \text{TCP transmission rate} \leq (W \times M)/RTT$$

The steady-state behavior of a TCP connection can be modeled with the above assumptions. The network drops a packet from the connection when the connection's window size increases to $W \times M$. The congestion window is then cut in half and then increases by one M per round-trip time until it reaches W packets again. This process repeats itself over and over again. In this model, the amount of bandwidth available to a connection varies from $(W \times M)/(2 \times RTT)$ to $(W \times M)/RTT$ because the available bandwidth increases linearly between the two extreme values. Therefore:

$$\text{Average throughput of a connection} = (0.75 \times W \times M)/RTT.$$

Using this highly idealized model for the steady-state dynamics of TCP, an interesting expression can be obtained that relates a connection's loss rate to its available bandwidth.

5.6.4. Modeling the Latency due to Connection Establishment and TCP Slow Start

Many TCP connections transport relatively small files from one host to another. When transporting a small file, TCP connection establishment and slow start may have a significant impact on the latency. This section presents an analytical model that quantifies the impact of connection establishment and slow start on latency. For a given message, latency is defined as the time from when the client initiates a TCP connection until when the client receives the requested message.

The analysis presented here assumes that that the network is not in the congested state; i.e., the TCP connection transporting the message does not have to share link bandwidth with other TCP or UDP traffic. Also, in order not to obscure the central issues, the analysis is performed in the context of the simple one-link network. This model closely fits the selected protocol architecture. In order to get a closed form solution to latency problem the following simplifying assumptions are made:

- The amount of data that the sender can transmit is solely limited by the sender's congestion window. (Thus, the TCP receive buffers are large.)
- Packets are neither lost nor corrupted, so that there are no retransmissions.
- All protocol header overheads (including TCP, IP and link-layer headers) are negligible and ignored.

- The message to be transferred consists of an integer number of segments of size M (maximum segment size).

- The only packets that have non-negligible transmission times are packets that carry maximum-size TCP segments. Request packets, acknowledgements and TCP connection establishment packets are small and have negligible transmission times.

- The initial threshold value for congestion control mechanism is a large number that is never reached by the congestion window.

The following notation is used in obtaining the relationship for the latency. Let:

- S be the size of the message to be transferred in bits.
- M be maximum size segment in bits.
- C be the transmission rate of the link from the server to the client in bps.
- RTT be the round-trip time in seconds.

where, RTT is defined as the time elapsed for a small packet to travel from client to server and then back to the client, excluding the transmission time of the packet. It includes the two end-to-end propagation delays between the two end systems and the processing times at the two end systems. In addition, it is assumed that RTT is also equal to the roundtrip time of a packet beginning at the server.

Although the analysis presented in this section assumes an uncongested network with a single link, it can be extended to the more realistic case of multi-link congested network. For a congested network, C roughly represents the amount of bandwidth received in steady state in the end-to-end network connection, and RTT represents a round-trip delay that includes queuing delays at the routers preceding the congested links. In the congested network case, the TCP connection is modeled as a constant-bit-rate connection of rate C bps preceded by a single slow-start phase. (This is roughly how TCP Tahoe behaves when losses are detected with triplicate acknowledgements).

What would be the latency if there was no congestion window constraint; i.e., if the server were permitted to send segments back-to-back until the entire message is sent? To answer this question, first note that one RTT is required to initiate the TCP connection. After one RTT the client sends a request for the message (which is piggybacked onto the third segment in the three-way TCP handshake). After a total of two RTTs, the client begins to receive data from the server. The client receives data from the server for a period of time S/C, the time for the server to transmit the entire message. Thus, in the case of no congestion window constraint, the total latency is $2 \times \text{RTT} + S/C$. This represents a lower bound. The slow start procedure with its dynamic congestion window will of course elongate this latency.

5.6.5. Latency with Static Congestion Window

Although TCP uses a dynamic congestion window, an analysis is performed for the case of a static congestion window. Let W (a positive integer) denote a fixed-size static congestion window. For the static congestion window, the server is not permitted to have more than W unacknowledged outstanding segments. When the server receives a request from the client, the server immediately sends W segments back-to-back to the client. The server then sends one segment into the network for each acknowledgement it receives from the client. The server continues to send one segment for each acknowledgement until all of the segments of the message have been sent. There are two cases to consider:

- $W \times M/C > RTT + M/C$. In this case, the server receives an acknowledgement for the first segment in the first window before the server completes the transmission of the first window.

- $W \times M/C < RTT + M/C$. In this case, the server transmits the first window's worth of segments before the server receives an acknowledgement for the first segment in the window.

For the case where ($W \times M/C > RTT + M/C$), one RTT is required to initiate the TCP connection. After one RTT, the client sends a request for the message (which is piggybacked onto the third segment in the three-way TCP handshake). After two RTTs, the client begins to receive data from the server. Segments arrive periodically from the server every M/C seconds, and the client acknowledges every segment it receives from the server. Because the server receives the first acknowledgement before it completes sending a window's worth of segments, the server continues to transmit segments after having transmitted the first window's worth of segments. And because the acknowledgements arrive periodically at the server every M/C seconds from the time when the first acknowledgement arrives, the server transmits segments continuously until it has transmitted the entire message. Thus, once the server starts to transmit the message at rate C, it continues to transmit the message at rate C until the entire message is transmitted. The latency therefore is ($2 \times RTT + S/C$).

For the case where ($W \times M/C < RTT + M/C$), after a total of two RTTs the client begins to receive segments from the server. These segments arrive periodically every M/C seconds, and the client acknowledges every segment it receives from the server. But now, the server completes the transmission of the first window before the first acknowledgment arrives from the client. Therefore, after sending a window, the server must stall and wait for an acknowledgement before resuming transmission. When an acknowledgement finally arrives, the server sends a new segment to the client. Once the first acknowledgement arrives, a window's worth of acknowledgements arrive, with each successive acknowledgement spaced by M/C seconds. For each of these acknowledgements, the server sends exactly one segment. Thus, the server alternates between two states: a transmitting state, during which it transmits W segments; and a stalled state, during which it transmits nothing and waits for an acknowledgement. The latency is equal to $2 \times RTT$ plus the time required for the server to transmit the message, S/C, plus the amount of time that the server is in the stalled state.

To determine the amount of time the server is in the stalled state, let $K = S/W \times M$; where K is rounded up to the nearest integer. K represents the number of windows of data in the message of size S. The server is in the stalled state between the transmission of each of the windows; i.e., for K-1 periods of time, with each period lasting RTT- (W-1) M/C. Thus, for Case 2:

$$\text{Latency} = 2 \times RTT + S/C + (K-1) \times [M/C + RTT - W \times M/C]$$

Combining the two cases, we obtain a closed form solution for latency.

$$\text{Latency} = 2 \times RTT + S/C + (K-1) \times [M/C + RTT - W \times M/C]^{+}$$

where $[x]^{+} = \max(x, 0)$

5.6.6. Latency with Dynamic Congestion Window

The analysis for dynamic windows is more complicated, but parallels the analysis for static windows. The goal is to obtain a closed form solution for latency when transferring messages under dynamic congestion window. The server first starts with a congestion window of one segment and sends one segment to the client. When it receives an acknowledgement for the segment, it increases its congestion window to two segments and sends two segments to the client (spaced apart by M/C seconds). As it receives the acknowledgements for the two segments, it increases the congestion window to four segments and sends four segments to the client (again spaced apart by M/C seconds). The process continues, with the congestion window doubling every RTT.

The ratio S/M is the number of segments in the message. The first window contains 1 segment; the second window contains 2 segments; and the third window contains 4 segments. More generally, the kth window contains 2^{k-1} segments. Let K be the number of windows that cover the message. In general K can be expressed in terms of S/M as follows:

$$\begin{aligned} K &= \text{minimum}\{k: 2^{0} + 2^{1} + \ldots + 2^{k-1} \geq S/M\} \\ &= \text{minimum}\{k : 2^{k} - 1 \geq S/M\} \\ &= \text{minimum}\{k : k \geq \log_2 (S/M + 1)\} \\ &= \text{ceiling}\{\log_2 (S/M + 1)\} \end{aligned}$$

After transmitting a window's worth of data, the server may stall while it waits for an acknowledgement. The amount of stall time after transmitting the kth window is the time from when the server begins to transmit the kth window until when the server receives an acknowledgement for the first segment in the window. This is (M/C + RTT). The transmission time of the kth window is $(M/C) \times 2^{k-1}$. The stall time is the difference of these two quantities and is given by:

$$[M/C + RTT - (2^{k-1}) \times (M/C)]^{+}.$$

The server can potentially stall after the transmission of each of the first (K-1) windows. (The server is done after the transmission of the Kth window.) The latency for transferring the message size S has three components: 2 × RTT for setting up the TCP connection and requesting the message; S/C, the transmission time of the message; and the sum of all the stalled times. Thus,

$$\text{Latency} = 2 \times RTT + S/C + \sum_{k=1}^{K-1} (M/C + RTT - 2^{k-1} \times M/C)$$

Comparing the above equation with the latency equation for static congestion windows, all the terms are the same except the term (W × M/C) for static windows is replaced by (2^{k-1} × M/C) for dynamic windows. Let Q be the number of times the server would stall if the message contained an infinite number of segments. Then,

$$\begin{aligned} Q &= \text{maximum } \{k : RTT + M/C - M/C \times 2^{k-1} \geq 0\} \\ &= \text{maximum } \{ k : 2^{k-1} \leq 1 + RTT/(M/C)\} \\ &= \text{maximum } \{ k : k \leq \log_2 (1 + RTT/(M/C) + 1\} \\ &= \text{floor } \{ [\log_2 (1 + RTT/(M/C)] + 1\} \end{aligned}$$

The actual number of times the server stalls is P = minimum {Q, K-1}. Combining the above two equations gives:

$$\text{Latency} = S/C + 2 \times RTT + \sum_{k=1}^{P} (RTT + M/C - M/C \times 2^{k-1})$$

And the sum of k=1 to P of $2^{k-1} = 2^P - 1$. Combining the above two equations gives the following closed-form expression for the latency:

$$\text{Latency} = 2 \times RTT + S/C + P \times (RTT + M/C) - (2^P - 1) \times M/C$$

Thus to calculate the latency, calculate K and Q, set P = minimum {Q, K-1}, and use P in the above formula. It is interesting to compare the TCP latency to the latency that would occur if there were no congestion control (i.e., no congestion window constraint). Without congestion control, the latency is (2 × RTT + S/C), which is defined to be the Minimum Latency.

From the above formula, TCP slow start will not significantly increase latency if RTT < S/C; i.e., if the round-trip time is much less than the transmission time of the message. If relatively large messages are sent over an uncongested, high-speed link, then slow start has an insignificant effect on latency.

The effect of slow start on latency is explored by looking at a number of scenarios. In all the example scenarios M (the segment size) is set to 536 bytes, a common default value for TCP. For a round trip time, RTT of 100 msec and a message of size S = 100 Kbytes, the number of windows that cover this message is K=8. Table 3 examines the effect of the slow-start mechanism on the latency for various transmission speeds. Figure 3 shows a graphical display of the data in Table 3.

