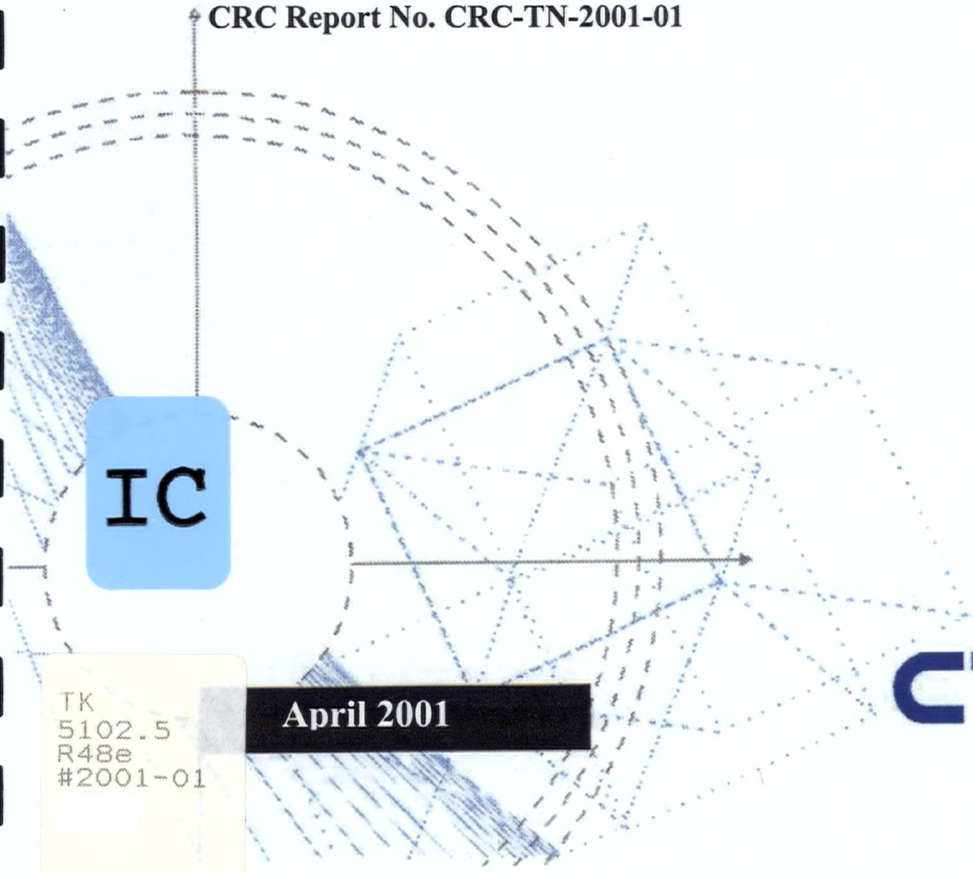
TCP OVER SATELLITE

A Review of TCP Over Satellite Links: Problems and Suggested Solutions

Peter Andreadis and Jing Peng

CRC Report No. CRC-TN-2001-01



April 2001

Communications Research Centre
Centre de recherches sur les communications

TK
5102.5
R48e
#2001-01

TCP OVER SATELLITE

A Review of TCP Over Satellite Links:

Problems and Suggested Solutions

Peter Andreadis and Jing Peng

CRC Technical Note No. CRC-TN-2001-01
Ottawa, April 2001

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The most dominant transport protocol used in sending Internet applications is the Transmission Control Protocol (TCP). Although TCP was designed to provide reliable end-to-end connections over various types of networks, it has several problems when running over long delay paths such as satellite links. Large bandwidth-delay products and high bit error rates are the two major factors that affect most connections using TCP over satellite. Current research suggests that two major problems of TCP are the coupling between congestion detection and error control, and the inefficiency of the congestion control and loss recovery algorithms over long delay paths such as satellite links.

In this document, a review of standard TCP mechanisms and functions is given. The problems TCP connections encounter over satellite links are presented, and a number of solutions including standard extensions, gateway options such as TCP splitting and TCP spoofing, and proposed TCP enhancements such as sharing TCP state information are analyzed. Implementing such extensions and enhancements will play an important roll in improving the efficiency of TCP connections over satellite networks.

RÉSUMÉ

Dans le domaine du transport de données sur Internet, le Transmission Control Protocol (TCP) est le protocole le plus dominant. Bien que TCP fut conçu dans le but d'offrir des connexions point à point fiables sous différents types de réseaux, son utilisation sur des liens à longs délais, comme les liens satellites, soulève plusieurs problèmes. Les deux principaux facteurs affectant la performance de TCP sur des liens satellites sont les produits délais-largeur de bande (*delay-bandwidth product*) élevés ainsi que les hauts taux d'erreurs (*bit error rate*). Des travaux de recherche suggèrent que les deux problèmes majeurs de l'utilisation de TCP sur des liens à longs délais sont la relation entre la détection de congestion et le control d'erreurs ainsi que l'inefficacité des algorithmes de contrôle de congestion et de correction d'erreurs.

Ce rapport présente d'abord un bilan des différents mécanismes et fonctions standards du protocole TCP. Les problèmes inhérents aux connexions TCP sur des liens satellites sont ensuite présentés et certaines solutions à ces problèmes telles que les extensions standards, les options de *gateway* (*TCP splitting*, *TCP spoofing*) ainsi que d'autres améliorations proposées (partage d'information d'état de TCP) sont analysées. L'utilisation de telles extensions jouera un rôle important au niveau de l'amélioration de l'efficacité de connexions TCP dans les réseaux satellites.

EXECUTIVE SUMMARY

Satellite communication offers many benefits, such as wide coverage areas, broadcast capabilities, and ability to reach remote and geographically adverse locations at relatively low cost. Communication satellites have been providing military communications, international telephony and broadcast TV for many years. Now, these satellites are being used as a complementary source to the terrestrial communication networks.

While satellite networks provide an extension to the Internet, they also cause some problems to the reliable end-to-end data transmission of Internet applications, which is realized by the Transmission Control Protocol (TCP). Although TCP was designed to be robust and flexible to operate in various environments, it cannot always perform efficiently over a satellite link. The main problems arise from the high bit error rates (BER), large delay-bandwidth products, intermittent connectivity, variable round trip time, and asymmetric link capacities associated with satellite links. All of these problems result in large bandwidth inefficiency and can lead to poor end-to-end application performance. For satellite links, where bandwidth is an expensive resource, inefficient use is costly.

In this document, many possible algorithms which can solve various problems that TCP connections encounter over satellite where examined. Some solutions have not been thoroughly tested. One such solution is sharing TCP state information, which can improve bandwidth efficiency while maintaining end-to-end reliability. Such algorithms will play an important roll in improving future use of TCP over satellite networks

Other solutions and enhancements have become part of IETF standards. The enhancements recommended by the authors are increasing the initial advertised window size, window scaling such that the BDP is satisfied, and use of the SACK mechanism. These enhancements should be implemented and be part of all standard TCP stacks for connections over satellite links. However, many of the have not been widely implemented at this time.

More modifications and extensions to these enhancements are being examined and tested, but it will be some time before they are reliable enough to become standards. These modifications will have a profound effect on connections using TCP over long delay paths, and will improve application performance and the bandwidth efficiency of satellite networks.

1. INTRODUCTION

Satellite communication offers many benefits, such as wide coverage areas, natural broadcast capabilities, and ability to reach remote and geographically adverse locations at relatively low cost. Communication satellites have been providing military communications, international telephony and broadcast TV for many years. Now, these satellites are being used as a complementary source to the terrestrial communication networks.

While satellite networks provide an extension to the Internet, they also cause some problems to the reliable end-to-end data transmission of the Internet, which is realized by the Transmission Control Protocol (TCP). Although TCP was designed to be robust and flexible to operate in a various environments, it cannot always perform efficiently over a satellite link. The main problems arise from the high bit error rates (BER), large delay-bandwidth products, intermittent connectivity, variable round trip time, and asymmetric link capacities associated with satellite links. In addition, most TCP optimizations have been made based on assumptions that are true for terrestrial networks but fail for satellite links. For example, TCP's congestion control mechanisms are based on the assumption that loss of segments is due to congestion rather than corruption, where such conditions usually cause excessive TCP timeouts, retransmissions, or even abortion of the connection. All of these problems result in large bandwidth inefficiency and can lead to poor end-to-end application performance. For satellite links, where bandwidth is an expensive resource, inefficiency in its use is costly.

This document first examines the OSI model with a more detailed look at TCP. Then the characteristics of satellite links and the problems of using TCP over such an environment are described and the performance issues that arise are examined. Solutions to these problems can be divided into two categories: modifications without the need to modify the TCP stack, and protocol enhancements to TCP for satellite use. They will be discussed in sections 4 "Improving Satellite Link Performance without Modifying the TCP Stack", and 5 " TCP Enhancements for Satellite Links", respectively.

2. OVERVIEW OF TCP

2.1 OSI MODEL

The OSI (Open Systems Interconnection) model is part of an international standard dealing with connecting systems that are open for communication with other systems[1]. The OSI model has seven layers. Each layer performs certain functions that allow users to communicate without noticing the underlying interfaces and actual physical links needed to send and receive data. Let us examine each layer starting with the physical layer.

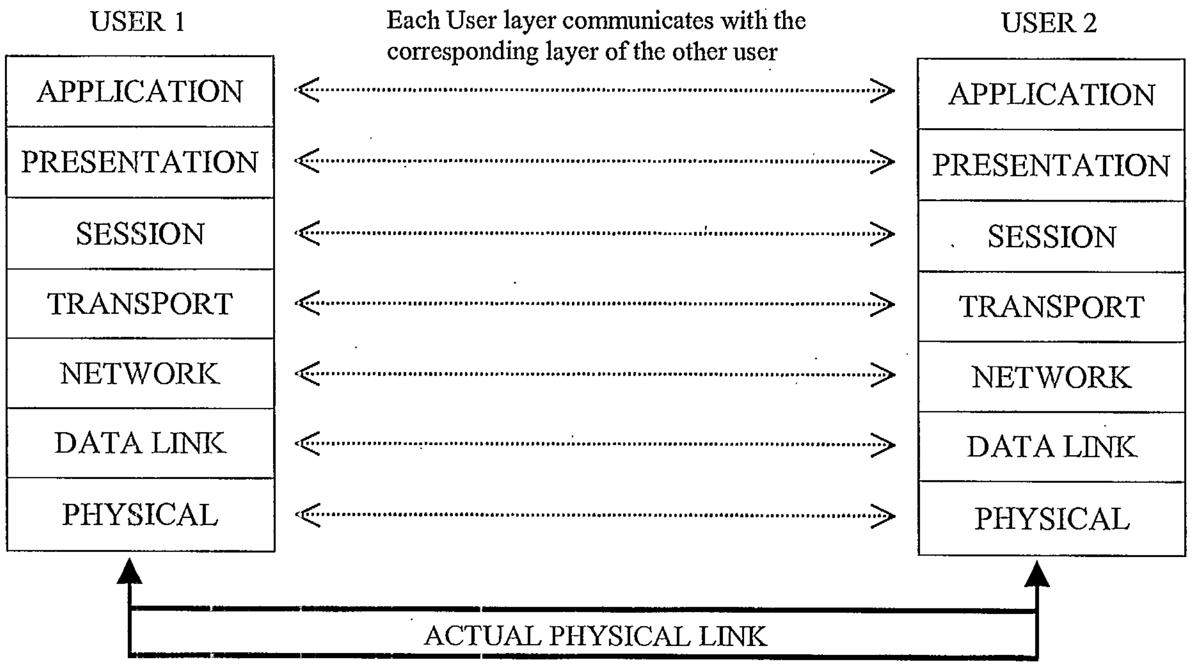


Figure 1: The OSI model

2.1.1 The Physical Layer

The physical layer is concerned with transmitting raw bits over the physical medium. Thus, it deals with bit formatting, bit rates, bit error rates, and all the physical and electrical interfaces between the actual user hardware and the network terminating equipment.