Table 3. Effect of Slow Start On Latency for RTT = 100 msec and S = 100 Kbytes

Line Speed in Mbps	Minimum Latency	Latency with Slow Start
1.5	0.733	1.170
5	0.36	0.957
10	0.28	0.928
16	0.25	0.917
25	0.232	0.911
45	0.217	0.906
50	0.216	0.905
100	0.208	0.902
155	0.205	0.901

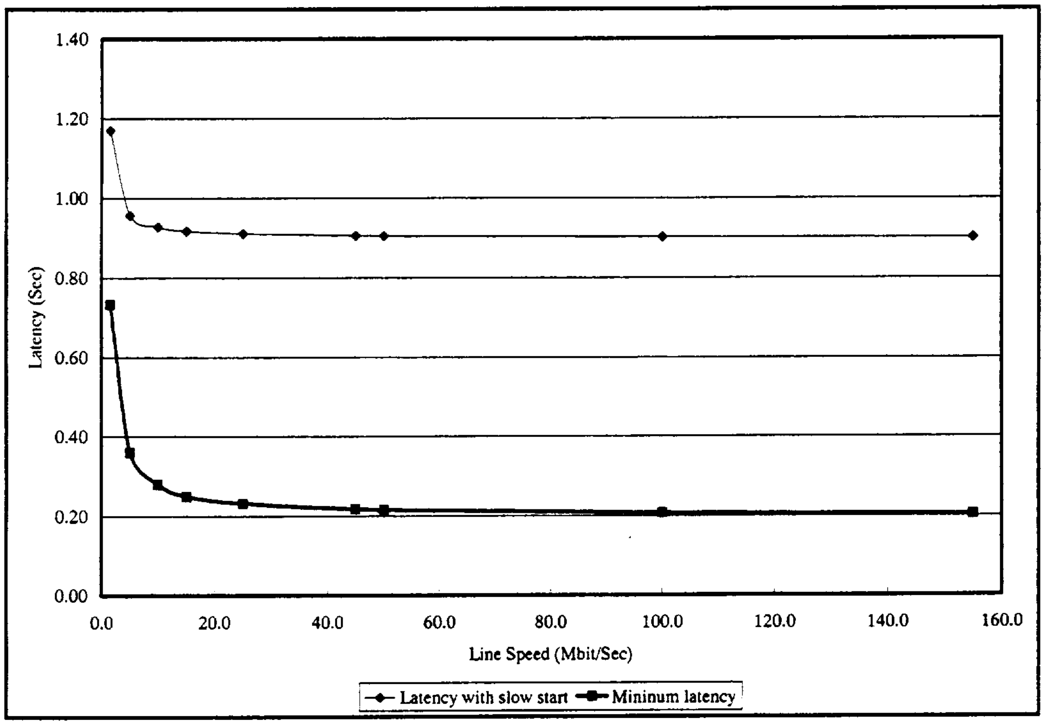

Figure 3. Effect of Slow Start On Latency for RTT = 100 msec and S = 100 Kbytes

Table 3 and Figure 3 show that for a large message, slow-start adds appreciable delay only when the transmission rate is high. If the transmission rate is low, then acknowledgments come back relatively quickly, and TCP quickly ramps up to its maximum rate. When R = 10 Mbps, the server stalls between each window, which causes a significant increase in the delay.

Now consider sending a small message of size S = 5 Kbytes. The number of windows that cover this message is K= 4. For various transmission rates, Table 4 presents the affect of the slow-start mechanism. Figure 4 shows a graphical display of the data in Table 4.

Table 4. Effect of Slow Start on Latency for RTT = 100 msec and S = 5 Kbytes

Line Speed in Mbps	Minimum Latency	Latency with Slow Start
5	0.28	0.504
10	0.204	0.502
16	0.205	0.501
25	0.201	0.50
45	0.20	0.50
50	0.20	0.50
100	0.20	0.50
155	0.20	0.50

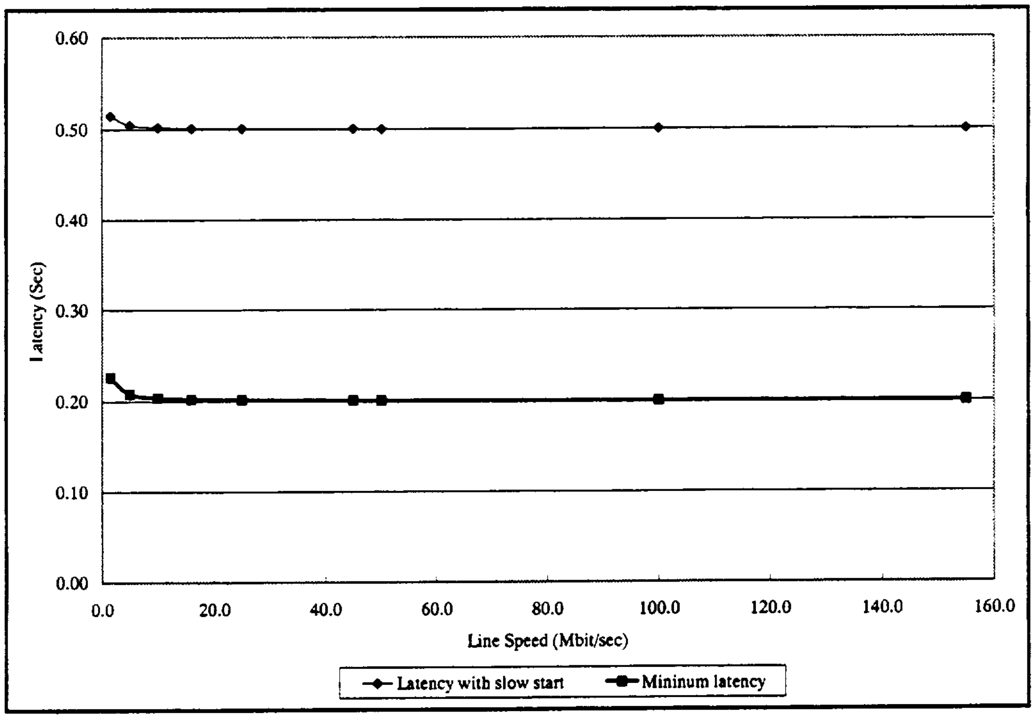

Figure 4. Effect of Slow Start on Latency for RTT = 100 msec and S = 5 Kbytes

Once again, slow start adds an appreciable delay when the transmission rate is high. For a larger RTT, the effect of slow start becomes significant for small message and smaller transmission rates. Table 5 shows the effect of slow start for RTT = 1 second and S = 5 Kbytes (K = 4). Figure 5 shows a graphical display of the data in Table 5.

Table 5. Effect of Slow Start For RTT = 1 sec and S = 5 Kbytes

Line Speed in Mbps	Minimum Latency	Latency with Slow Start
1.5	2.026	5.015
5	2.008	5.004
10	2.004	5.002
16	2.002	5.00
25	2.001	5.0
45	2.00	5.0
50	2.00	5.0
100	2.00	5.0
155	2.00	5.0

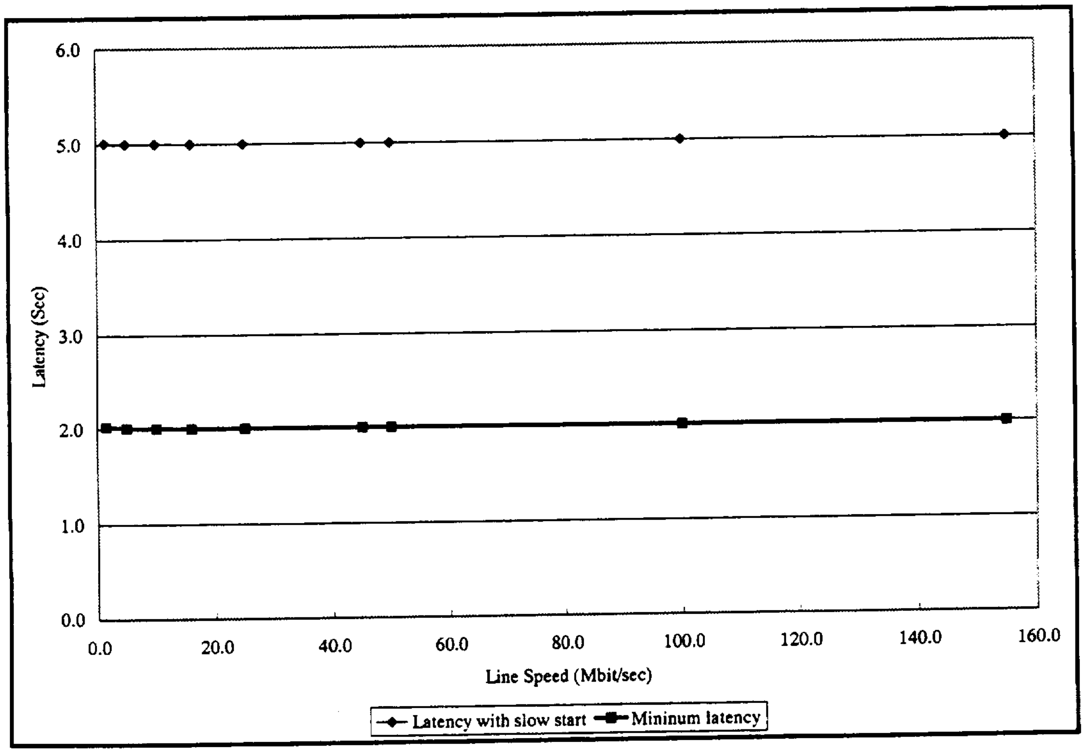

Figure 5. Effect of Slow Start For RTT = 1 sec and S = 5 Kbytes

In summary, slow start can significantly increase latency when the message size is relatively small and the RTT is relatively large.

NETWORK LAYER ISSUES

The main function of the network layer, and those below, is to provide a data transfer capability across the communications subnetwork. The functionality provided is specific to the subnetwork and must be implemented by the end systems as well as intermediate systems in the network. Intermediate systems in the subnetwork take the responsibility of routing and relaying information. The network layer shields the transport layer from all concerns about the subnetwork.

The routing function routes packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to reach the destination. The notable exception is broadcast type networks. Even here routing is an issue if the source and destination are not on the same network. The algorithm that chooses the routes and the data structures that they use is a major area of network layer design.

5.7. IP Throughput Issues

IP (Internet Protocol) is the network layer protocol in the TCP/IP protocol suite. IP's function is to provide a protocol to integrate heterogeneous networks together. In brief, a media-specific way to encapsulate IP datagrams is defined for each media (e.g., satellite, Ethernet, or Asynchronous Transfer Mode). Devices called "routers" move IP datagrams between the different media and their encapsulations. Routers pass IP datagrams between different media according to routing information in the IP datagram. This mesh of different media interconnected by routers forms an IP "Internet" in which all hosts on the integrated mesh can communicate with each other using IP.

The actual service IP implements is unreliable datagram delivery. IP simply promises to make a reasonable effort to deliver every datagram to its destination. However, IP is free to occasionally lose datagrams. Since IP provides such a simple service, one might assume that IP places no limits on throughput. Broadly speaking, this assumption is correct. IP places no constraints on how fast a system can generate or receive datagrams. A system transmits IP datagrams as fast as it can generate them. However, IP does have two features that can affect throughput: the IP Time to Live and IP Fragmentation.

IP Time to Live. In certain situations IP datagrams may loop among a set of routers. These loops are transient (a datagram may loop for a while and then proceed to its destination) or long-lived. To protect against datagrams circulating semi-permanently, IP places a limit on how long a datagram may live in the network. The limits are imposed by a Time to Live (TTL) field in the IP datagram. The field is decremented at least once at every router the datagram encounters. When the TTL reaches zero, the datagram is discarded.

Originally, the IP specification required that the TTL also be decremented at least once per second. Since the TTL field is 8-bits wide, this means a datagram could live for approximately

4.25 minutes. In practice, the injunction to decrement the TTL once a second is ignored. Perversely, specifications for higher layer protocols like TCP usually assume that the maximum time a datagram can live in the network is only two minutes.

The significance of the maximum datagram lifetime is that it means higher layer protocols must be careful not to send two similar datagrams (in particular, two datagrams which could be confused for each other) within a few minutes of each other. This limitation is particularly important for sequence numbers. If a higher layer protocol numbers its datagrams, it must ensure that it does not generate two datagrams with the same sequence number within a few minutes of each other. Otherwise, IP might deliver the second datagram first and confuse the receiver.

IP Fragmentation. Different network media have different limits on the maximum datagram size. This limit is typically referred to as the Maximum Transmission Unit (MTU). When a router is moving a datagram from one media to another, it may discover that the datagram, which was of legal size on the inbound media, is too big for the outbound media. To get around this problem, IP supports fragmentation and reassembly, in which a router can break the datagram up into smaller datagrams to fit on the outbound media. The smaller datagrams are reassembled into the original larger datagram at the destination (not the intermediate hops).

Fragments are identified using a fragment-offset field that indicates the offset of the fragment from the start of the original datagram. Datagrams are uniquely identified by their source, destination, higher layer protocol type, and a 16-bit IP identifier, which must be unique when combined with the source, destination and protocol type.

Observe that there is a clear link between the TTL field and the IP identifier. An IP source must ensure that it does not send two datagrams with the same IP identifier to the same destination, using the same protocol within a maximum datagram lifetime. Otherwise, fragments of two different datagrams may be incorrectly combined. If the maximum datagram lifetime is two minutes, we are limited to a transmission rate of only 546 datagrams per second because the IP identifier is only 16 bits long. That is clearly not fast enough. The maximum IP datagram size is 64 KB, so 546 datagrams is (at best) a bit less than 300 Mbps.

The problem of worrying about IP identifier consumption has largely been solved by the development of MTU Discovery – a technique for IP sources to discover the MTU of the path to a destination. MTU Discovery is a mechanism that allows hosts to determine the MTU of a path reliably. The existence of MTU discovery allows hosts to set the Don't Fragment (DF) bit in the IP header to prohibit fragmentation because the hosts will learn through MTU discovery if their datagrams are too big. Sources that set the DF bit need not worry about the possibility of having two identifiers active at the same time. Systems that do not implement MTU discovery (and thus cannot set the DF bit) need to be careful about this problem.

5.8. Required Error Performance of the Subnetwork Service

The required error performance of the subnetwork service depends on the performance requirements of the applications to be supported. The network layer, which is responsible for routing packets to the correct destination, is affected by the errors in the packet header. Therefore performance is affected by the probability that the data in the network layer header has been corrupted by bit-errors. Figure 6 shows the relationship between network layer header length, bit-error rate, and the probability that the network layer header is corrupted.

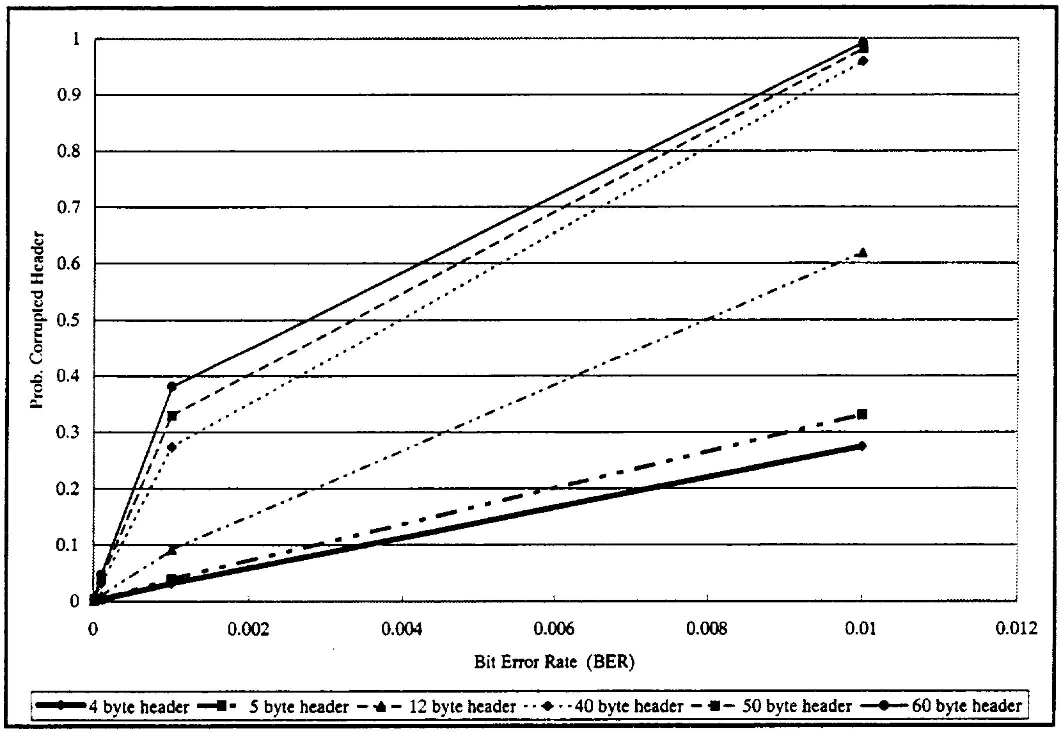

Figure 6. Probability of Corrupted Header

5.9. Probability of Corrupted Header

Equation 1 presents the probability that a network layer packet header is corrupted. This probability is based on an assumption that errors are distributed according to a Bernoulli process, and may, therefore, be affected if link-layer error correction techniques are applied to achieve the required bit-error rate. Figure 6 can be used to determine the error probability. Figure 6 allows one to determine whether the error characteristics exhibited by the data link layer are adequate to meet the applications requirements regarding the probability of misrouted data. The results can be used to determine the appropriate network layer header length. In this equation, BER is the bit-error rate of the link, and N is the number of octets in the network layer packet header.

$$P\text{ (Network layer packet header is corrupted)} = 1 - (1 - \text{BER})^{8 \times \text{N}} \quad \text{.............................(1)}$$

5.10. Probability of Network Protocol Data Unit (NPDU) Being Received Correctly

The service provided by the network layer in the protocol architecture is an unreliable one, meaning that the network protocol does not provide acknowledgment and retransmission capabilities. The network layer can provide various routing options such as flood routing, which can increase the probability of a datagram being received without retransmission. However, this is at the cost of greatly increased traffic throughout the network. The increase in traffic and the improvement in reliability depend on the network topology.

The equation used to generate the probability of a packet being received without an error is identical to equation 1, with N set to the entire length of the packet rather than just to the length of the network header.

5.11. Routing

The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to reach the destination. The notable exception is broadcast type networks. Even here routing is an issue if the source and destination are not on the same network. The algorithm that chooses the routes and the data structures that they use is a major area of network layer design.

The routing algorithm is that part of the network layer software responsible for deciding on which output line an incoming packet should be transmitted. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is set up. Thereafter, data packets just follow the previously established route. The latter case is sometimes called session routing because a route remains in force for an entire user session.

5.11.1. Routing Requirements

Regardless of whether routes are chosen independently for each packet or only when new connections are established, there are certain properties that are desirable in a routing algorithm: correctness, simplicity, robustness, stability, fairness, and optimality. Correctness and simplicity hardly require comment, but the need for robustness may be less obvious. Once a network becomes operational, it may be expected to run continuously for years without system wide failures. During that period there will be hardware and software failures of all kinds. End systems, routers, and lines will go up and down repeatedly, and the topology will change many times. The routing algorithm should be able to cope with changes in the topology and traffic without requiring all applications in the end systems to be aborted and the network to be restarted whenever some router goes down.

Stability is also an important goal for the routing algorithm. There exist routing algorithms that never converge to equilibrium, no matter how long they run. Fairness and optimality may sound obvious – surely no one would oppose them – but as it turns out, they are often contradictory goals. Evidently, some compromise between global efficiency and fairness to individual connection is needed.

Before one can even attempt to find trade-off between fairness and optimality, one must decide what it is to be optimized. Minimizing mean packet delay is an obvious candidate, but so is maximizing total network throughput. Furthermore, these two goals are also in conflict, since operating any queuing system near capacity implies a long queuing delay. As a compromise, many networks attempt to minimize the number of hops a packet must make. Reducing the number of hops tends to improve the delay and also reduce the amount of bandwidth consumed, which tends to improve the throughput as well.

5.11.2. Routing Techniques

Routing algorithms can be grouped into two major classes: non-adaptive and adaptive. Non-adaptive routing algorithms do not base their routing decisions on measurements or estimates of the current traffic and topology. Instead, the choice of the route to use to get from source to destination (for all sources and destinations) is computed in advance, off-line, and downloaded to the routers when the network is initialized. This procedure is sometimes called static routing.

Adaptive routing algorithms, in contrast, change their routing decisions to reflect changes in the topology and usually the traffic as well. Adaptive routing algorithms differ in where they get their information (i.e., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every delta t sec, when the load changes or when the topology changes), and what metric is used for optimization (i.e., distance, number of hops, or estimated transit time).

5.11.3. Shortest Path Routing

Shortest path routing algorithms are widely used because they are simple and easy to understand. The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line. To choose a route between a given pair of routers, the algorithm finds the shortest path between them using the graph representation.

One way of measuring path length is the number of hops. Another metric is the geographic distance in kilometers. However, many other metrics are also possible besides hops and physical distance. For example, each arc could be labeled with the mean queuing and transmission delay for some standard test packet as determined by hourly test runs. With this graph labeling, the shortest path is the fastest path, rather than the path with the fewest arcs or kilometers. In the most general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, and other factors. By changing the weighting function, the algorithm would then compute the "shortest" path measured according to any one of a number of criteria, or a combination of criteria.

There are a number of algorithms for computing the shortest path between two nodes of a graph. But the one due to Dijkstra is widely used. Each node is labeled with its distance from the source node along the best-known path. Initially, no paths are known so all nodes are labeled with infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be either tentative or permanent. Initially all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

5.11.4. Flooding

Another static algorithm is flooding, in which every incoming packet is sent out on every outgoing line except the one it arrived on. Flooding obviously generates vast numbers of duplicate packets. In fact, an infinite number is generated unless some measures are taken to dampen the process. One such measure is to have a hop counter contained in the header of each packet. The hop counter is decremented at each hop, with the packet being discarded when the counter reaches zero. Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely the full diameter of the subnet.

An alternative technique for damping the flood is to keep track of which packets have been flooded and avoid sending them out a second time. One way to achieve this goal is to have the source router put a sequence number in each packet it receivers from its hosts. Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen. If an incoming packet is on the list, it is not flooded. To prevent the list from growing without bound, each list should be augmented by a counter k, meaning that all sequence numbers through k have been seen. When a packet comes in, it is easy to check if the packet is a duplicate. If so, it is discarded. Furthermore, the full list below k is not needed, since k effectively summarizes it.

5.11.5. Selective flooding

A variation of flooding that is slightly more practical is selective flooding. In this algorithm the routers do not send every incoming packet out on every line, but only on those lines that are going approximately in the right direction. Flooding is not practical in most applications, but it does have some uses. For example, in military applications where large numbers of routers may not be in service at any instant, the tremendous robustness of flooding is highly desirable. In distributed database applications, it is sometimes necessary to update all the databases concurrently; in which case flooding can be useful. A third possible use of flooding is as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path because it chooses every possibly path in parallel. Consequently, no other algorithm can produce a shorter delay. But this has to be traded off against the overhead generated by the flooding process itself.