The most common forms of the physical medium are coaxial cable, twisted copper pair, fibre optics, and wireless. A common protocol of this layer is the Physical Layer Convergence Protocol (PLCP), which maps the data frames into cells of bits to be transmitted over the physical medium. Other interface standards at the physical layer are the EIA (EIA 232-E, EIA 449) and V series (V.2, etc.) recommendations.

2.1.2 The Data Link Layer

The data link layer's main task is to ensure the reliable delivery of data across the physical link[2]. It accomplishes this task by breaking the input data into data frames and provides the frame identification, error control and flow control needed to transform the link into a line that appears free of undetected transmission errors. Widely used data link layer protocols are High-level Data Link Control (HDLC), Synchronous Data Link Control (SDLC), Serial Line Internet Protocol (SLIP), and the Point-to-Point Protocol (PPP). Some consider the ATM Adaptation Layer (AAL) as an upper sub-layer protocol of the data link layer.

Broadcast networks, such as satellites, must control access to the shared channel. This function is done within a sub-layer of the data link layer called the Medium Access Control (MAC) sub-layer. Protocols have been developed specifically for the MAC sub-layer, such as the ALOHA protocols and the Carrier Sense Multiple Access (CSMA) protocols. The above-mentioned protocols for the lower two layers of the OSI model help create the IEEE 802.xx or ISO 8802.x standards for LANs and WANs.

2.1.3 The Network Layer

The network layer is concerned with the transfer of data over multiple paths through the network to the proper destination. Its main function is to determine the destination of the data, route the packets through a proper path and make sure the resources exist to do this[3].

The Internet Protocol or IP is used as the protocol of choice for the network layer. It has many versions such as IPv4, IPv6, and the ISO-IP. Each version of IP varies slightly with respect to their functions. However, all versions of IP contain control protocols, such as Internet Control Message Protocol (ICMP), Address Resolution Protocol (ARP), Reverse ARP (RARP), and BOOTP, which help with the task of locating the IP source and destination addresses [4].

2.1.4 The Transport Layer

The transport layer must provide data transport from the source machine to the destination machine, by passing the data accepted from the source session layer through the network to the final destination. It acts as an end-to-end layer, independent of the physical network or networks it is running over, performing end-to-end transfer and flow control of data and ensuring it arrives error free at the proper destination machine. This is why it is considered the heart of the whole OSI model protocol hierarchy[2].

Two main protocols are used in this layer: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP is a reliable transport layer protocol that performs error and ordering correction. UDP is considered unreliable, only ensuring that messages reach the proper destination. Some also consider the adaptation layer (AAL) for ATM networks as a transport layer protocol of sorts[4].

2.1.5 The Session Layer

The session layer allows users working on different machines to create and establish sessions between them. Thus, it is responsible for setting up a communication channel between two users, for the duration of the complete network transaction. This allows the machines to send data back and forth. It also provides enhanced services that allow the machines to synchronize their dialogue and manage their data exchange. Some examples of these services are token management and remote login.

2.1.6 The Presentation Layer

The presentation layer is concerned with the actual syntax and shared semantics (representation) of the information transmitted. The data structures to be exchanged can be defined in an abstract way, and these abstract data types are then encoded using a standard (transfer syntax) that both machines can understand, such as ASCII or Unicode. The presentation layers agree on what standard should be used to ensure proper data transfer. ASN.1[1] is an example of a presentation layer protocol defined by ISO 8824 and 8825 standards.

2.1.7 The Application Layer

The application layer provides the user interface with a range of network-wide distributed information services. These include file transfer access and management, electronic mail, and other message interchange services. The layer contains the many different protocols that are needed to solve the problem of the many incompatible terminal types and different file systems that exist. Examples of such protocols are TELNET, FTP, SMTP, and SNMP for the TCP/IP suite, and VT, FTAM, MOTIS, CMISE for the OSI – ISO standards. Others include HTTP, MIME, DNS, MMS, JTM, and DTP[1].

2.2 TCP/IP SUITE

The OSI reference model follows the seven-layer design as described in 2.0. However, many computer scientists and network specialists follow a modified five layer OSI model equivalent, which is a hybrid OSI - TCP/IP reference model (Fig. 2-2)[2]. This is known as the TCP/IP suite of protocols.

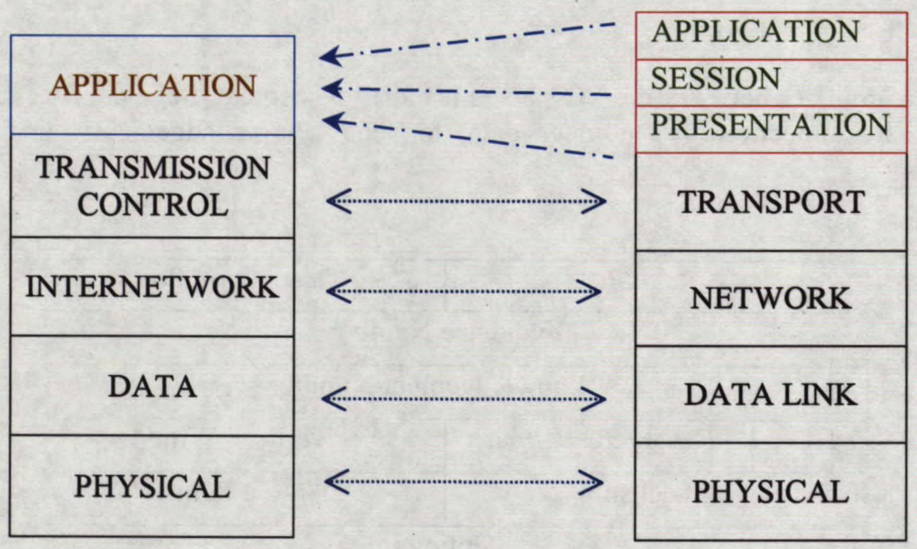


Figure 2: Hybrid OSI-TCP/IP reference model compared to the OSI model

The hybrid reference model's application layer is equivalent to the three upper levels of the OSI model. Thus, it contains such protocols as TELNET, FTP, SMTP, DNS, HTTP, and other presentation and session protocols such as the ISO 8822 standards[2]. The last four layers have a one-to-one correspondence.

Most multimedia/web applications today, run over TCP/IP. However, TCP behavior is highly sensitive to the delay and BER of satellite links. The following two chapters give an overview of the characteristics of satellite links and the problems with TCP over such links. Before discussing the specific problems of running TCP over a satellite link, a more detailed overview of TCP is needed.

2.3 USING TCP

TCP (Transmission Control Protocol) is the most widely used transport protocol designed to provide a reliable end-to-end byte stream service over an unreliable internetwork. TCP assures that data is delivered reliably, in sequence, and without duplication or loss. It is a full-duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction[5]. It provides the flow control that enables the receiver to regulate the rate at which the sender may transmit data. It also has congestion control mechanisms that let the sender adjust its own behavior according to the network conditions.

2.3.1 TCP Segment Format

The unit of transfer between the TCP peers is called a segment. Segments are exchanged to establish connections, to transfer data, to send acknowledgements, and to close connections.

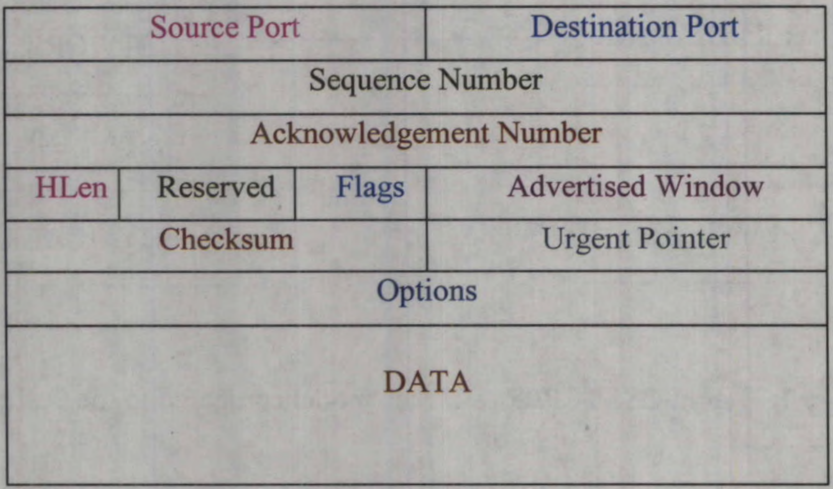


Figure 3: TCP segment format

Figure 3 displays the format of a segment. Each segment includes a header followed by data. Some segments may carry only an acknowledgement without any data. In the TCP header, fields *Source Port* and *Destination Port* contain the TCP port numbers that identify the application programs at the ends of the connection. The *Sequence Number* field contains the sequence number for the first octet of data transmitted in this segment. The *Acknowledgement Number* field specifies the sequence number of the next octet that the source application expects to receive. The *HLen* field contains an integer that specifies the length of the segment header measured in 32-bit multiples. The *Reserved* field, as its name implies, is reserved for future use.

There are six flags in the TCP header, and all fit in the six bits available to the *Flags* section. These are:

URG:	Indicates that this segment contains urgent data, which starts at the beginning of *Data* field of the segment. The *Urgent Pointer* field indicates the amount of Urgent Data bytes in the segment.
ACK:	Indicates that the *Acknowledgement field* is valid (this flag is set for all but the initial SYN segment).
PSH:	Indicates that data should be delivered promptly.
RST:	Indicates an error; also used to abort a session.
SYN:	Set to 1 during connection setup.
FIN:	Set to 1 during graceful close.

The *Advertised Window* field specifies how many octets of data the source application is prepared to accept. Finally, the *CheckSum* field is computed over the TCP header, the TCP data, and the pseudo header, which is made up of the Source IP Address, Destination IP Address, Protocol, and TCP Length fields from the IP header. Finally, the *Options* field enables the sending TCP entity to indicate the receiver the maximum number of octets in the user data field of a segment it is prepared to receive.

2.3.2 Establishing and Releasing a TCP Connection

The TCP connection procedure is called a three-way handshake because three messages (SYN, SYN + ACK, and ACK) are transmitted to set up the connection.

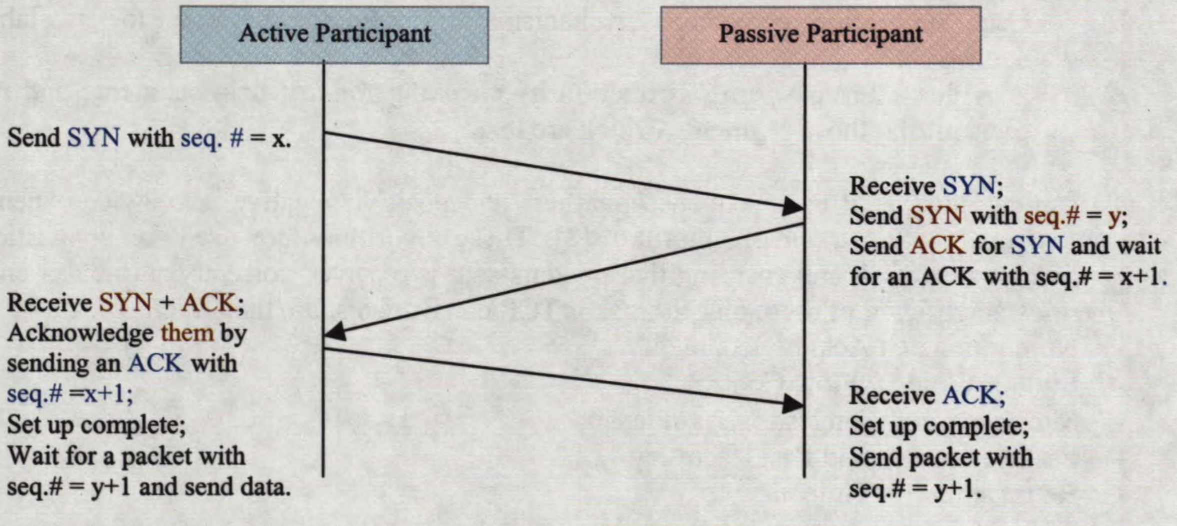

Active Participant
Passive Participant
Send SYN with seq. # = x.
Receive SYN;
Send SYN with seq.# = y;
Send ACK for SYN and wait for an ACK with seq.# = x+1.
Receive SYN + ACK;
Acknowledge them by sending an ACK with seq.# =x+1;
Set up complete;
Wait for a packet with seq.# = y+1 and send data.
Receive ACK;
Set up complete;
Send packet with seq.# = y+1.

Figure 4: TCP connection set up

Three important pieces of information are exchanged during three-way handshake. Each side notifies its partner of:

- The initial sequence number it will use for numbering outgoing data (field *Sequence Number*)
- The maximum buffer space available to receive data (placed in the *Advertised Window* field)
- The maximum amount of data that an incoming segment may carry (the maximum segment size MSS in field *Options*)

Data Transfer begins after the completion of the three-way handshake. It follows as a simple data transfer process where the connected pair updates their Advertised Window sizes, and sends Data and ACKs accordingly.

A connection request can be initially rejected, by responding with the RST flag set. Once a connection is established, one side may send a segment with the RST flag set to abort a connection if necessary. In this case, the transfer in both directions ceases immediately. If the connection proceeds without any abnormalities, when the application is completed, it will release or close the connection with a three-way handshake similar to that of initial connection set-up.

2.4 CORE TCP FUNCTIONS AND MECHANISMS

TCP performs three important functions, which turn a connection between two users within a network into a reliable data transmission path:

a) Sets up a transmission or receiving window, which allows the receiving TCP to control the flow of data being sent to it at any given moment,

b) Uses congestion avoidance mechanisms to probe the network for available bandwidth,

c) Provides a level of error correction by checking for lost data segments and re-transmitting those segments, which are lost.

To achieve this, TCP uses sequence numbers and positive/negative acknowledgements implemented within a set of algorithms[3]. These algorithms are used for congestion control, flow control, and ensuring that the data sent is received correctly at the user end. The most significant of these algorithms, or TCP mechanisms, are the:

- Numbering and Acknowledgments
- Sliding Window Flow Control
- Slow Start and Congestion Avoidance
- Fast Retransmit and Fast Recovery
- Retransmission Timeouts

This section describes the mechanisms that TCP uses to deliver data correctly, in order, and without loss or duplication.

2.4.1 Numbering and Acknowledgements

To make data delivery reliable, TCP employs numbering and acknowledgement (ACK) schemes. TCP's numbering scheme associates every octet of data sent on a TCP connection with a sequence number. A segment's TCP header contains the sequence number of the first octet of data in the data field of the segment.

The receiver is expected to acknowledge (ACK) received data. The acknowledgement number (ACK) field in a segment identifies the sequence number of the next octet that the receiver expects to receive, implicitly acknowledging all earlier sequence numbers from the sender. TCP's cumulative acknowledgement scheme can save bandwidth by using a single ACK to acknowledge more than one segment at a time.

The receiving TCP monitors incoming sequence numbers to ensure arriving data are in order and that no data is missing. Since ACKs occasionally are lost or late, duplicate segments may arrive at the receiver. The receiver can tell duplicated segments by the sequence numbers and just discard them.

2.4.2 Sliding Window Flow Control

In an Internet environment, having a mechanism for flow control is essential because there are machines of various speeds and sizes communicating through networks. The sliding window algorithm realizes TCP flow control. The receiver advertises how much data it is willing to accept in the advertised window field, and the sender must stay within this limit.

TCP on the sending side maintains a send buffer used to store data that has been sent but not yet acknowledged, as well as data that has been written by the sending application, but not transmitted. On the receiving side, TCP maintains a receive buffer. This buffer holds received data that the application process has not read.

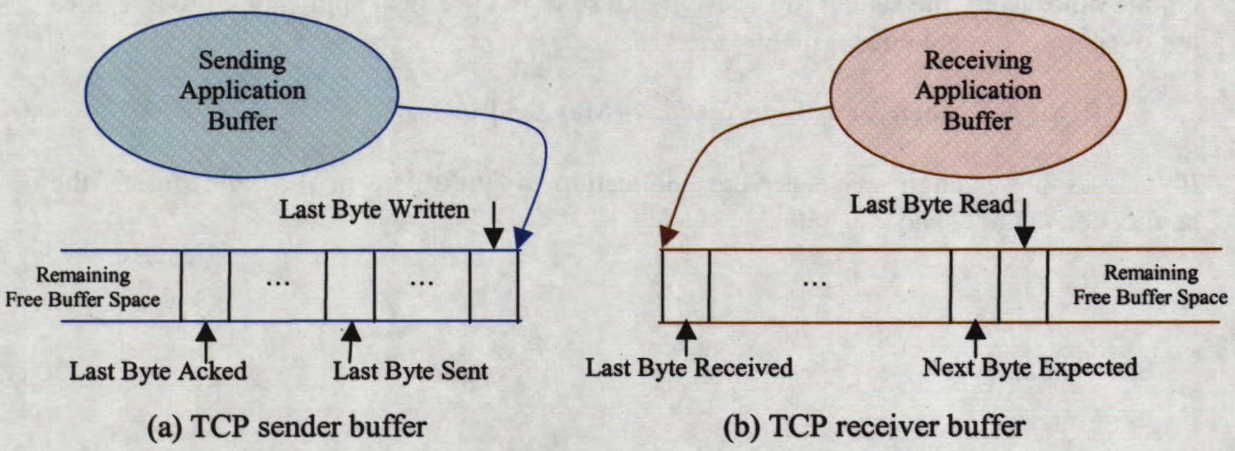


Figure 5: TCP buffer size manipulation

In the TCP sliding window algorithm, the size of the advertised window sets the amount of data that can be sent without waiting for acknowledgement from the receiver. Suppose that the size of the send buffer is MaxSendBuffer, and the size of the receive buffer is MaxRcvBuffer.

TCP on the receive side must keep:

Last Byte Rcvd – Last Byte Read <= MaxRcvBuffer

to avoid overflowing its buffer.

Therefore, it advertises a window size of:

Advertised Window = MaxRcvBuffer – (Last Byte Rcvd – Last Byte Read)

which represents the amount of free space remaining in the receive buffer. How the advertised window size changes, depends on how fast the data arrives and how fast local the application process is consuming data. If the local process is reading data just as fast as it arrives (causing Last Byte Read to be incremented at the same rate as Last Byte Rcvd), then the advertised window stays unchanged. However, if the receiving process falls behind, then the advertised window grows smaller with every segment that arrives, until it eventually goes to zero.

TCP on the send side must adhere to the advertised window it gets from the receiver. This means that at any given time, it must ensure that:

LastByteSent – LastByteAcked <= AdvertisedWindow

Therefore, the effective window that limits how much data the sender can send is:

EffectiveWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)

On the other hand, the sender must also make sure that the local application process does not overflow the send buffer, that is:

LastByteWritten – LastByteAcked <= MaxSendBuffer

If there is not enough space for the application to write data in the send buffer, the application cannot send any data.

2.4.3 Slow Start and Congestion Avoidance

While flow control prevents the sender from overrunning the capacity of the receiver, slow start and congestion avoidance prevent too much data from being injected into the network. TCP maintains a state variable for each connection, called the Congestion Window, which is used by the sender to limit how much data it can put into the network at any given time.

Slow Start defines a mechanism when starting traffic on a new connection or recovering from congestion indicated by a timeout event. Specifically, the source starts out by setting Congestion Window to one segment. For each segment that is ACKed successfully, the Congestion Window is increased by one segment. That is, the Congestion Window is doubled for every Round Trip Time (RTT) completed. When starting a new connection, if there is no loss of segments, the Congestion Window can keep increasing until it reaches the same size as the Advertised Window.

Additive Increase/Multiplicative Decrease algorithms are used in TCP to react to the network congestion and adjust the Congestion Window size accordingly. In the current Internet environment, it is rare that a packet is dropped because of an error during transmission. Therefore, TCP interprets timeouts as a sign of congestion and reduces the rate at which it is transmitting. Specifically, each time a timeout occurs, the sender sets Congestion Window to half of its previous value. This mechanism is called *multiplicative decrease*. When the congestion is relieved, the Slow Start mechanism is implemented. Once the Congestion Window reaches one half of its original size before congestion occurred, the *additive increase* mechanism is used beyond this point to linearly increase the Congestion Window size.

With the additive increase mechanism, if each packet sent out has been ACKed before the timeout, roughly one MSS (Maximum Segment Size) is added to the Congestion Window. The maximum unacknowledged amount of data is now the minimum of the Congestion Window and the Advertised Window. Thus, the Effective Window indicating how much data the sender can send is revised as follows:

Effective Window = min (Advertised Window, Congestion Window) – (LastByteSent – LastByteAcked)

2.4.4 Fast Retransmit / Fast Recovery

Fast Retransmit is a heuristic that sometimes triggers the retransmission of a dropped segment sooner than the regular timeout mechanism. Whenever the receiver gets an out-of-order segment, it sends an ACK that identifies the first octet of the missing data (the same ACK the receiver sent when it received the last in-order segment). After receiving three duplicate ACKs, TCP performs a retransmission of the lost segment indicated in the ACK, without waiting for the retransmission timer to expire. Fast retransmit is implemented in TCP Tahoe

When the fast retransmit mechanism signals congestion, the sender cuts the congestion window in half and resumes with additive increase, rather than going into the Slow Start. This mechanism is called *Fast Recovery*. Fast Retransmit/Fast Recovery (called FR/FR) is implemented in TCP Reno.

When multiple segments are lost in transmission of one Congestion Window (cwnd) size worth of data, only one of these segments can be resent using fast retransmit and the rest of the dropped segments usually have to wait for the Retransmit Time Out (RTO), which will trigger the costly Slow Start mechanism[6].

2.4.5 Partial and Selective Acknowledgments

An experimental loss recovery mechanism is proposed to detect multiple losses in the same window using *Partial Acknowledgements*. A Partial ACK covers some new data, but not all data in flight when a particular loss event starts. If there are multiple losses, when TCP resend the first lost segment indicated by several duplicated ACKs, the receiver will send back an ACK indicating loss of another segment which is then the first lost segment. More than one lost segment per window can be found and retransmitted by this means. Partial ACKs are implemented in TCP New Reno.

Partial ACKs avoids timeout but cannot result in a recovery faster than one loss per RTT. The sender needs to wait for the ACK of the retransmission to discover the next loss. Some studies show that in some cases (i.e. multiple-losses of segments) relying on the RTO timer may be more efficient than simply using partial ACKs to retransmit all lost segments [1].

To solve this problem, TCP provides another acknowledgement strategy called *Selective Acknowledgements* (SACKs). SACKs allow TCP receivers to inform TCP senders exactly which segments have arrived so that the sender can resend lost segments quickly and avoid unnecessary retransmissions.

The SACK option field contains a number of SACK blocks, where each SACK block reports a non-contiguous set of data that has been received and queued, as illustrated in Figure 6 [37]. SACKs enables the receiver to inform the sender exactly which segments have arrived so that the sender can resend lost segments quickly and avoid unnecessary retransmissions.

Segment Sent	Acknowledgment Received						
	Ack	First Block		Second Block		Third Block	
3000	3500						
3500 (lost)	--						
4000	3500	4000	4500				
4500 (lost)							
5000	3500	5000	5500	4000	4500		
5500(lost)							
6000	3500	6000	6500	5000	5500	4000	4500

Figure 6: SACK option field example

The sender is also able to transmit segments (retransmissions or new segments) during the recovery phase, therefore sustaining the ACK clock. SACKs generally allow TCP to recover from multiple segment losses in a window of data within one RTT of loss detection[16].

In one SACK implementation[6], the sender enters Fast Recovery when the data sender receives three duplicate acknowledgments. The sender retransmits a packet and cuts the congestion window in half. During fast recovery, SACK mechanism maintains a variable called *pipe* that represents the estimated number of packets outstanding in the path. *Pipe* is initialized to the value of the cwnd size and adjusted as following:

-- For each duplicate ACK received, decrease by one segment;
-- For each partial ACK received, decrease by two segments;
-- For each segment sent, increase by one segment

The sender only sends new or retransmitted data when the estimated number of packets in the path is less than the congestion window, that is pipe < cwnd. The sender exits fast recovery when a recovery acknowledgment is received acknowledging all data that was outstanding when fast recovery was entered. When a retransmitted packet is lost, the SACK mechanism detects the drop with a RTO, retransmits the dropped packet, and then implements Slow Start [6].