5.11.6. Flow-Based Routing

The routing algorithms presented so far take only the topology into account. They do not consider the load. In this section a static algorithm that uses both topology and load for routing are presented. It is called flow-based routing. In some networks, the mean data flow between each pair of nodes is relatively stable and predictable. For example, in a corporate network messages follow a predefined pattern so that the total volume of traffic varies little from day to day. Under conditions in which the average traffic from source to destination is known in advance and to a reasonable approximation constant in time, it is possible to analyze the flows mathematically to optimize the routing.

The basic idea behind the analysis is that for a given line, if the capacity and average flow are known, it is possible to compute the mean packet delay on that line from queuing theory. From the mean delays on all the lines, it is straightforward to calculate a flow-weighted average to get the mean packet delay for the whole subnetwork. The routing problem then reduces to finding the routing algorithm that produces the minimum average delay for the subnetwork.

To use this technique, certain information must be known in advance. First, the subnetwork topology must be known. Second, the traffic matrix must be given. Third, the line capacity matrix specifying the capacity of each line must be available. Finally, a possible routing algorithm must be chosen.

To evaluate a different routing algorithm, the entire process is repeated with different flows to get a new average delay. If one restricts oneself to only single path algorithms, there are only a finite number of ways to route packets from each source to each destination. It is always possible to write a program to simply try them all, one after another, and find out which one has the smallest mean delay. Since this calculation can be done off-line in advance, the fact that it may be time consuming is not necessarily a serious problem. This one is then the best routing algorithm.

5.11.7. Distance Vector Routing

Modern computer networks generally use dynamic routing rather than the static ones described above. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular.

Distance vector routing algorithms operate by having each router maintain a table giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with its neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it. It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by and containing one entry for each router in the subnetwork. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar. The router is assumed to know the "distance" to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special echo packets that the receiver timestamps and sends back as fast as it can.

As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every T msec each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor X, with X_i being X's estimate of how long it takes to get to router i. If the router knows that the delay to X is msec, it also knows that it can reach router i via X in $X_i {}^{+}$ m msec via X. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.

5.11.8. Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 Kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 Kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed. The algorithm often took too long to converge, even with tricks like split horizon. For these reasons, an entirely new algorithm called link state routing replaced it. Variants of link state routing are now widely used. The concept behind link state routing is simple and consists of five steps. Each router must:

- Discover its neighbors and learn their network addresses.
- Measure the delay or cost to each of its neighbors.
- Construct a package telling all it has just learned.
- Send this packet to all other routers.
- Compute the shortest path to every other router.

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then, Dijkstra's algorithm can be used to find the shortest path to every other router. The last three steps are presented below.

5.11.9. Building Link State Packets

Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age, and a list of neighbors. For each neighbor, the delay to that neighbor is given. Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically at regular intervals. Another possibility is to build them when some significant event occurs, such as a line or neighbor going down or coming back up again, or changing its properties appreciably.

5.11.10. Distributing the Link State Packets

The hardest part of the algorithm is distributing the link state packets reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, different routes may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

First, the basic distribution algorithm is presented. The fundamental idea is to use flooding to distribute the link state packets. To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see. When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded. If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete.

This algorithm has a few problems, but they are manageable. First, if the sequence numbers wrap around, confusion will reign. The solution here is to use a 32-bit sequence number. Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet will be rejected as a duplicate. Third, if a sequence number is ever corrupted, packets up to the current sequence number are rejected.

The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded. Normally, a new packet comes in, say, ever 10 minutes, so router information only times out when a router is down. The age field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time.

Some refinements to this algorithm make it more robust. When a link state packet comes in to a router for flooding, it is not queued for transmission immediately. Instead it is put in a holding area to wait a short while first. If another link state packet from the same source comes in before it is transmitted, their sequence numbers are compared. If they are equal, the duplicate is discarded. If they are different, the older one is thrown out. To guard against errors on the router-router links, all link state packets are acknowledged. When a line goes idle, the holding area is scanned in round robin order to select a packet or acknowledgment to send.

5.11.11. Computing the New Routes

Once a router has accumulated a full set of link state packets, it can construct the entire subnetwork graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately. Now Dijkstra's algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal operation resumed.

For a subnetwork with n routers, each of which have k neighbors, the memory required to store the input data is proportional to (k × n). For large subnetworks, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

However, problems with the hardware or software can wreak havoc with this algorithm. For example, if a router claims to have a line it does not have, or forgets a line it does have, the subnetwork graph will be incorrect. If a router fails to forward packets, or corrupts them while forwarding them, trouble will arise. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the subnetwork grows to the range of tens of hundreds of thousands of nodes, the probability of some router failing occasionally becomes non-negligible. The trick is to try to arrange to limit the damage when the inevitable happens. Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order. The OSPF protocol, which is increasingly being used in the Internet, uses a link state algorithm.

Another important link state protocol is IS-IS (Intermediate System-Intermediate System), which was designed for DECnet and later adopted by ISO for use with its connectionless network layer protocol (CLNP). Since then it has been modified to handle other protocols as well, most notably IP. IS-IS is used in numerous Internet backbones and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces in its link state information which network layer addresses it can reach directly. These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time.

Many of the innovations designed for IS-IS were adopted by OSPF. These include: a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics. As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multi-protocol environment.

5.11.12. Hierarchical Routing

As networks grow in size, the router routing tables grow proportionally. Not only is router memory consumed by ever increasing tables, but also more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network. When hierarchical routing is used, the routers are divided into domains, with each router knowing all the details about how to route packets to destinations within its own domain. The routers know nothing about the internal structure of other domains. When different networks are connected together, it is natural to regard each one as a separate domain in order to free the routers in one network from having to know the topological structure of the other ones.

For large networks a two-level hierarchy may be insufficient. It may be necessary to group the domains into clusters, the clusters into zones, the zones into groups, and so on. When routing is done hierarchically, there are entries for all the local routers as before, but all other domains have been condensed into a single router. Hierarchical routing has reduced the table entries. As the ratio of the number of domains to the number of routers per domain grows, the savings in the table space increases. Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of increased path length.

When a single network becomes very large, an interesting question is: How many levels should the hierarchy have? For example, consider a sub-network with 720 routers. If there is no hierarchy, each router needs 720 routing table entries. If the sub-network is partitioned into 24 domains of 30 routers each, each router needs 30 local entries plus 23 remote entries for a total of 53 entries. If a three-level hierarchy is chosen with eight clusters each containing 9 domains of 10 routers, each router needs 10 entries for local routers, 8 entries for routing to other regions within its own cluster, and 7 entries for distant clusters, for a total of 25 entries. Kamoun and Kleinrock have discovered that the optimal number of levels for an N router sub-network is (log N), requiring a total of ($e \times \log N$) entries per router. They have also shown that the increase in effective mean path length caused by hierarchical routing is sufficiently small that it is usually acceptable.

5.11.13. Routing for Mobile Hosts

Mobile hosts introduce a new complication to the routing issue. To route a packet to a mobile host, the network first has to find it. The subject of incorporating mobile hosts into a network is very young, but some of the issues are identified here and give a possible solution.

The model of the world that network designers typically use have a WAN consisting of routers and hosts. Connected to the WAN are LANs, MANs, and wireless cells. Users who never move are said to be stationary. They are connected to the network by copper wires or fiber optics. In contrast, there are two other kinds of users. Migratory users are basically stationary users who move from one fixed site to another from time to time but use the network only when they are physically connected to it. Roaming users actually compute on the run and want to maintain their connections as they move around. The term mobile users used to identify either of the latter two categories; i.e., all users who are away from home.

All users are assumed to have a permanent home location that never changes. Users also have a permanent home address that can be used to determine their home locations. The routing goal in systems with mobile users is to make it possible to send packets to mobile users using their home addresses, and have packets efficiently reach them wherever they may be. The issue, of course, is to find them.

The network world is divided up (geographically) into small units. The small units are called areas, where an area is typically a LAN or wireless cell. Each area has one or more foreign agents which keep track of all mobile users visiting the area. In addition, each area has a home agent which keeps track of users whose home is in the area, but who are currently visiting another area. When a new user enters an area, either by connecting to it or just wandering into the cell, his computer must register itself with the foreign agent there. The registration procedure typically works like this:

- Periodically, each foreign agent broadcasts a packet announcing its existence and address. A newly arrived mobile host may wait for one of these messages, but if none arrives quickly enough, the mobile host can broadcast a packet saying: "Are there any foreign agents around?"

- The mobile host registers with the foreign agent, giving its home address, current data link layer address, and some security information.

- The foreign agent contacts the mobile host's home agent and says: "One of your hosts is over here." The message from the foreign agent to the home agent contains the foreign agent's network address. It also includes the security information, to convince the home agent that the mobile host is really there.

- The home agent examines the security information, which contains a timestamp, to prove that it was generated within the past few seconds. If it is happy, it tells the foreign agent to proceed.

- When the foreign agent gets the acknowledgement from the home agent, it makes an entry in its tables and informs the mobile host that it is now registered.

Ideally, when a user leaves an area, that too should be announced to allow deregistration. However, many users abruptly turn off their computers when done. When a packet is sent to a mobile user, it is routed to the user's home LAN because that is what the address says should be done. Packets sent to the mobile user on its home LAN are intercepted by the home agent. The home agent then looks up the mobile user's new (temporary) location and finds the address of the foreign agent handling the mobile user. The home agent then does two things. First, it encapsulates the packet in the payload field of an outer packet and sends the latter to the foreign agent. This mechanism is called tunneling. After getting the encapsulated packet, the foreign agent removes the original packet from the payload field and sends it to the mobile user as a data link frame.

Second, the home agent tells the sender to henceforth send packets to the mobile host by encapsulating them in the payload of packets explicitly addressed to the foreign agent, instead of just sending them to the mobile user's home address. Subsequent packets can now be routed directly to the user via the foreign agent, bypassing the home location entirely.

The various schemes that have been proposed differ in several ways. First, there is the issue of how much of this protocol is carried out by the routers and how much by the hosts, and in the latter case, by which layer in the hosts. Second, a few schemes have routers along the way record mapped addresses so they can intercept and redirect traffic even before it gets to the home location. Third, in some schemes each visitor is given a unique temporary address; in others, the temporary address refers to an agent that handles traffic for all visitors. Fourth, the schemes differ in how they actually manage to arrange for packets that are addressed to one destination to be delivered to a different one.

One choice is changing the destination address and just retransmitting the modified packet. Alternatively, the whole packet, home address and all, can be encapsulated inside the payload of another packet sent to the temporary address. Finally, the schemes differ in their security aspects. In general when a host or router gets a rerouted message, it might have a couple of questions about whom it was talking to and whether or not this is a good idea.

5.11.14. Broadcast Routing

For some applications, hosts need to send messages to many or all other hosts. For example, a service distributing weather reports, stock market updates, or live radio programs might work best by broadcasting to all machines and letting those that are interested read the data. Sending a packet to all destinations simultaneously is called broadcasting. Various methods have been proposed for doing it.

One broadcasting method that requires no special features from the sub-network is for the source to simply send a distinct packet to each destination. Not only is the method wasteful of bandwidth, it also requires the source to have a complete list of all destinations. In practice this may be the only possibility, but it is the least desirable of the methods.

Flooding is another obvious candidate. Although flooding is ill-suited for ordinary point-to-point communication, for broadcasting it might rate serious consideration, especially if none of the methods described below are applicable. The problem with flooding as a broadcast technique is the same problem it has as a point-to-point routing algorithm. It generates too many packets and consumes too much bandwidth.