SACK leads to a significant improvement in performance when multiple losses appear in the same window. Studies show that TCP with SACK options performs very well in long-delay environments with moderate losses (under 50 percent of the window size)[18]. The partial ACK is needed only in the absence of SACKs. However, the recovery of the SACK mechanism is still sensitive to the loss of ACKs[28]. In addition, the TCP header is limited to 64 bytes, thus the SACK can indicate at most three lost blocks[24] per window.

2.4.6 Timeout and Retransmission

TCP retransmits each segment if an ACK is not received for the segment before a specific TCP clock times out. This time out is called the RTO (Retransmit Time Out). TCP sets the RTO clock as a function of the RTT (Round Trip Time). In an internetwork environment, TCP must accommodate both the vast differences of delays between different pairs of machines and the variance of delay between the same pair.

TCP uses an adaptive retransmission mechanism that monitors delays on each connection and adjusts its timeout parameter accordingly. Every time TCP sends a data segment, it measures a Sample RTT:

$$\text{SampleRTT} = T_{ack} - T_{send}$$

Where:
T_{send} is the time when TCP sends a data segment,
T_{ack} is the time when TCP receives an ACK to that segment.

Algorithms of Jacobson and Karn[2][10] enable TCP to obtain a reasonable estimation of retransmission timeout from the sample RTT. Jacobson's algorithm can deal with the high variance of delay, and Karn's algorithm is used whenever there is a retransmission.

Jacobson's Algorithm calculate the timeout as follows:

$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$

$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta \times \text{Difference})$$

$$\text{Deviation} = \text{Deviation} + \delta\ (|\text{Difference}| - \text{Deviation})$$

$$\text{TimeOut} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}$$

Where:
δ is a fraction between 0 and 1,
μ is typically set to 1 and ϕ is set to 4.
Thus, when the variance is small, TimeOut is close to Estimated RTT. A large variance causes the Deviation term to dominate the calculation.

When a segment is retransmitted and then an ACK arrives at the sender, it is impossible to determine if this ACK should be associated with the first or the second transmission of the segment for the purpose of measuring the sample RTT. Karn's algorithm provides a simple solution: TCP only measures Sample RTT for segments that have been sent only once. Whenever TCP retransmits a segment, it stops taking samples of the RTT, and sets the next timeout value to be twice the last timeout for subsequent packets until a valid sample RTT is obtained. This may cause TCP to wait too long to determine a loss and delay the retransmission in the next RTT.

2.5 TCP EXTENSIONS FOR HIGH PERFORMANCE

2.5.1 TCP Window Scale Option

TCP performance depends not only upon the transfer rate itself. The window size and RTT play as important role on TCP performance. A helpful quantity that can help predict performance is the product of the transmission rate and the round-trip delay time[14], called the ***Bandwidth-Delay Product*** (BDP). The BDP is equivalent to the recommended amount of data that the pipe or channel can accommodate at any given time. Thus:

BDP = Bandwidth * RTT Delay

The sender should send as much data as the BDP specifies to keep the pipe full. However, the maximum amount of unacknowledged data a TCP sender can transmit into a network is limited by the receiver's Advertised Window size. This amount cannot exceed the throughput. TCP throughput is given by the following formula [25]:

Throughput = Window Size / RTT

The maximum window size for standard TCP Reno is 65,535 bytes (~ 525 kilobits). For a satellite link with a RTT of around 0.6 seconds, the throughput cannot exceed 875 kbps. Thus, no matter how large the channel bandwidth is, the throughput will not exceed 875 kbps. For example, a DS-1 speed satellite channel using TCP Reno has a BDP of 925 kilobits. Thus using TCP Reno over such a channel will give bandwidth utilization of less than 57%. TCP cannot fully utilize the available bandwidth because of a small window[14]. Hence, TCP performance problems arise when the BDP is much larger than the Advertised Window. Thus, if the maximum window size were increased to 925kbits (~ 115,000 bytes), then the throughput would reach the speed of a DS-1.

To achieve the optimal throughput, the window size should be at least the same as the Bandwidth-Delay Product. RFC 1323 [22] defines a set of window scaling options that enable TCP to operate over large BDP networks including satellite links. This set involves an option that defines a *scaling factor* for the Advertised Window, which supports large window sizes up to 2^{30} bits. The window scaling option specifies how many bits each side should left-shift the Advertised Window field in the TCP header before using its contents to compute an Effective Window size[10].

2.5.2 Round-Trip Time Measurement (RTTM)

RTT estimates are essentially important to TCP's reliability. Poor RTT estimation may delay necessary retransmission or cause unnecessary retransmissions. Many TCP implementations base their RTT measurements on a sample of only one packet per window. While this yields an adequate approximation to the RTT for small windows, it results in an unacceptably poor RTT estimate for a network with large BDP[14]. In addition, according to Karn's algorithm, each time TCP retransmits a segment, it stops taking samples of the RTT and sets the next timeout value to be twice the last timeout value for subsequent packets. Simply doubling the timeout value will cause TCP to wait too long to determine a loss and delay the retransmission during frequent loss period.

The timestamp option provides a solution to these problems. Each time when TCP is about to send a segment, it reads the system clock and puts the time in the timestamp option of the segment. The receiver echoes these timestamps back in the ACK segments. Then the sender can obtain an accurate RTT measurement for every ACK segment from the current system clock and the echoed timestamp. This mechanism is called Round-Trip Time Measurement (RTTM).

2.5.3 Protection Against Wrapped Sequence Numbers (PAWS)

TCP allocates its sequence numbers from a 32-bit 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. However, TCP usually assumes that the maximum time a datagram can live in the network is only two minutes. Thus, when TCP sends a byte in an IP datagram, the sequence number of that byte cannot be reused in two minutes. A 32-bit sequence space spread over two minutes gives a maximum data rate of only 286 Mb/s[8]. Table 2.2 shows how long it takes the 32-bit sequence number to wrap around on networks with various bandwidths[10].

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2G)	28 seconds

Figure 7: Table of times until 32-bit sequence number space wraps around

TCP PAWS extensions prevent TCP's 32-bit Sequence Number field wrapping around too fast on a high-speed network. PAWS uses the same TCP Timestamps option as the RTT mechanism. The TCP receiver decides whether to accept a segment based on a 64-bit identifier that has the Sequence Number field in the lower-order 32 bits and the timestamp in the high-order 32 bits[10]. This extension is sufficient for link speeds of between 8 Gb/s and 8 Tb/s[8].

2.5.4 The Heart of TCP

The reliable transmission and flow control of data are at the heart of the TCP layer. Within these functions, problems arise when the assumptions inherent to the above algorithms are violated. Such violations occur over high bandwidth-delay and bit error rate (BER) links, such as satellite networks, and the above enhancements are not adequate to prevent these problems. The next chapter examines the characteristics of satellite links and the violations to the TCP algorithms due to these characteristics.

3. TCP OVER SATELLITE LINKS

A satellite network (or link) is comprised of a series of terminals on the surface of the Earth, called ground stations, which transmit and receive microwave signals, and one or more satellites. A satellite acts as an overhead relay or repeater for communications between two geographically remote locations[16]. As illustrated by the sample configuration in Figure 8, a router (within Building 1) is connected to a ground station (shown as a satellite dish) that takes the incoming traffic, converts it into a microwave signal, and transmits it on a specific frequency up to the satellite. The satellite receives the signal, amplifies it, and then transmits over the downlink on a different frequency. The ground station then receives the signal, converts it to a terrestrial link format, and passes it on to a router in Building 2.

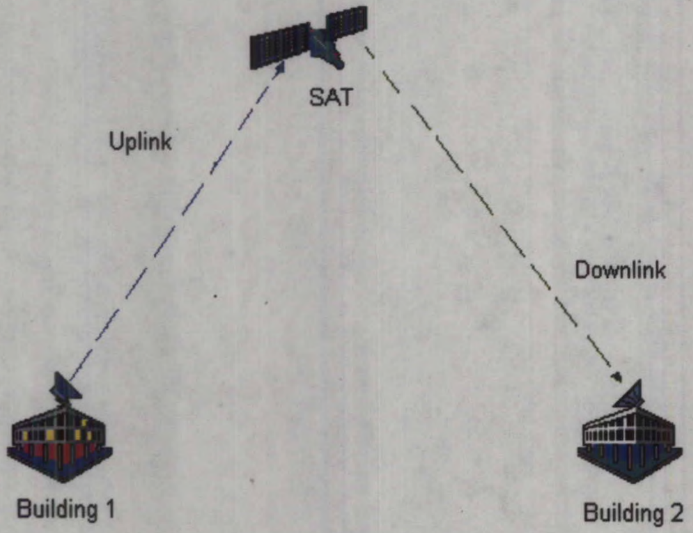


Figure 8: Simple satellite network

Satellite channels have several characteristics that differ from most terrestrial channels. These characteristics may have negative effect on TCP's utilization of satellite bandwidth[7]. The most important of these is the large distance between the satellite orbits and the receiving Earth stations. These distances can range from 800 km for Low Earth Orbit (LEO) satellites through and up to 36,000 km for Geosynchronous orbit (GEO) satellites. Thus, the distance the signal must travel is large, creating a delay, while also being more susceptible to poor atmospheric conditions that can lead to corruption of the data[17]. These issues, as well as the problems they cause for TCP performance, will be discussed in this section.

3.1 TCP PERFORMANCE OVER SATELLITE

Three parameters that provide a measure of performance for TCP are the maximum possible throughput or data rate (Max d.r.), the Bandwidth-Delay Product (BDP), and the actual throughput or goodput[9]. These parameters are related in the following equations valid for satellite links using TCP[11]:

$$BDP = BW \times RTT \qquad \text{(eq.1)}$$

And

$$Max\ d.r. = SWS\ /\ RTT \qquad \text{(eq.2)}$$

Where,

BW: is the link's bandwidth or pipeline capacity

RTT: is the Round Trip Time

SWS: is the Sliding Window Size or receiver buffer size

Thus, RTT and BW must be known so the SWS and other TCP attributes can be modified to enhance performance. Hence, to be able to achieve improved performance, the sliding window (receiver buffer) must be at least as large as the BDP. From here the maximum possible data rate can be found. However, maximizing buffer size and all other TCP parameters also depends on the operating system kernel. If the appropriate hardware is not available, the buffer size cannot be increased. A lot of reconfiguring would be needed.

Even if all parameters can be modified easily to the recommended value, the RTT, the BER and the Maximum Transmission Unit (MTU) of the underlying protocols effect the actual throughput or goodput[12]. The equation below shows this.

$$Goodput = (1 - OH) \times (1 - L) \times \frac{W}{1 + 2 \times C \times I\ /\ MTU} \times BW \qquad \textbf{(eq.3)}$$

Where,

OH: Overhead from all sources (as a fraction)

L: Probability of error occurring within a TCP window

W: TCP window size (in MTUs)

C: Flow-controlled data rate (Max d.r. in Kbps or Mbps)

I: One way delay from all sources (distance + processing)

BW: Pipeline capacity in Kbps or Mbps

MTU: The IP MTU, measured in bits, and varies depending on the data link protocol IP runs over.

Thus the type of link, the delays along the link and the BER will define the goodput. If the BER is large, many retransmissions will be needed, thus reducing the goodput further, but most importantly, increasing the time it takes to correctly send an application from one end to another[13]. Thus, the quality of real time applications can be severely degraded, while delays for file transfers, accessing web pages, and remote login can be large.

Some of the most important properties of satellite links that can affect the performance of protocols running over them are: long delay paths, large bandwidth-delay products, increased channel errors, channel asymmetry, variable Round Trip Times, and Intermittent Connectivity.

3.2 SATELLITE LINK PROPERTIES AND THEIR AFFECT ON TCP

3.2.1 Long Delay Paths

Latency in a satellite environment is generally higher than that in a terrestrial environment. Satellite network delays are influenced by several factors. The main factor is the orbit type. In most LEO systems, one-way propagation delays are 20~25ms. The propagation delays increase to 110~150 ms for medium earth orbit systems (MEO) and go up to 250~280 ms for GEO systems [6][18]. The RTT may also be increased by some other factors in satellite networks such as on-board processing, signal handoff and, in the future, intersatellite links. These long delays hurt interactive applications such as telnet and remote login, as well as multimedia applications running over TCP, since the flow control and congestion control algorithms of TCP are affected. The affect of a satellite environment on the behaviour of TCP algorithms is discussed in the following sections.

3.2.1.1 Acknowledgments

The TCP acknowledgment is essential to the reliable delivery of data. Due to the long propagation delay of some satellite links, it may take a long time for a TCP sender to determine whether a packet has been successfully received at the final destination. The sender also uses the acknowledgment as network feedback for rate adjustment for slow start, congestion avoidance, and loss recovery.

The large propagation times will delay the execution of these functions and affect TCP throughput. The Delayed ACK suggested in RFC 1122 could make the situation worse, especially for Slow Start. Since the data sender increases the size of cwnd based on the number of arriving ACKs, reducing the number of ACKs slows the cwnd growth rate[27]. According to[13], under no-loss conditions, with a window size of 10 MB and a RTT of about 500 ms, it takes TCP about 10 seconds to fill the OC-3 pipe when delayed ACK is used. Compare this to a time 0.5 seconds to fill an OC-3 terrestrial WAN pipe, or 2 milliseconds to fill a LAN pipe. Possible high bit error rates will cause a loss of ACK packets, which in turn will lead to unnecessary retransmissions and diminished throughput.

3.2.1.2 Congestion Avoidance

TCP uses linear increase to slowly probe the network for additional capacity after Slow Start. Whenever losses occur, TCP congestion control mechanisms halve the transmission rate with both the drawbacks of slowing-down the sending rate and long-lasting loss recovery occurring. However, losses are inevitable since this linear increase mechanism continuously increases its cwnd without any mechanisms to predict the incipient of congestion. In fact, TCP always needs to create losses to find the available bandwidth of the connection. This is inefficient for use over a long delay satellite link because a large amount of time is required for loss recovery and for reaching the optimum transmission rate after rate reduction.

Another problem arises with the linear increase: during congestion avoidance, in the absence of loss, the TCP sender adds approximately one segment to its congestion window during each RTT. Several researchers have observed that this policy leads to unfair sharing of bandwidth when multiple connections with different RTTs traverse the same bottleneck link, with the long RTT connections obtaining only a small fraction of the share of the bandwidth[16].

3.2.2 Large Bandwidth-Delay Product

The product of bandwidth and delay determines how much unacknowledged data a TCP sender should transmit into the network to fully utilize the capacity of the link. The delay in this equation is the RTT, and the bandwidth is the maximum bandwidth of the slowest link in the path. Satellite channels usually use large bandwidths (larger than many of the terrestrial networks); factor in the long propagation delays, and the bandwidth-delay product can be quite large, especially for GEO systems. This means that the sender and receiver must be able to handle larger amounts of data in flight[6].

3.2.2.1 TCP Window size

As we discussed in 2.5.1, larger windows enable TCP to better utilize the available bandwidth; therefore, the TCP large window extension is recommended for use in a satellite environment[6].

The sliding window algorithm allows multiple segments to be sent in a "window" from sender to receiver. The bandwidth-delay product is equivalent to the amount of data that the network can at most accommodate, at any given time. The sender should send as much data as possible to keep the pipe full. Thus, the window size should be at least the same as the bandwidth-delay product. The larger the window, the more data can be in transmission, and the capacity of the data link can be maintained at or near its maximum capacity.

The current maximum TCP window size is 64K. This means a TCP sender can only fill a channel with a bandwidth-delay product less than 64K. If the round trip time is 0.5 second, the link has a transmission rate less than 1 Mbps. In other words, if the bandwidth is larger than 1 Mbps, the link cannot be fully utilized. RFC 1323 defines a set of window scaling options available to TCP implementations that operate over large bandwidth-delay networks including satellite links.

While enlarging the window would compensate for the large bandwidth-delay product in satellite networks, it will also provide some complications, such as increased variation of the measured RTT and increased probability of multiple losses within a single window because of the increased number of segments per window, and increased likelihood of data bursts [14][32]. As a result, the SACK option is recommended in a satellite environment because it can help accelerate the lengthy recovery procedure. In addition, the RTTM using the timestamp option is recommended in obtaining a more precise measurement of RTT when using the large window option[16].

3.2.2.2 Slow Start

The slow-start period is one of the most important factors for a TCP connection's performance over satellite links. Slow Start is triggered at the connection establishment phase, and after a retransmission timeout or possibly after an extended idle period.

During Slow Start, the Congestion Window (cwnd) is increased by one segment for each ACK received. Under ideal conditions, this would yield a doubling of cwnd per RTT. Over GEO satellites, the increase in the transmission rate will be much slower due to the long propagation delay, because the increase of the transmission window relies on the acknowledgment. It usually takes several seconds to reach maximum throughput [25].

When lost segments trigger congestion avoidance, the resulting throughput decrease can continue for several minutes. Slow Start is particularly inefficient for transfers that are short compared to the BDP of the network. In this case, TCP may never reach the full rate available.

Large bandwidth-delay products of satellite links make slow-start threshold (ssthresh) estimation critical. A too small ssthresh causes the majority of cwnd growth to be linear (very slow); a too large ssthresh increases the possibility of multiple-losses within a window. When Slow Start resumes, the new ssthresh (old_cwnd/2) will cause the bulk of future cwnd growth to be linear[32]. In addition, because the cwnd is roughly doubled every RTT during Slow Start, the likelihood of a data burst is increased when using a large window size[20].

3.2.3 Increased Transmission Errors

Signal strength attenuates proportionally to the square of the distance traveled and this distance is large for a satellite link. Thus, the signal becomes weak before reaching its destination. This results in a low signal-to-noise ratio and a high bit error rate. Some frequencies are particularly susceptible to atmospheric effects such as rain attenuation. In a satellite environment, the raw BER usually ranges from 10^{-3} to 10^{-6}. With the aid of Forward Error Correction (FEC) schemes the BER ranges from the 10^{-7} to 10^{-9} range[18]. This is much higher than in terrestrial environments where the typical BER is better than 10^{-10} [15].

The use of FEC coding on a satellite link is recommended in RFC 2488 [25] to reduce the link BER[6]. However, FEC does not come without cost. FEC requires additional hardware and uses some of the available bandwidth. It can add delay and timing jitter due to the increased complexity. In addition, there are some situations where FEC cannot solve the problem such as noise caused by rain fade[6].

3.2.3.1 Slow Start

The high BER of satellite links makes slow start even more inefficient. If there is a loss due to corruption rather than congestion, Slow Start will make the cwnd return to its minimum value and start over again. This action will cause the slow-start to prematurely terminate and may have a significant impact on throughput for the remainder of the connection.

3.2.3.2 RTO based on RTT Estimation

High BER and increased delay variances in satellite links can adversely affect TCP timer mechanisms. One such example is the RTT estimation algorithm needed to set the RTO of TCP. Poor RTT estimation may trigger unnecessary retransmissions, or delay necessary retransmission. According to Karn's algorithm, each time TCP retransmits a segment, it stops taking samples of the RTT, and sets the next timeout value to be twice the last timeout for subsequent packets. Simply doubling the timeout value will cause TCP to wait too long to determine a loss and delay the retransmission during frequent loss periods. This variability affects the timers directly, resulting in false timeouts and unnecessary retransmissions, yielding incorrect window sizes, and thus reducing the overall bandwidth efficiency.

3.2.3.3 Congestion vs. Corruption

Differentiating between congestion and corruption is particularly important in a high BER environment because the actions that TCP should take in the two cases are entirely different. In the case of congestion, TCP sender should immediately reduce its congestion window to avoid making the congestion worse, and retransmit the lost segment at the appropriate time. In the case of corruption, TCP should merely retransmit the damaged segment as soon as its loss is detected. There is no need for TCP to reduce its congestion window; however, there is no specific mechanism defined in TCP to differentiate between congestion losses and link corruption losses. It always interprets segment loss as a sign of congestion and reduces the rate at which it is transmitting. This greatly degrades performance when the loss is caused by corruption. Further research is needed into mechanisms that allow TCP to respond to corruption loss in an appropriate manner.

3.2.4 Channel Asymmetry

Satellite communication networks are often constructed asymmetrically due to the expense of the equipment used to send data to satellites and various engineering tradeoffs. Some studies show that it is not uncommon for the ratio of downlink to uplink capacity to approach 100 or more [15][19]. Another situation involving channel asymmetry is sending all outgoing traffic over a slow terrestrial link such as a dial-up modem channel, and receiving incoming traffic via the satellite channel.

TCP has not been designed for asymmetric networks. If a satellite is transmitting data over a high capacity channel, the returning acknowledgements may overrun the reverse channel. For example, if the data sender uses 1500 byte segments, and the receiver generates 40 byte acknowledgments (Ipv4, TCP without options), the reverse link will congest with ACKs for asymmetries of more than 75:1 if delayed ACKs are used, and 37:1 if every segment is acknowledged[16]. The congestion of ACKs increases the RTT, which in turn increases end-to-end delay. Current congestion control mechanisms are aimed at controlling the flow of data segments, but do not regulate the flow of ACKs.

3.2.5 Variable Round Trip Delays

Since the coverage of LEO systems is relatively small, satellite constellations with dynamic inter-satellite routing are required to provide continuous coverage over larger regions. This factor increases end-to-end delay variability in satellite communications since the delay will change depending on the number of satellites, the propagation distances of the paths, constellation topology, inter-satellite routing algorithms, and so on. Onboard processing overhead and buffering can also increase the delay variability. Variable delay can cause difficulty in RTT estimation, resulting in false timeout or unnecessary retransmission. In the case of large windows, TCP's RTTM mechanism is recommended to mitigate the delay variance in a satellite environment.

3.2.6 Intermittent Connectivity

For satellites, connectivity on a given communication link is often intermittent. Contact may be interrupted for a number of reasons, including ground station handoffs, changing network topology, antenna obscurations, weather, and orbit dynamics.

The intermittent connectivity of satellite links causes serious problems for TCP. If the TCP sender does not receive the expected acknowledgments, it will invoke congestion control algorithms and repeatedly retransmit and back off its retransmission timer. If the maximum retransmission threshold is reached before connectivity is restored, the connection will be aborted. Drastically delayed acknowledgments may cause the same problem on TCP as the intermittent connectivity feature described above.

4. IMPROVING SATELLITE LINK PERFORMANCE WITHOUT MODIFYING TCP

The IETF and other network specialists have specified the problems of using TCP over GEO satellite links and outlined possible solutions. In this section, we examine some of the possible solutions that can be used to enhance satellite link performance without enhancing TCP mechanisms and modifying TCP behaviour.

4.1 APPLICATION LEVEL

Some of TCP's shortages when used over long-delay networks can be avoided if Internet applications use TCP more effectively.

4.1.1 Multiple TCP Connections

This method has been used at the application level to overcome TCP's inefficiencies in a satellite environment. It uses multiple TCP connections to transfer a given file. This method accelerates the growth of the aggregate cwnd, but increases the aggressiveness of the transfer and hence increases the losses in the network. This may seem to be mitigated by the smaller aggregate cwnd decrease during congestion avoidance. However, this smaller decrease will defeat the purpose of the congestion avoidance mechanism if congestion, not corruption is the culprit of the losses in the network. An adaptive mechanism has been proposed to change the number of connections as a function of network congestion [16][28].

4.1.2 Persistent TCP Connections

Persistent TCP connections are another solution at the application level, which can accelerate the transfer of Web pages. A typical Web page consists of many small objects. It usually takes tens of seconds to fetch such a page over a GEO satellite if independent TCP connections are used to fetch every object in a page[34]. By using persistent TCP connections, the client establishes a persistent connection and asks the server to send all the objects on it. Only the first object suffers from the long slow start phase, and the remaining objects are transferred at a high rate[28].