A third algorithm is multi-destination routing. If this method is used, each packet contains either a list of destinations or a bit map indicating the desired destinations. When a packet arrives at a router, the router checks all the destinations to determine the set of output lines that will be needed. The router generates a new copy of the packet for each output line to be used and includes in each packet only those destinations that are to use the line. In effect, the destination set is partitioned among the output lines. After a sufficient number of hops, each packet will carry only one destination and can be treated as a normal packet. Multi-destination routing is like separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

A fourth broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast, or any other convenient spanning tree for that matter. A spanning tree is a subset of the sub-network that includes all the routers but contains no loops. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some spanning tree for it to be applicable. Sometimes this information is available (e.g., with link state routing) but sometimes it is not (e.g., with distance vector routing).

The last broadcast algorithm is an attempt to approximate the behavior of the previous one, even when the routers do not know anything at all about spanning trees. The idea is remarkably simple. When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the line that is normally used for sending packets to the source of the broadcast. If so, there is an excellent chance that the broadcast packet itself followed the best route from the router and is therefore the first copy to arrive at the router. This being the case, the router forwards copies of it onto all lines except the one it arrived on. If, however, the broadcast packet arrived on a line other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

An example of the algorithm is called reverse path forwarding. The principal advantage of reverse path forwarding is that it is both reasonably efficient and easy to implement. It does not require routers to know about spanning tress, nor does it have the overhead of a destination list or bit map in each broadcast packet, as does multi-destination addressing. Nor does it require any special mechanism to stop the process, as flooding does (either a hop counter in each packet and a priori knowledge of the subnetwork diameter, or a list of packets already seen per source).

5.11.15. Multicast Routing

For some applications, widely-separated processes work together in groups; e.g., a group of processes implementing a distributed database system. It frequently is necessary for one process to send a message to all the other members of the group. If the group is small, it can just send each other member a point-to-point message. If the group is large, this strategy is expensive. Sometimes broadcasting can be used, but using broadcasting to inform 1,000 machines on a million-node network is inefficient because most receivers are not interested in the message. Thus, there is a need for a way to send messages to well-defined groups that are numerically large in size but small compared to the network as a whole.

Sending a message to such a group is called multicasting, and its routing algorithm is called multicast routing. To do multicasting, group management is required. Some way is needed to create and destroy groups, and for processes to join and leave groups. How these tasks are accomplished is not of concern to the routing algorithm. What is of concern is that when a process joins a group, it informs its host of this fact. It is important that routers know which of their hosts belong to which groups. Either hosts must inform their routers about changes in group membership, or routers must query their hosts periodically. Either way, routers learn about which of their hosts are in which groups. Routers tell their neighbors, so the information propagates through the subnetwork.

To do multicast routing, each router computes a spanning tree covering all other routers in the subnetwork. When a process sends a multicast packet to a group, the first router examines its spanning tree and prunes it, removing all lines that do not lead to hosts that are members of the group. Multicast packets are forwarded only along the appropriate spanning tree. Various ways of pruning the spanning tree are possible. The simplest one can be used if link state routing is used, and each router is aware of the complete subnetwork topology, including which hosts belong to which groups. Then, the spanning tree can be pruned by starting at the end of each path and working toward the root, removing all routers that do not belong to the group in question.

With distance vector routing, a different pruning strategy can be followed. The basic algorithm is reverse path forwarding. However, whenever a router with no hosts interested in a particular group and no connections to other routers receives a multicast message for that group, it responds with a prune message telling the sender not to send it any more multicasts for that group. When a router with no group members among its own hosts has received such messages on all its lines, it can respond with a prune message. In this way, the subnetwork is recursively pruned.

One potential disadvantage of this algorithm is that it scales poorly to large networks. Suppose that a network has n groups, each with an average of m members. For each group, m pruned spanning trees must be stored, for a total of $m \times n$ trees. When many large groups exist, considerable storage is needed to store all the trees. An alternative design uses core-base trees. Here, a single spanning tree per group is computed, with the root (the core) near the middle of the group. To send a multicast message, a host sends it to the core, which then does the multicast along the spanning tree. Although this tree will not be optimal for all sources, the reduction in storage costs from m trees to one tree per group is a major saving.

6. SUBNETWORK LAYER ISSUES AND ANALYSIS

It is true to say that almost any data communications network can be a subnetwork. The NASA Protocol architecture is based on an internetwork - and has the facilities to integrate both existing and new networking technologies. The minimum requirements for an subnetwork are that:

- It supports packet mode communications;
- Except for point-to-point data links, each system attached to the network must be individually addressable; and
- It must support the transparent communication of octet aligned data.

In practice only the first two requirements are absolute, the third can always be handled by a special adaptation layer. Other desirable features such as subnetwork prioritization of data may be important when the network will share both safety related and routine data transfer between different pairs of communicating systems. Some networks can prioritize virtual circuits on an a priori basis prioritizing the data of different users, while others can support this on a per packet or a per virtual circuit basis.

At the subnetwork layer, the networks are technology specific and there are a number of subnetworks and protocols each providing specialized features and services. Technologies such local area networks (LAN) improved on earlier technologies by introducing shared medium, multipoint communication, significantly higher bandwidth, low latency and very low error rates. The metropolitan area network (MAN) technologies, asynchronous transfer mode (ATM), frame mode services (frame relay), and the synchronous optical network (SONET) further increase the choice for subnetwork layer.

6.1. Services Assumed From Underlying Layers

The network layer in the protocol architecture is expected to operate over a wide variety of underlying communication services, either at the data-link layer or the network layer of the OSI reference model. One consistent characteristic of the underlying service assumed by the network layer protocol is that the service does not require explicit connection establishment and connection release. For underlying services that do require such operations, a "Subnetwork-Dependent Convergence Function" may be developed to emulate a connectionless service. The network layer protocol requires a minimum set of services from underlying protocols.

The subnetwork is responsible for not only routing packets from source to destination but also for providing error correction. This function is very important in a satellite environment Historically satellite links have provided error rates as good as or better than equivalent terrestrial links. To achieve the good error rate in spite of a signal-to-noise ratio much worse than on terrestrial links requires appropriate equipment design and possibly the use of error correcting codes. Whether on a noisy channel or not, data needs to be protected with efficient error detecting codes. These codes are now well understood and can protect data with a high level of safety. When data is found to be in error, the source is notified, and it retransmits the corrupted data frame.

Error detection and retransmission can give efficient operation when the error rate is 1 bit in 10^5 or better. With error rates worse than the above, error detection alone is generally a poor technique for most purposes and needs to be supplemented with the use of error correcting codes.

6.2. Channel Efficiency

The efficiency with which a link operates is related by the probability of a frame of data having to be retransmitted. Suppose that a frame that is transmitted contains D bits of data and the number of header bits is H. These header bits contain address and control information.

Let PB be the probability of a bit in error. Assuming that the bit errors occur independently, the probability that all the bits in a frame are error free given by:

$$\text{Probability a frame is error free} = (1 - PB)^{(D+H)} \quad \ldots\ldots(1)$$

Therefore, the probability that the frame has an error (PE) is given by:

$$PE = 1 - (1 - PB)^{(D+H)} \quad \ldots\ldots(2)$$

On terrestrial links the bit errors are not all independent and error tend to occur in burst. Hence, the probability that a frame is in error (PE) is best assessed using empirical measurements rather than using probabilistic models. On satellite links the noise caused by the poor quality of the space link is Gaussian and the probabilistic models give a good approximation of PE.

The probability that a frame has to be retransmitted once because of error is PE and the probability that it has to be retransmitted a second time is PE^2 and the probability that the frame has to be retransmitted n times is PE^n. Therefore, the mean number of bits (N) transmitted is given by:

$$N = (D + H)(1 + PE + PE^2 + PE^3 + \ldots\ldots) = (D + H) / (1 - PE) \quad \ldots\ldots(3)$$

Now substituting the PE in the above equation gives:

$$N = (D + H) / (1 - PB)^{(D+H)} \quad \ldots\ldots(4)$$

The transmission efficiency of a link with bit error rate of PB is obtained by taking the ratio of the information bit to the average number of bit transmitted to successfully transfer the data across the link. Therefore,

$$\text{Link Efficiency} = D \times (1 - PB)^{(D+H)} / (D + H) \quad \text{.........(5)}$$

The link efficiency for different bit error rates and a fixed header size of 100 bits is shown in Figure 7. It can be observed that for any given error rate there is an optimum transmission frame size. For error rates better than 1 bit in 10^5, the transmission efficiency can be better than 90 percent. For error rate of 1 bit in 10^3 or worse, transmission efficiency is below 50 percent, and often well below it.

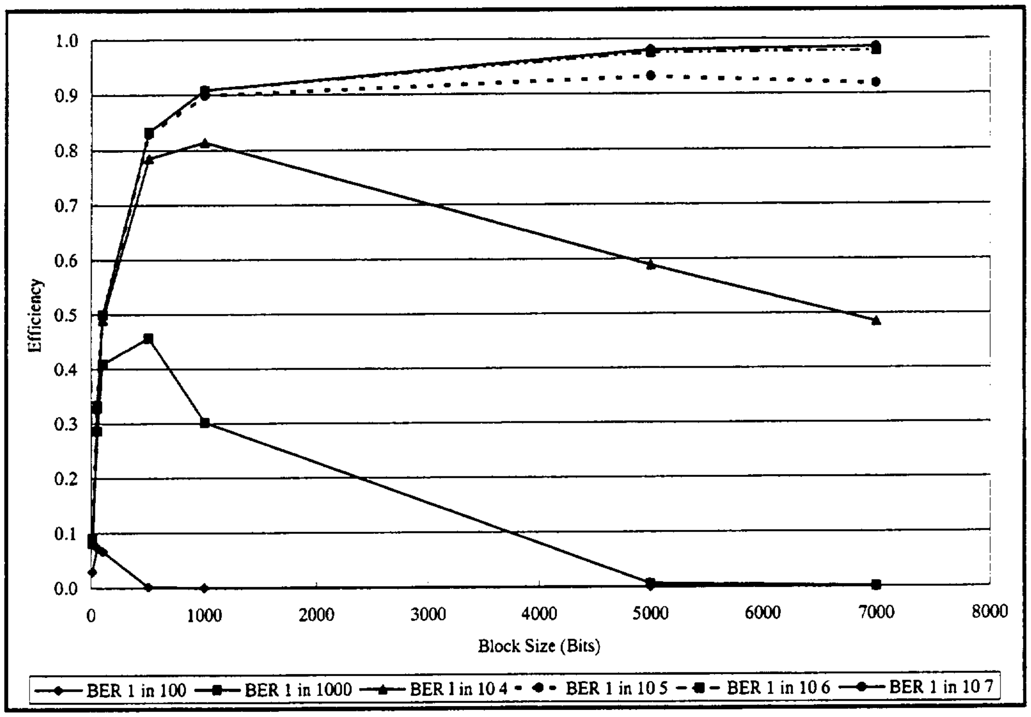

Figure 7. Transmission Efficiency for Different Error Rates with Overhead of 100 bits

6.3. Response Time

Retransmission of frames in error has a more serious effect on response time in a satellite environment because of the long propagation time. A source has to wait at least 540 milliseconds before it receives an acknowledgement from the destination saying that the frame was received correctly. If it has many frames to transmit, it will be transmitting new frames before the acknowledgement for the frame in error is received. It must therefore have a buffer large enough to hold many frames, all waiting for acknowledgement.