4.1.3 Application Layer Proxies

Some application protocols employ many unnecessary round trips, lots of headers and/or inefficient encoding which may have a significant impact on performance when using on a long-delay link. By using application layer proxies in an intermediate node, this unnecessary overhead can be reduced and the performance of both the application protocol and TCP can be improved[36]. Application-specific proxies can use domain knowledge to match network constraints and reduce the effect of latency[34]. Such proxies are widely used in today's Internet for web caching and relaying Mail Transfer Agents.

4.2 NETWORK LEVEL

4.2.1 TCP Spoofing

TCP spoofing is a technique used to mitigate the problems associated with high latency. In this approach, TCP acknowledgments are manipulated in an intermediate gateway without waiting for the actual acknowledgment from the receiver [15][34], as shown in Figure 9. The gateway will then take responsibility for delivering that data using an optimized transport protocol.

This transport protocol is tuned to quickly increase its transmission rate without the need for a long slow start phase[15]. Once arriving at the output end of this link, another TCP connection may need to be used to transmit the packets to the destination. Because packets have already been acknowledged, any loss between the input of the link and the destination must be locally retransmitted on behalf the source. In addition, ACKs from the receiver must be discarded silently to not confuse the source.

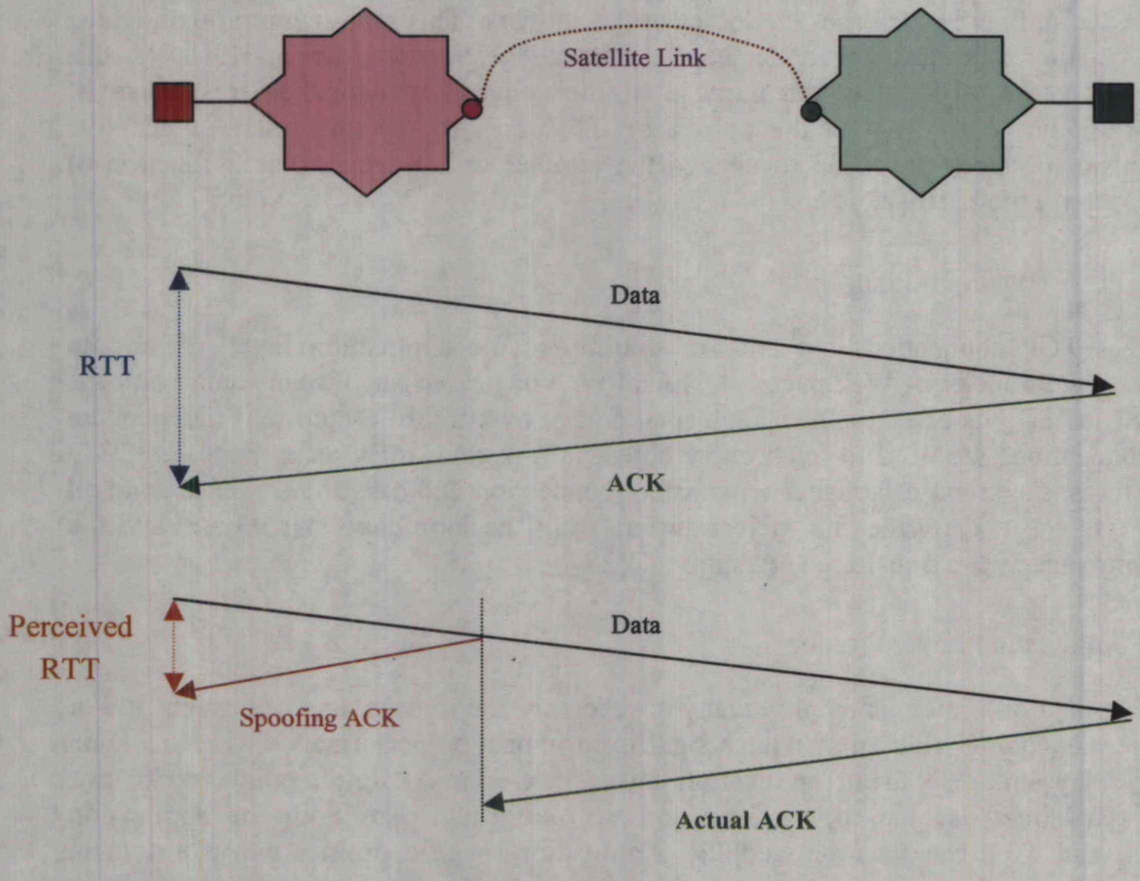


Figure 9: TCP spoofing

The main gain of TCP spoofing in performance comes from not using SS on the long delay link. The window increases more quickly because of the rapid feedback, which improves performance. However, TCP spoofing has a number of problems. It breaks the end-to-end semantics of TCP since a packet is acknowledged before reaching its end destination. It also introduces a heavy overload on network routers. Furthermore, it does not work when encryption is accomplished at the IP layer[28].

4.2.2 TCP Splitting

TCP splitting is an approach that uses a gateway at the periphery of the satellite network to convert TCP traffic into an intermediate protocol that is well suited for the satellite environment. On the other end of the satellite link, the protocol will be converted back to TCP [15][28][34]. TCP splitting breaks an end-to-end TCP connection into two or three segments, as shown in Figure 10.

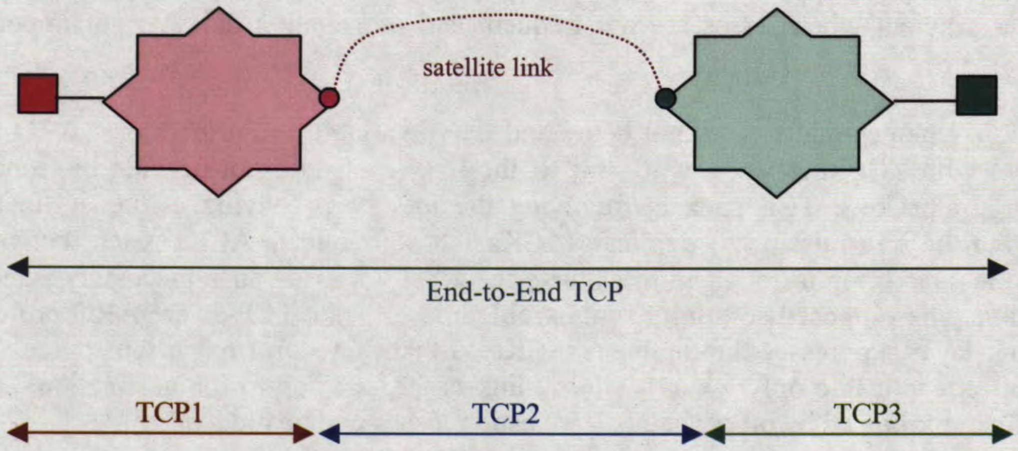


Figure 10: TCP splitting

TCP splitting allows optimization of the connection between the two intermediate gateways over a satellite link. In general, it tends to address a mismatch in TCP capabilities between two end systems[36]. The protocol converter may simply convert between different versions of TCP. Incoming TCP connections that may not be operating with protocol stacks and/or applications appropriate for use over satellite links will be converted to a connection that takes full advantage of the TCP extensions suitable for satellite links. Another approach to TCP splitting is to use a protocol other than TCP on the satellite segment of the network. This will largely enhance TCP performance, but it must be able to look at the TCP headers. This means it will not work with encryption techniques that encrypt the transport header unless the gateway is a trusted system[15]. It is believed that TCP splitting works well in cases where applications actively participate in TCP connection management[8].

There are two possible ways to handle intermittent connection in TCP splitting. The first one is to hide the link disconnection in the intermediate gateways. The intermediate gateways employ a modified TCP version, which retains the state and all unacknowledged data segments during the period of disconnection and then performs local recovery when the link is restored. Another way to handle this problem is that the sender-side gateway retains the last ACK before losing the connection, so that it can shut down the TCP sender's window by sending the last ACK with a window set to zero. Thus, the TCP sender will go into persist mode, sending periodic probe packets without repeated time-out and retransmissions[36].

4.3 LINK LAYER

Link layer approaches are proposed to reduce the BER of the satellite channel. One well-known mechanism is Automatic Repeat Request (ARQ). The link layer fragments user datagrams into smaller link-layer frames and ensures the reliable delivery of these frames using a variety of approaches, such as stop-and-wait, go-back-N, or selective-repeat. ARQ is efficient when losses are not frequent and propagation delay is not important [18][28].

All ARQ schemes add to channel delay and delay variability. Furthermore, ARQ may interfere with TCP mechanisms[16][28]. If the link layer does not provide in-sequence delivery of packets, TCP packets following the loss keep arriving at the destination, triggering the transmission of duplicate ACKs. These duplicate ACKs reach the source while the link layer is retransmitting the packet. This causes an unnecessary window reduction. The proposed solution to this problem is to use a TCP-aware ARQ protocol. The link layer suppresses the duplicate ACKs so that they don't reach the source. This solution is applicable only when the lossy link is the last hop to the destination. If the lossy link is followed by other routers, congestion losses will be hidden[28].

4.4 OTHER MECHANISMS

4.4.1 TCP Agent

This kind of solution tries to improve link quality by retransmitting packets via a TCP agent located in the router at the input of the lossy link. The TCP agent keeps a copy of every data packet. It discards this copy when it sees the ACK of the packet, and it retransmits the packet on behalf of the source when it detects a loss. This technique has been proposed for terrestrial wireless networks where the delay is not so important as to require the use of FEC[28].

In fact, this solution is no other than link-level recovery implemented at the TCP level. Similar to a link-level solution, because the TCP agent hides all losses, congestion losses must not occur between the TCP agent and the destination. Otherwise, without a signal of the congestion, the TCP sender will continue to increase the sending rate and make the congestion worse[28].

4.4.2 Path MTU Discovery

The use of Path MTU (Maximum Transmission Unit) Discovery is recommended in RFC 2488 to allow TCP to use the largest possible MTU over the satellite channel. The sender transmits a packet with a certain size appropriate for the local network and sets the IP "Don't Fragment" (DF) bit. If the packet is too large, a router will return an ICMP message to the TCP sender indicating the size of the largest packet that can be forwarded by the router[6].

Large packets reduce the packet overhead by sending more data bytes per overhead byte. In addition, the TCP's congestion window is increased on a segment basis, rather than a byte-by-byte basis; therefore, larger segments enable TCP senders to increase the congestion window more rapidly, in terms of bytes, compared to smaller segments.

The disadvantage of Path MTU Discovery is that it may spend a large amount of time determining the maximum allowable packet size on the network path between the sender and receiver. Satellite delays can aggravate this problem. Storing the MTU values can reduce latencies for future connections in relatively static topologies[18].

5. TCP ENHANCEMENTS FOR SATELLITE LINKS

In this section, we will discuss some suggested solutions to the problems we discussed before, using TCP enhancements. Some solutions may need the support of network modification.

5.1 TCP FOR TRANSACTIONS (T/TCP)

Many TCP applications involve only simple communications between the client and the server: a client sends a request to a server and the server replies. Under standard TCP, even a small transmission involving a single request segment and a reply must undergo TCP's three-way handshake prier to data transmission. This connection setup requires 1.5 (RTTs) for the active participant and one RTT for the passive participant. This is especially inefficient on a long-delay satellite path.

An experimental extension of TCP for Transactions (T/TCP) suggested in RFC 1644 [23] provides a solution to this problem. After the first connection between a pair of hosts is established, T/TCP is able to bypass the three-way handshake, allowing the data sender to begin transmitting data in the first segment along with the SYN. With T/TCP, a short transmission can be done by only two messages and an ACK, as showed in Figure 11.