Let TB be the time taken to transmit a frame if it is received correctly the first time. In a satellite environment TB is approximately 270 milliseconds for a short frame. Let TR be the elapsed time between when a frame is received incorrectly and when it is received again after retransmission. The source might typically wait for 560 milliseconds after sending a frame for an acknowledgement of successful transmission and then try again. Again, TR is approximately 560 milliseconds for a short frame. If PE the probability that a transmitted frame is in error, then the response time RT is given by:

$$RT = TB + TR\,(PE + PE^2 + PE^3 + PE^4 + \ldots\ldots\ldots)$$

$$= TB + TR \times PE / (1 - PE) \quad \ldots\ldots (6)$$

Now substituting PE from equation 2, we get:

$$TR = TB + TR\,[\{1 - (1 - PB)^{(D+H)}\} / (1 - PB)^{(D+H)}]$$

$$= TB + TR\,[1 / (1 - PB)^{(D+H)} - 1] \quad \ldots\ldots (7)$$

Depending on the link level protocol used, the delay may be dominated by the propagation delay or the transmission time may become significant. For the following analysis it is assumed that the frame transmission time is significant and has to be included in calculating the response time RT.

Let C be the channel capacity in bits/sec. Then, the time to transmit a frame is given by (D + H)/S seconds. If TD is the propagation delay in seconds, then:

$$TB = TD + (D + H)/S \quad \ldots\ldots (8)$$

Therefore,

$$TR = TD + TA + TB \quad \ldots\ldots (9)$$

Where, TA is the processing time at the destination to send an acknowledgement. Substituting the values of TB and TR from equations 8 and 9 into 7, the mean time to transmit a frame is:

$$RT = TD + (D + H) / S + [2 \times TD + TA + (D + H) / S]\,\{[1 / (1 - PB)^{(D+H)}] - 1\} \quad \ldots\ldots (10)$$

Figure 8 shows the mean delay as a function of packet size for various bit error rates when the line speed is 1 Mbps.

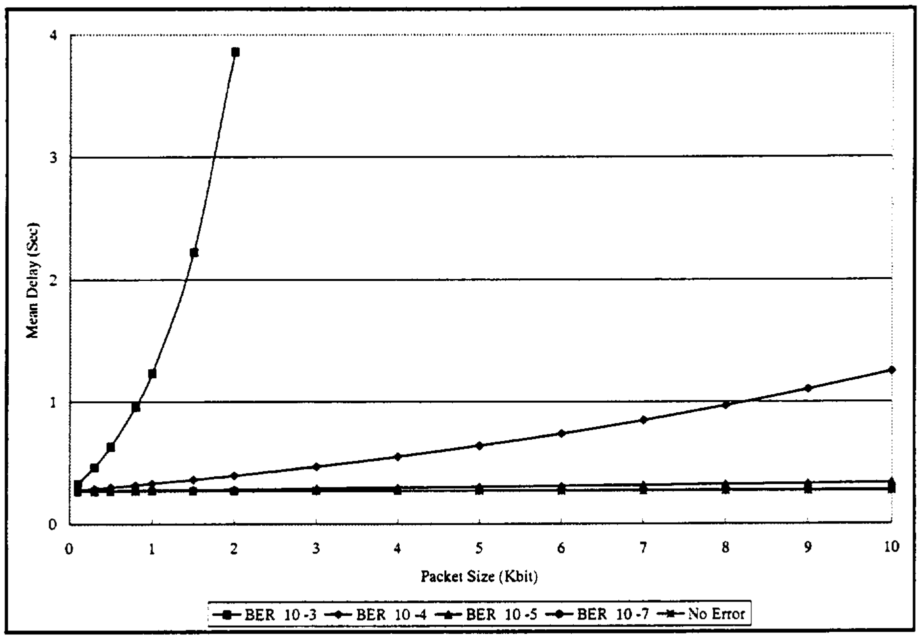

Figure 8. Mean Delay with Error Detection and Retransmission for a Line Speed of 1.5 Mbps.

Figure 9 shows the mean delay as a function of packet size for various bit error rates when the line speed is 5 Mbps.

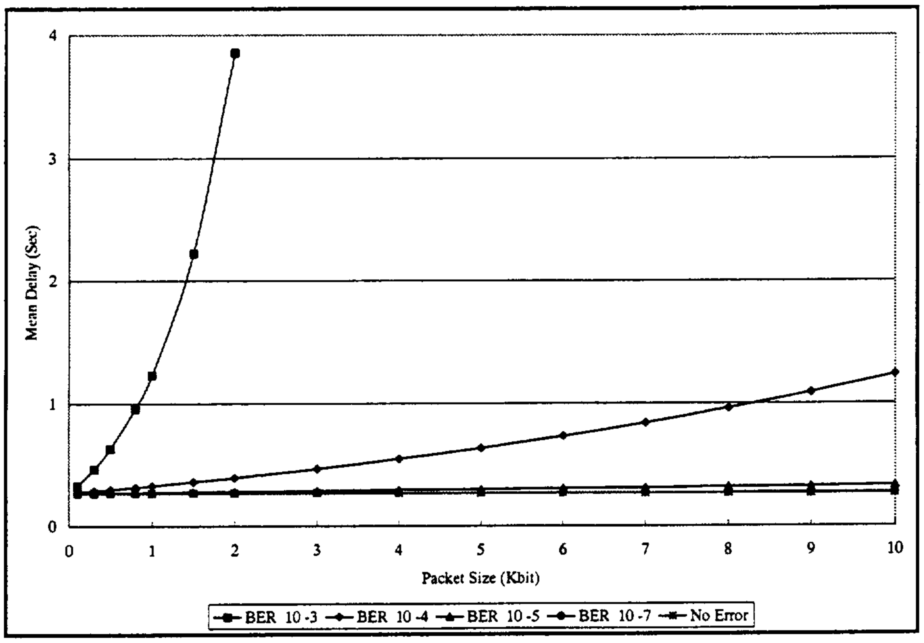

Figure 9. Mean Delay with Error Detection and Retransmission for a Line Speed of 5 Mbps

Figure 10 shows the mean delay as a function of packet size for various bit error rates when the line speed is 50 Mbps.

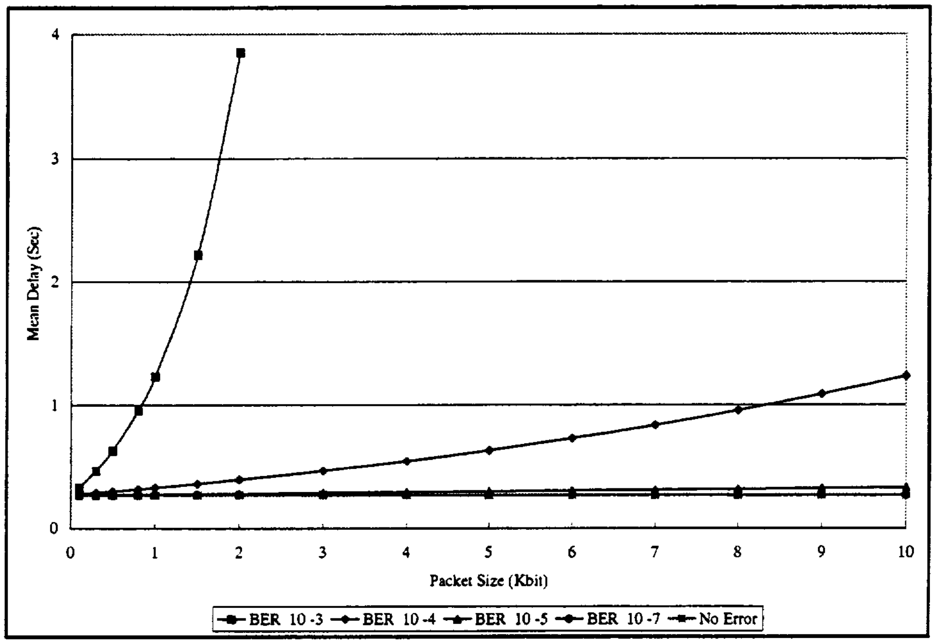

Figure 10. Mean Delay with Error Detection and Retransmission for a Line Speed of 50 Mbps

6.4. Automatic Repeat Request (ARQ)

A set of protocols used to detect an error in the data frame and retransmit the erred frames is called an Automatic Repeat Request (ARQ) protocol. Most terrestrial transmission links use some form of ARQ for error correction. ARQ methods come in a number of flavors. The most popular are Stop-and-wait ARQ, Go-back-N continuous ARQ, and Selective-repeat continuous ARQ.

Stop-and-wait ARQ uses the simple stop-and-wait acknowledgement scheme. The source transmits a single frame and then waits for an acknowledgment. In the mean time no frames of data can be sent until the destination's reply arrives at the source. The destination sends a positive acknowledgment (ACK) if the frame is received without error and a negative acknowledgment (NAK) if the frame is received in error. There are a couple of issues that need to be addressed to make this scheme reliable.

The transmitted frame may be corrupted by noise and the destination never receives the frame. In this case the source waits indefinitely for an acknowledgment to come. This issue can be resolved by using a timer at the source. After the source transmits a frame, it waits for an acknowledgment. It an acknowledgment is not received before the timer expires, the same frame is transmitted again. This scheme requires that the source keep a copy of a transmitted frame until an ACK is received for that frame.

The second issue is related to the receipt of duplicate frame by the destination. For example, the source sends a frame and the frame is received without error by the destination. The destination forwards an ACK but the ACK is damaged in transit and is not recognizable by the source. Therefore, after time out the source retransmits the frame again. This duplicate frame arrives at the destination and is accepted as new frame. This issue can be resolved using alternate labeling scheme for frames and acknowledgements.

The principle advantage of stop-and-wait ARQ is its simplicity. It's principle disadvantage is that it is an inefficient protocol from the point of line utilization. The line utilization can be improved by using sliding window techniques. This is referred to as continuous ARQ.

One variant of continuous ARQ is known as go-back-N-ARQ. In this protocol, a sender may send a series of frames determined by the window size. The destination detects an error in a frame, it sends a NAK for that frame. The destination discards all future incoming frames until the frame in error is correctly received. Thus, when it receives a NAK, the source must retransmit the frame in error plus all succeeding frames.

Selective-repeat ARQ provides a more refined approach than go-back-N. In this scheme the frames that are retransmitted are those that receive a NAK. This would appear to be more efficient than the go-back-N approach. On the other hand, the destination must have storage to save post-NAK frames until the error frame is retransmitted, and the logic for inserting the frame in the proper sequence. In addition, the source will require more complex logic to be able to send frames out of sequence. Because of such complications, the go-back-N protocol is more commonly used.

The window size requirements is more restrictive for selective-repeat than for go-back-N. In the go-back-N protocol, a window size of 2^{n-1} can be used out of the sequence number space of 2^{n}. But, for selective-repeat ARQ protocol, there is an overlap between the sending and receiving windows. In order to overcome this problem, the maximum widow size should be no more than half the range of sequence numbers to avoid confusion.