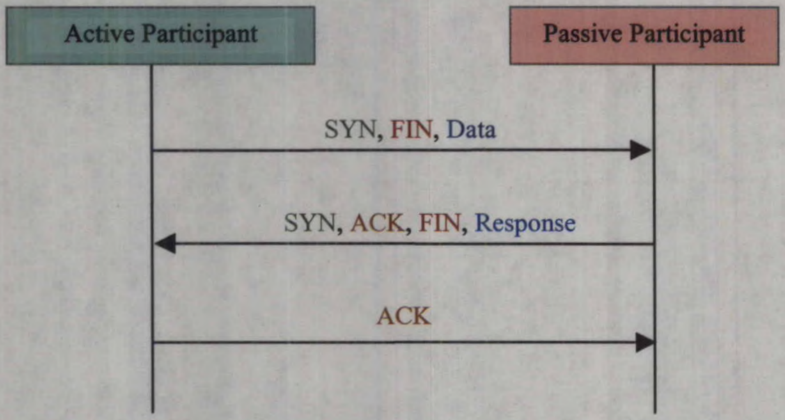


Figure 11: T/TCP, establishing and closing a connection

5.2 SLOW START

Slow Start is a safeguard against transmitting an inappropriate amount of data into the network when starting up or recovering from congestion. However, it can also waste available network capacity in satellite networks due to the long delay and the large bandwidth product. The following are some proposals that have been suggested to make slow start more efficient to operate over satellite links.

5.2.1 Larger Initial Window

One method that will reduce the amount of time required by slow start is to increase the initial value of cwnd. An experimental TCP extension outlined in RFC2414 allows the initial size of cwnd to be increased from one segment to that defined by equation (4):

$$\text{Min}(4{*}\text{MSS}, \max(2{*}\text{MSS}, 4380\ \text{byte})) \qquad \text{(eq.4)}$$

By increasing the initial value of cwnd, more packets are sent during the fist RTT, which will trigger more ACKs, allowing the cwnd to increase more rapidly. In addition, it avoids the ACK delay timer interval when delayed ACKs are used. Studies have shown that an initial value of four segments improves startup times significantly without a noticeable increase in the packet loss[18]. In RFC 2581[26], a TCP connection is allowed to use an initial cwnd of up to two segments. This change is highly recommended for satellite networks[16][36].

5.2.2 Delayed ACKs after Slow Start

As discussed in section 2, TCP increases the cwnd based on the number of arriving ACKs. Moreover, since the delayed ACKs recommended in RFC 1122 reduce the number of ACKs by roughly half, the rate of growth of the cwnd is reduced. One proposed solution to this problem is to use delayed ACKs only after the slow start phase. This provides more ACKs while TCP is aggressively increasing the congestion window, and less ACKs while TCP is in steady state. Studies show that simulations using Delayed ACKs After Slow Start (DAASS) improve transfer time when compared to a receiver that always generates delayed ACKs. However, DAASS also slightly increases the loss rate due to the increased rate of cwnd growth[30].

5.2.3 Byte Counting

Byte counting is another solution to delayed ACKs during slow start. It can also benefit asymmetric networks where the ACKs are heavily delayed.

Using standard ACK counting, the congestion window is increased by one segment for each ACK received during slow start. While using byte counting, the congestion window increase is based on the number of acknowledged bytes covered by the incoming ACK. This makes the increase relative to the amount of data transmitted, rather than depending on the ACK interval used by the receiver.

There are two forms of byte counting, unlimited byte counting (UBC) and limited byte counting (LBC). UBC simply uses the number of previously unacknowledged bytes to increase the congestion window each time an ACK arrives. LBC limits the amount of cwnd increase to two segments. Both UBC and LBC may cause burst of data since they allow cwnd to increase faster. LBC prevents large line-rate bursts when compared to UBC, and therefore offers better performance. RFC 2581 allows TCP to use byte counting to increase cwnd during congestion avoidance, but not during slow start[36].

5.2.4 Reducing Burstiness in Slow Start

In slow start phase, for each segment that is ACKed successfully, the congestion window is increased by one segment. That is, the congestion window is doubled in every RTT. The double-numbered segments are usually transferred in a bursty manner. The study in [20]shows TCP is likely to produce losses during slow start due to data burst in a network with long delay and large bandwidth. This kind of loss may not be a signal of network congestion, but rather of the overflow of limited buffer size of routers. It will cause unnecessary retransmission, reduction of transmission rate, and an underestimate of available bandwidth of the link. To avoid bursty data transfer, controlling the interval of data transfer is required. The proposed solution is called smooth slow start.

Smooth slow start uses a timer interrupt routine to control data transfer. When sender receives an ACK, it decides whether or not to expand the interval of next transmission. If so, it only sends one segment when receiving an ACK and set a flag for later cwnd increment. The timer interrupt routine will check the flag periodically. If the flag is set, it will increase the congestion window and invoke the segment transfer routine to transmit the delayed segment.

Figure 12 shows an example of smooth slow start. This example compares smooth slow start with current slow start implementation. As seen in Figure 12, the number of transmitted segments is doubled in every 500 ms. Compared with normal Slow Start, smooth slow start reduces burst segment transfer. Furthermore, despite the expanded interval of segment transfer, the number of transmitted segments in one RTT is the same as normal slow start.

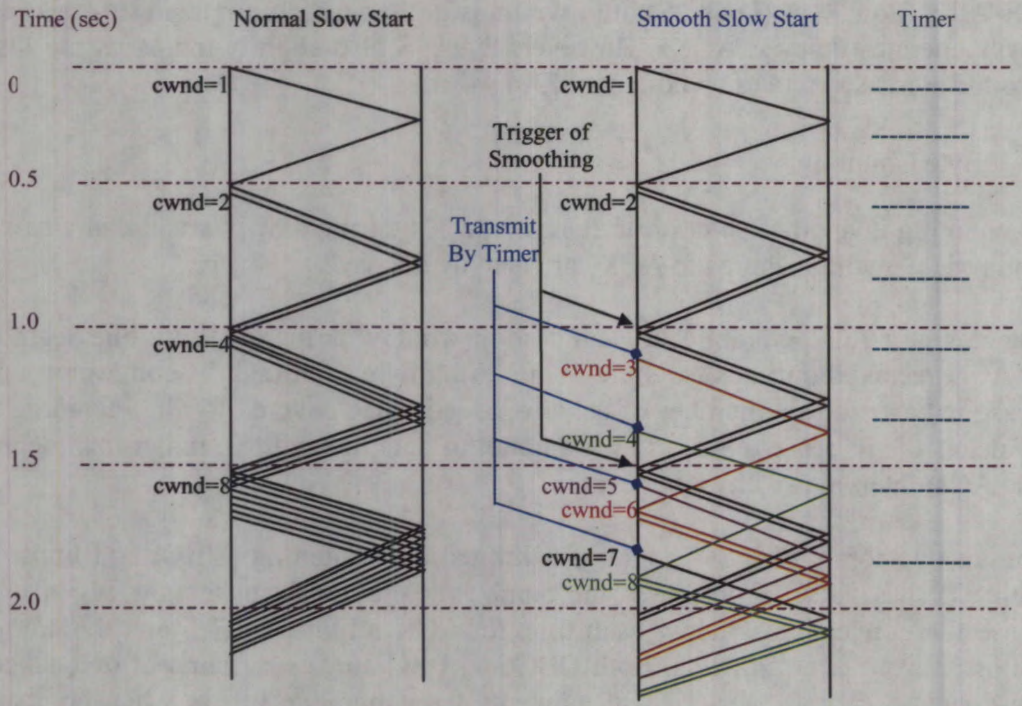


Figure 12: Example of smooth slow start

5.2.5 Terminating Slow Start at the Right Time

Slow start is terminated when TCP detects losses, or when the size of cwnd reaches the ssthresh. When TCP initially starts up, without the knowledge of the capacity of the link, TCP roughly doubles the size of cwnd every RTT. This leads very quickly to congestion. When congestion is detected, TCP sets the ssthresh to half of the congestion window. In other words, TCP needs to create losses to find the available bandwidth of the connection. Due to the long delay in satellite networks, the recovery can be very time-consuming. Therefore, terminating slow start at the right time is very useful. There are two proposals for the right time to terminate slow start.

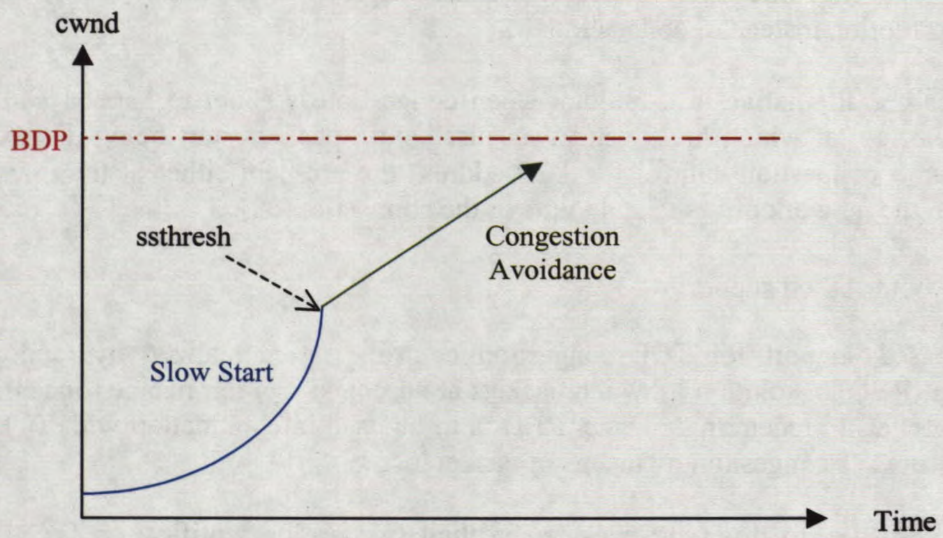


Figure 13: Terminating slow start

One proposal is to use the packet-pair algorithm and the measured RTT to determine a more appropriate value for ssthresh. The algorithm observes the spacing between the first few returning ACKs to determine the bandwidth of the bottleneck link. Together with the measured RTT, the BDP is determined and ssthresh is set to this value[16]. The challenge of this approach is that obtaining an accurate estimate of the available bandwidth in a dynamic network is not easy.

Another proposal for terminating slow start properly is TCP Vegas's slow start mechanism. To be able to detect and avoid congestion during slow start, Vegas allows exponential growth only every other RTT. In between, the congestion window stays fixed so a valid comparison of the expected and actual rates can be made. When the actual rate falls below the expected rate by a predefined value, Vegas changes from slow start mode to congestion avoidance mode[31]. However, increasing window size every other RTT may not be suitable for satellite links. Alternative mechanisms need to be used to predict network congestion during slow start.

5.3 CONGESTION AVOIDANCE

TCP in its congestion avoidance phase repeatedly increases the load it imposes on the network until congestion occurs, and then it backs off from this point. TCP congestion avoidance implements linear increase and multiplicative decrease algorithms to avoid incurring congestion and to recover from congestion loss. These algorithms have a negative impact on TCP performance in a satellite environment. The linear increase slowly probes the network for additional capacity, which is especially inefficient over long-delay satellite channels because of the large amount of time required for the sender to obtain feedback from the receiver. On the other side, the multiplicative decrease halves the transmission rate whenever loss occurs. It is too conservative when used with a slow increase algorithm like linear increase. It causes more waste of bandwidth if the loss is due to corruption instead of congestion.

An appealing alternative is to predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded[10]. Solutions to congestion control for TCP address the problem either at the intermediate routers in the network or at the endpoints of the connection[29].

5.3.1 Router-based support

Router-based support for TCP congestion control can be realized by random early detection (RED), a solution in which packets are dropped in a fair manner once the router buffer reaches a predetermined size. RED is to be used in conjunction with TCP, which currently detects congestion by means of packet loss.

As an alternative to dropping packets, Explicit Congestion Notification (ECN) allows routers to inform TCP senders about the imminent congestion so that the source will slow down its sending rate. There are two major forms of ECN: backward ECN (BECN) and forward ECN (FECN). A router employing BECN transmits messages directly to the data originator informing it of congestion. The arrival of a BECN signal may or may not mean that a TCP data segment has been dropped, but it is a clear indication that the TCP sender should reduce its sending rate. FECN routers mark data segments with a special tag when congestion is imminent, but forward the data segment. The data receiver then echoes the congestion information back to the sender in the ACK packet.

ECN may be part of the solution that helps TCP react properly on congestion loss and corruption loss. If all the sources, receivers, and routers are compliant, congestion losses will considerably decrease. However, on a satellite link, the main losses are mostly caused by problems other than congestion. Given that non-congestion losses require only retransmission without window reduction, the disappearance of congestion losses may lead to the definition at the source of a new congestion control algorithm, which reacts less severely to losses. However, if some non-compliant routers cannot provide the source with the required information, TCP still needs to consider losses as signs of congestion and reduce its window accordingly[28].