6.5. Performance

It appears that go-back-N and Selective-repeat are more efficient than stop-and wait protocols. An approximate relationship for link utilization is developed below to compare the degree of improvement in link performance. Note that there are a number of detailed models in the literature and this area has been a fertile ground for methods that offer incremental improvement in link performance. Let:

TR - time to transmit a frame

TT	-	total time line is engaged in transmitting a single frame
A	-	ratio of propagation time to the transmit time of a frame
TP	-	propagation time
U	-	utilization of the link
NR	-	expected number of transmissions of a frame
N	-	window size

Utilization of the link is defined as the ratio of the time to transmit a frame to the total time the line is engaged in the transmission of a single frame.

$$U = TR / TT \qquad (11)$$

For error free operation using the stop-and-wait ARQ,

$$U = TR / (TR + 2 \times TP)$$

Factoring TR from the right side of the equation yields,

$$U = 1/ (1 + 2 \times TP/TR) = 1/ (1 + 2 \times A)$$

If an error occurs during the transmission of the frame, this can be taken into account by modifying equation 11. Therefore,

$$U = TR/(NR \times TT)$$

Therefore, for the stop-and-wait ARQ error correction protocol, the link utilization is given by:

$$U = 1 / (NR \times (1 + 2 \times A)) \qquad (12)$$

A simple expression for the expected number of retransmissions of a frame can be derived in terms of the line error rate PB. In order to reduce the complexity, it is assumed that the ACKs and NAKs are delivered without being affected by error. Therefore, the probability that it will take exactly i attempts to transmit a frame successfully is PB $^{(i-1)}$ (1 – PB). Therefore,

$$NR = \sum_{i=1}^{\alpha} i \times PB^{(i-1)}(1-PB) = 1/(1-PB)$$

Substituting the value of NR in equation 12, for stop-and-wait ARQ:

$$U = (1 - PB) / (1 + 2 \times A) \qquad (13)$$

When more than one frame is transmitted using the window size of N, there are two distinct cases. In case 1, where N > (2 × A + 1), acknowledgment for the first frame reaches the source before it has exhausted it's window. Thus, the source can transmit continuously with no pause. Therefore, the line utilization is 1.

In case 2, when the N < (2 × A + 1), the source exhaust it's window and cannot send additional frames until it receives an acknowledgment. Thus, the line is used N time units out of a period of (2 × A + 1) time units. Therefore, for error free operation the utilization of line is given by:

$$U = \begin{cases} 1 & N > (2 \times A + 1) \\ N/(2 \times A + 1) & N < (2 \times A + 1) \end{cases}$$

For selective-repeat ARQ, using the same argument as stop-and-wait ARQ, the line utilization when line error is present is obtained by dividing the error free utilization by NR and substituting the value for NR = 1 / (1 – PB), the line utilization for Selective-repeat:

$$U = \begin{cases} 1 - P & N > (2 \times A + 1) \\ N(1 - P)/(2 \times A + 1) & N < (2 \times A + 1) \end{cases} \quad \text{.......................................}(14)$$

For go-back-N ARQ, the same reasoning still applies, but the approximate value for NR has to be derived. To get the value for NR, notice that each error generates a requirement to retransmit K frames rather than just one frame. Therefore,

NR = E [number of transmitted frames to successfully transmit one frame]

$$NR = \sum_{i=1}^{\alpha} f(i) \times PB^{(i-1)}(1 - PB)$$

Where, f(i) is the total number of frames transmitted if the original frame must be transmitted i times. This can be expressed as,

$$f(i) = 1 + (i - 1)K = (1 - K) + K \times i$$

Substituting the value of f(i), in the equation for NR,

$$NR = (1 - K)\sum_{i=1}^{\alpha} PB^{(i-1)}(1 - PB) + k\sum_{i=1}^{\alpha} i \times PB^{(i-1)}(1 - PB)$$

$$NR = 1 - K + K/(1 - PB) = (1 - PB + K \times PB)/(1 - PB)$$

For sliding window protocols, the value of K can be approximated to (2 × A + 1) when N > (2 × A + 1), and K = N for N < (2 × A + 1). Therefore, the utilization for the Go-back-N protocol is given by:

$$U = \begin{cases} (1 - PB)/(1 + 2 \times A \times PB) & N > 2 \times A + 1 \\ N \times (1 - PB)/[(2 \times A + 1)(1 - PB + N \times PB)] & N < 2 \times A + 1 \end{cases} \quad \ldots\ldots\ldots\ldots\ldots (15)$$

By substituting N = 1, utilization for both selective-repeat and go-back-N reduces to that of Stop-and-wait. The equations for line utilization are approximations because the effect of errors in acknowledgment frames and, in the case of Go-back-N, errors in retransmitted frames other than the frame initially in error have been ignored.

Figure 11 presents the normalized throughput as a function of the parameter a when the bit error rate is 10^{-5} for Go-back-N and Selective Reject error correction techniques using window size of 7 and 127. The Go-back-N and Selective Reject for window size 7 overlap one another. The same is true for window size 127.

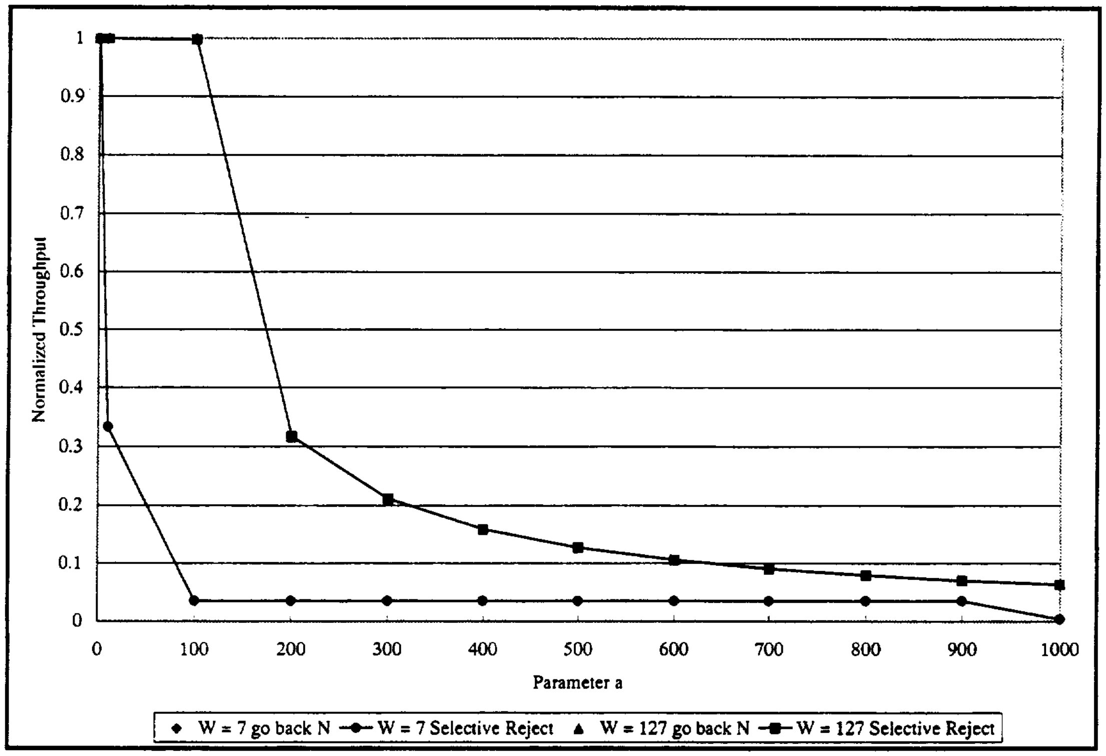

Figure 11. Normalized Throughput as a Function of Parameters a for BER 10^{-5} using Go-Back-N and Selective Reject Protocols

Figure 12 shows the normalized throughput as a function of the parameter a when the bit error rate is 10^{-7} for Go-back-N and Selective Reject error correction techniques using window size of

7 and 127. The Go-back-N and Selective Reject for window size 7 overlap one another. The same is true for window size 127.

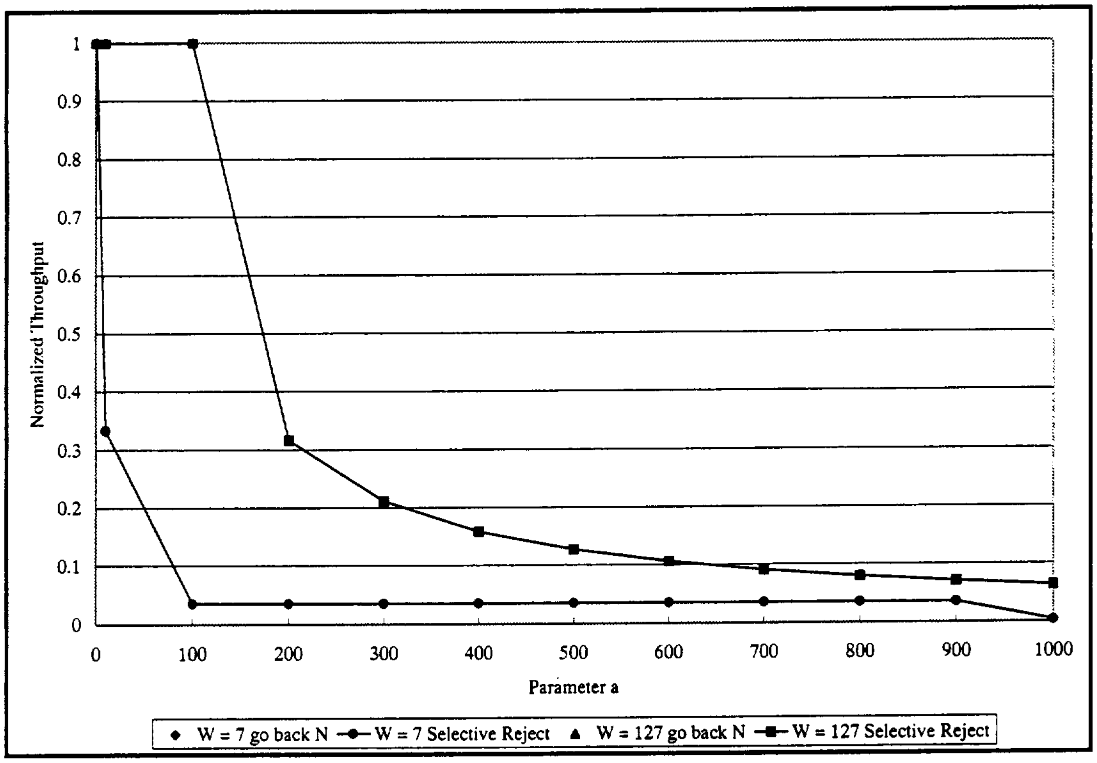

Figure 12. Normalized Throughput as a Function of Parameters a for BER 10^{-7} using Go-Back-N and Selective Reject Protocols

Figure 13 shows the normalized throughput as a function of the parameter a when the bit error rate is 10^{-9} for Go-back-N and Selective Reject error correction techniques using window size of 7 and 127. The Go-back-N and Selective Reject for window size 7 overlap one another. The same is true for window size 127.

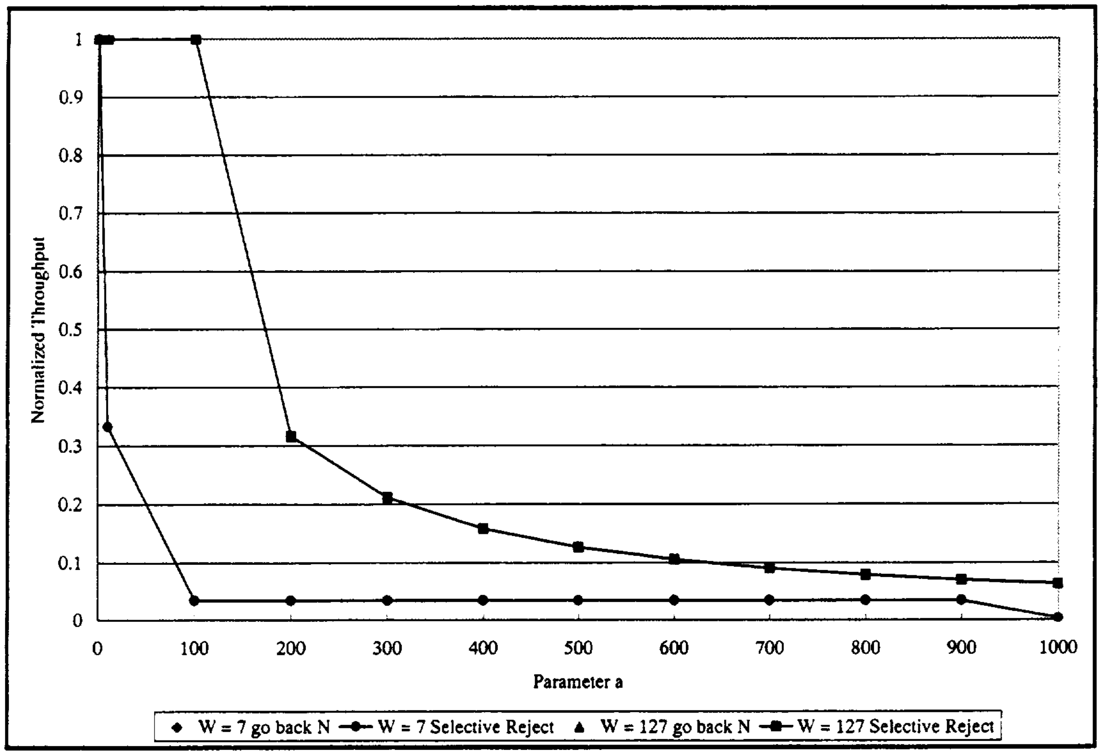

Figure 13. Normalized Throughput as a Function of Parameters a for BER 10 $^{-9}$ using Go-Back-N and Selective Reject Protocols

Figure 14 presents the normalized throughput as a function of window size when the bit error rate is 10 $^{-5}$ and the parameter a = 10 and a = 100 for Go-back-N and Selective Reject error correction techniques.

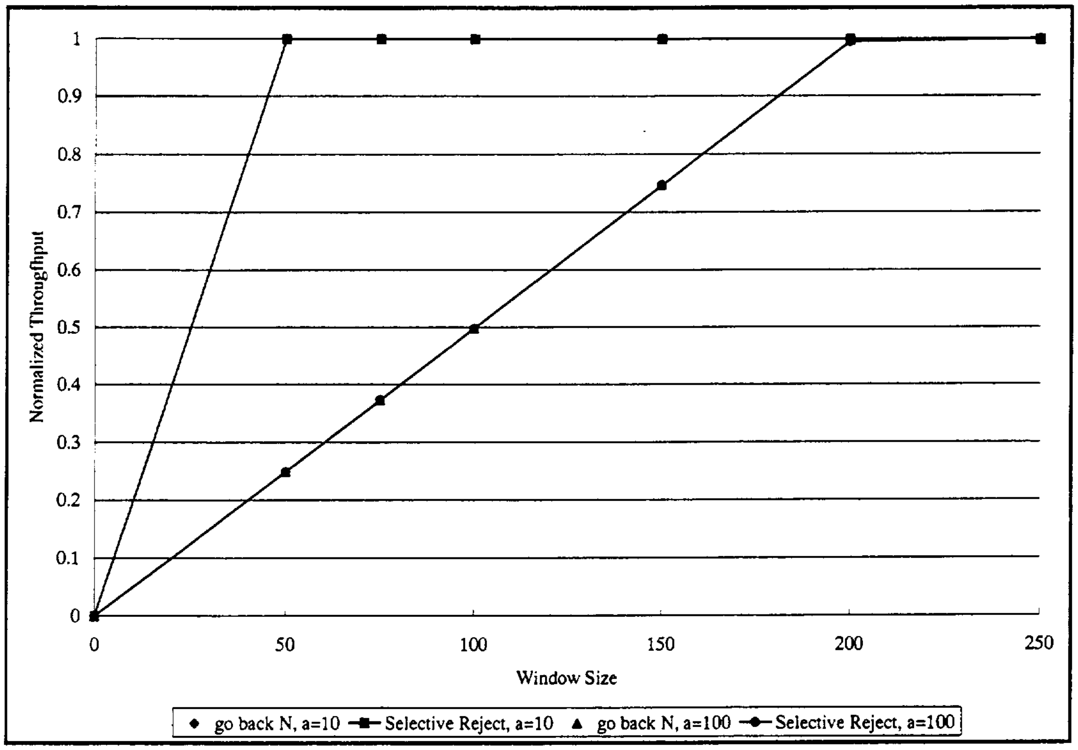

Figure 14. Normalized Throughput as a Function of Window Size for a = 10 and a = 100

6.6. Link Layer Enhancements

As discussed previously, link error rates are a major concern in the satellite networking environment. As a result, designers have proposed a variety of solutions. Some of these have been presented above. This section looks at the forward error correction technique.

Forward Error Correction Schemes

One means of improving link conditions is to simply increase transmitter power and/or use larger antennas. However, this is a rather expensive remedy and more workable link-level solutions are commonplace. A variety of forward error correction (FEC) encoding schemes have been proposed for data corruption, ranging from well-known convolution codes to concatenated codes. The level of encoding used depends on the desired link bit error rate, the prevailing link/atmospheric conditions and even the traffic type. Many advanced coding schemes also make use of bit interleaving to reduce the effects of burst errors, by spreading out the errors over multiple packets. This can improve the effectiveness of existing error checksums and single-bit error correction mechanisms. Typically, good systems can achieve bit error rate values over the 10^{-7} range, resulting in packet error rates of 10^{-9} or better. However, increased encoding complexity can slow down satellite modems and reduce bandwidth efficiency since data redundancy incurs additional overhead. Hence, many advanced satellite networks support a range of encoding schemes for different types of traffic and network conditions. For example, reliable transmission of delay-sensitive streams usually requires the highest level of encoding/interleaving, and more delay-insensitive applications can use reduced encoding levels. In addition, many sate-of-the-art satellite devices also use data compression techniques to counteract bandwidth inefficiencies caused by encoding.

7. REFERENCES

1. Borman, D., Braden, R., and Jacobson, V., "TCP Extensions for High Performance; RFC-1323," Internet Requests for Comments, no. 1323, May 1992.

2. Brakmo, L. S., O'Malley, S. W., and Peterson, L. L., "TCP Vegas: New Techniques for Congestion Avoidance and Control," Proc. ACM SIGCOMM '94, August 1994.

3. Cheriton, D., "VMTP: Versatile Message Transaction Protocol", RFC 1045, February 1988.

4. Clark, D., Lambert, M., and Zhang, L. "NETBLT: A Bulk Data Transfer Protocol", RFC 998, March 1987.

5. Computer Networks & Software, Inc., "Architectural Methodology Report" to NASA GRC for the In Space Internet Node Development Project, NASA Contract No. NAS 3 99165, Task Order 1, August 16, 1999

6. Computer Networks & Software, Inc., "Protocol Architecture Model Report" to NASA GRC for the In Space Internet Node Development Project, NASA Contract No. NAS 3 99165, Task Order 1, August 16, 1999

7. Comer, D. E., Internetworking with TCP/IP, Vol. 1: Principles, Protocols and Architecture, 2nd ed., Prentice Hall, 1991.

8. Floyd, S. and Jacobson, V., "On Traffic Phase Effects in Packet-Switched Gateways," Internetworking: Research and Experience, vol. 3, no. 3, September 1992.

9. Hoe, J. C., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," Proc. ACM SIGCOMM '97, August 1996.

10. Jacobson, V., "4BSD Header Prediction", ACM Computer Communication Review, April 1990.

11. Jacobson, V., "Congestion Avoidance and Control," Proc. ACM SIGCOMM '88, August 1988.

12. Jacobson, V., and R. Braden, "TCP Extensions for Long-Delay Paths", RFC-1072, October 1988.

13. Jacobson, V., Braden, R., and Zhang, L., "TCP Extension for High-Speed Paths", RFC-1185, October 1990.

14. Jain, R., "Divergence of Timeout Algorithms for Packet Retransmissions", Proc. Fifth Phoenix Conf. on Comp. and Comm., March 1986.

15. Karn, P. and Partridge, C., "Estimating Round-Trip Times in Reliable Transport Protocols", Proc. SIGCOMM '87, August 1987.

16. Kent C.A. and Mogul, J. C., "Fragmentation Considered Harmful," Proc. of ACM SIGCOMM '87, August 1987.

17. Keshav, S., "A Control-Theoretic Approach to Flow Control," Proc. ACM SIGCOMM '91, September 1991.

18. Liu, Z., et al., "Evaluation of TCP Vegas: Emulation and Experiment," Proc. ACM SIGCOMM '95, August 1995.

19. Mathis, M., and Mahdavi, J., "Forward Acknowledgment: Refining TCP Congestion Control," Proc. ACM SIGCOMM '96, August 1996.

20. Mogul, J. and Deering, S., "Path MTU Discovery; RFC-1191," Internet Requests for Comments, no. 1191, November 1990.

21. Nagle, J., "Congestion Control in TCP/IP Internetworks", RFC 896, FACC, January 1984.

22. Partridge, C. and Shepard, T. J., TCP/IP Performance Over Satellite Links, IEEE Network, September/October 1997.

23. Paxson, V., "End-to-End Routing Behavior in the Internet," Proc. ACM SIGCOMM '97, August 1996.

24. Postel, J, "Internet Protocol; RFC-791", Internet Requests for Comments, no. 791, September 1981.

25. Stevens, W.R., TCP/IP Illustrated, Vol. 1, Addison Wesley, 1994.

26. Zhang, Y., et al., "Satellite Communications in the Global Internet – Issues, Pitfalls, and Potential," Proc. INET '97, 1997.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	**2. REPORT DATE** January 2000	**3. REPORT TYPE AND DATES COVERED** Final Contractor Report

4. TITLE AND SUBTITLE

Trade-off Analysis Report

5. FUNDING NUMBERS

WU–632–50–51–00
NAS3–99165
Task 1

6. AUTHOR(S)

Chris Dhas

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Networks and Software, Inc.
7405 Alban Station Ct.
Springfield, Virginia 22150

8. PERFORMING ORGANIZATION REPORT NUMBER

E–12071

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135–3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CR—2000-209785

11. SUPPLEMENTARY NOTES

Project Manager, Thomas Wallett, Communications Technology Division, NASA Glenn Research Center, organization code 5610, (216) 433–3673.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category: 32 Distribution: Nonstandard

This publication is available from the NASA Center for AeroSpace Information, (301) 621–0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

NASA's Glenn Research Center (GRC) defines and develops advanced technology for higi priority national needs in communications technologies for application to aeronautics and space. GRC tasked Computer Networks and Software Inc. (CNS) to examine protocols and architectures for an In-Space Intemet Node. CNS has developed a methodology for network reference models to support NASA's four mission areas: Earth Science, Space Science, Human Exploration and Development of Space (HEDS), Aerospace Technology. CNS previously developed a report which applied the methodology to three space Internet-based communications scenarios for future missions. CNS conceptualized, designed, and developed space Internet-based communications protocols and architectures for each of the independent scenarios. GRC selected for further analysis the scenario that involved unicast communications between a Low-Earth-Orbit (LEO) International Space Station (ISS) and a ground terminal Internet node via a Tracking and Data Relay Satellite (TDRS) transfer. This report contains a tradeoff analysis on the selected scenario. The analysis examines the performance characteristics of the various protocols and architectures. The tradeoff analysis incorporates the results of a CNS developed analytical model that examined performance parameters.

14. SUBJECT TERMS

Network; Protocol

15. NUMBER OF PAGES	72
16. PRICE CODE	A04

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	