5.3.2 Source-based Congestion Avoidance

Source-based congestion avoidance can detect the incipient stages of congestion from the end hosts before losses occur. The general idea of these techniques is to watch for some signs from the network that a router's queue is building up and that congestion will occur if nothing is done about it.

One approach takes advantage of the fact that there is a measurable increase in the RTT as packet queues build up in the network's routers. For every two RTTs, this approach checks to see if the current RTT is greater than the average of the minimum and maximum RTTs seen so far. If it is, then the algorithm decreases the congestion window by one-eighth; otherwise, the window size is increased as usual.

Another proposal does something similar. However, the decision as to whether or not to change the current window size is based on the changes to both the RTT and the window size. The window is adjusted once every two round-trip delays based on the product:

$$(\text{CurrentWindow} - \text{OldWindow}) * (\text{CurrentRTT} - \text{OldRTT})$$

If the result is positive, the source decreases the window size by one-eighth. If the result is equal to or less than zero, the source increases the window size by one maximum packet length. Note that the window changes during every adjustment. Hence, it oscillates around its optimal point.

In addition to the change of RTT, another sign of network congestion is the flattening of the sending rate. Rate control congestion approaches change the current window size by examining the changes in the throughput. One proposal is to increase the window size by one packet every RTT and compare the throughput achieved to the throughput when the window was one packet smaller. If the difference is less than one-half the throughput achieved when only one packet was in transit – as was the case at the beginning of the connection – the algorithm decreases the window by one packet. An alternative solution calculates throughputs differently, and instead of looking for a change in the throughput slope, it compares the measured throughput rate with an expected throughput rate. This solution is implemented as TCP Vegas. The results in terrestrial networks using TCP Vegas indicate over 30 percent improvement in throughput and much fewer losses than TCP Reno does. However, more studies are necessary for tuning TCP Vegas in satellite networks[18].

A problem with rate-control and relying upon RTT estimates is that variations of congestion along the reverse path cannot be identified and separated from events on the forward path. Therefore, an increase in RTT due to reverse-path congestion or even link asymmetry will affect the performance and accuracy of these algorithms[29]. A proposal in [29] makes use of an additional timestamp returned from the receiver to estimate the level of queuing in the bottleneck link of a connection. The receiver attaches a timestamp in every ACK packet that specifies the arrival time of the packet at the destination. The sender then calculates the relative delay, which is defined as following:

$$D^F_{j,i} = R_{j,i} - S_{j,i}$$

Where

$R_{j,i}$:	is the time interval between the receipt of packet j and i,
$S_{j,i}$:	is the time interval between the transmission of packet j and i,
$D^F_{j,i}$:	represents the change in forward delay experienced by packet j with respect to packet i.

From the relative delay measurement the sender can determine whether congestion is increasing or decreasing in either the forward or the reverse path of the connection.

5.3.3 Controlling ACK Congestion

There are two proposals addressing the ACK congestion: ACK Congestion Control and ACK filtering.

ACK Congestion Control (ACC) extends the concept of flow control for data segments to acknowledgment segments. When detecting ACK congestion, the receiver dynamically adjusts the rate of acknowledgments using the multiplicative decrease and additive increase as in general congestion control mechanisms. There are two ways to detect ACK congestion: Explicit Congestion Notification and Relative Delay mechanism discussed in 5.3.1 and 5.3.2.

In ACK Filtering (AF), the bottleneck router in the low speed link will scan the queue for redundant ACKs for the same connection, i.e. ACKs that acknowledge portions of the window, which are included in the most recent acknowledgement. All of these "earlier" ACKs are removed from the queue and discarded.

Both of the two mechanisms may cause unwanted side effects, such as increased likelihood of segment bursts from the data sender, and the decrease of sender's cwnd growth rate even if the data link is non-congested. ACK spacing is suggested to reduce the burstiness by smoothing out the flow of ACKs. Finally, ACK Reconstruction (AR) is recommended when using AF. However, AR requires sharing and storage of TCP state information in the exit router, and more research is needed before implementing AR.

5.3.4 Reducing Unfairness of Linear Increase

Another problem with the linear increase occurs during congestion avoidance. In the absence of loss, the TCP sender adds approximately one segment to its congestion window during each RTT. Several researches have observed that this policy leads to unfair sharing of bandwidth when multiple connections with different RTTs traverse the same bottleneck link, with the long RTT connections obtaining only a small fraction of their fair share of the bandwidth. The solution to the unfairness of linear increase at the TCP sender is to change the window increase policy.

The "Constant-Rate" increase policy attempts to equalize the rate at which TCP senders increase their sending rate during congestion avoidance. However, the proper selection of a constant for the increase rate is an issue. This policy may be difficult to incrementally deploy in an operational network[16].

The "Increase-by-K" policy can be selectively used by long RTT connections. It simply changes the slope of the linear increase, with connection over a given RTT threshold adding "K" segments to the congestion window every RTT, instead of one. This policy, when used with small values of "K", may be successful in reducing the unfairness while keeping the link utilization high when a small number of connections share a bottleneck link[16].

5.4 LOSS RECOVERY

The large BDP causes more losses for TCP in satellite environment. Because of the conservative congestion control mechanisms and the long delayed feedback from the receiver, TCP takes a long time to recover from congestion loss. Selective ACK (SACK) is recommended in RFC 2488 to help TCP survive multiple segment losses within a single window without incurring a retransmission timeout. Fast recovery with SACK is more efficient than fast recovery with or without partial ACK scheme[25]. However, SACK is generally viewed as a method to address data recovery. It has not been widely investigated to control congestion while recovering from dropped segments[35]. Fast recovery with SACK is unable to prevent excessive timeouts under extreme losses[18]. One proposed solution, called Forward Acknowledgment (FACK), works in conjunction with the SACK option to add more precise data transmission control during the recovery phase.

FACK uses additional information provided by the SACK option to keep an explicit measure of the total number of bytes of data outstanding in the network. It introduces two new state variables, send_fack and retran_data. The send_fack is to reflect the forward-most data successfully received by the receiver. The value of send_fack is equal to the highest sequence number known to have been received plus one. The retran_data is used to hold the number of outstanding retransmitted data segments in the network. For convenience, a variable awnd is used to denote the estimate of the actual quantity of data outstanding in the network, and send_nxt denotes the sequence number TCP is about to send next. When all unacknowledged segments have left the network:

$$awnd = send\_nxt - send\_fack$$

During recovery, retransmitted data must also be included in the computation of awnd:

$$awnd = send\_nxt - send\_fack + retran\_data$$

The values of these variables are adjusted as following:
-- If TCP retransmits old data, it will increase retran_data;
-- If TCP sends new data, it advances send_nxt;
-- When receiving an ACK, it decreases retran_data or advances send_fack.

If the sender receives an ACK which results in a send_fack beyond the value of send_nxt at the time a segment was retransmitted, the sender know that the segment which was retransmitted has been lost[35]. In another words, FACK detects the loss of a retransmission by the receipt of a segment that was sent later than the retransmitted segment while the retransmitted one is unacknowledged.

The current implementation of FACK is FACK with rate halving[33]. The rate-halving algorithm adjusts the congestion window by spacing transmissions at the rate of one data segment per two segments acknowledged over the entire recovery period, thereby sustaining TCP's self-clocking and avoiding transmission burst. The FACK algorithm is triggered after receiving three duplicate SACK blocks. The missing segment indicated in SACK is retransmitted. The connection will perform rate halving for one RTT after the retransmission. For every two ACKs received during recovery, it checks for any hole that equals or exceeds the retransmit threshold and retransmits that segment. If no segment exceeds the retransmit threshold, new data can be sent if the cwnd allows[37].

The FACK mechanism separates the recovery algorithm from the retransmission algorithm, providing a simple and direct way to use SACK to improve congestion control. In addition, FACK causes less data burst than fast recovery with SACK and is more robust against heavy losses[18]. Studies showed the performance of FACK is much closer to the theoretical maximum for TCP than either TCP Reno or fast recovery with SACK extensions. Although more studies are needed for using FACK over noisy and long-delay satellite links, it is expected to provide good performance gains [18] [16].

5.5 DETECTING CORRUPTION LOSS

5.5.1 Explicit Corruption Notification

This approach uses a new "corruption experienced" ICMP error message generated by routers that detect corruption. These messages are sent in the forward direction, toward the packet's destination, rather than in the reverse direction as is done with ICMP Source Quench Messages. Each TCP receiver that gets this information must forward it to its respective sender. The TCP sender then assumes that packet loss is due to corruption rather than congestion for two round trip times (RTT) or until it receives additional link state information. However, in shared networks, ignoring segment loss for two RTTs may aggravate congestion by making TCP unresponsive[16].

5.5.2 Detecting Corruption Loss in TCP

In reference [21], a non-congestion packet loss detection algorithm is proposed. This introduces a feedback mechanism from the receiver to inform the transmitter of packet losses that are not due to congestion.

The algorithm distinguishes packet loss due to congestion from a loss due to link errors. If a router within the network drops a packet, a number of consecutive packets will usually be dropped; if a packet is lost due to random link errors, the probability of losing the next packet is independent of previous losses and depends only on the link BER. In this case, the next packet will arrive without a relative delay.

This algorithm at the receiver waits for a calculated period after a packet loss is detected, before it decides whether to send an indication to the transmitter. This is called the back-off timeout (BTO). The BTO is calculated in a similar way to the TCP RTO.

The errors on a link may result in the loss of a number of consecutive packets; however most satellite links may be accurately modeled by a random packet loss model, and therefore there is only a low probability of two or more consecutive packets being corrupted. β is used to denote the number of consecutive packet losses, an indication of congestion. A value from 2 to 4 for β is recommended.

On reception of an out-of-order segment, the receiver checks whether the number of segments that have been lost is less than β, and the following segments arrive soon with an interarrival time less than BTO. If this happens, the receiver will start a BTO timer. When this timer expires without reception of the missing segments, the receiver sends a non-congestion, packet loss indicator in the TCP header back to the transmitter. The transmitter then retransmits the lost packets without reducing the sending rate.

5.6 HANDLING INTERMITTENT CONNECTIVITY

Rate-based pacing (RBP) is a technique used to maintain an intermittent connection. In the absence of incoming ACKs, the data sender temporarily paces TCP segments at a given rate to restart the ACK clock. Upon receipt of the first ACK, pacing stops and normal TCP ACK clocking resumes[16].

A more specific solution is proposed in[19]. The mechanism for identification of the onset of a link outage is link dependent. In general, a link outage may be identified at the ground station by loss of carrier lock or the received signal strength falling below a threshold. Once the ground station (or spacecraft) detects the link outage, it sends a link-outage ICMP message to any host on its own side of the served link from which it receives traffic. The ICMP message is triggered by incoming traffic. It contains the TCP header of the packet that caused the message to be generated. The sender's response to a link outage signal is to enter persist mode, sending periodic probe packets. During this period, TCP does not repeatedly time-out, retransmit, and back off the retransmission timer. Instead, it suspends its timers and ceases transmitting, except for the occasional probe packets. TCP exits persist state when it receives a link-restored ICMP message from the ground station, or when one of the probes is acknowledged.

5.7 SHARED TCP STATE INFORMATION AMONG SIMILAR CONNECTIONS

TCP includes a variety of parameters, many of which are set to initial values that can severely affect the performance of TCP connections traversing satellite links. Various suggestions have been made to change these initial conditions in an effort to support TCP over satellite links. However, it is difficult to select any single set of parameters which is effective for all environments.

An alternative solution to attempting to select these parameters appropriately is sharing state information across TCP connections and using this information when initializing a new connection. Sharing TCP state information can automatically tune TCP to the surrounding environments and coordinate multiple TCP connections sharing a satellite link. For example, if all connections to a subnet result in extended congestion window of 1 megabyte, it is probably more efficient to start new connections with this value, than to rediscover it using slow start that may cost dozens of round-trip times. However, several problems need to be addressed before using this approach, such as what information to share, with whom to share, how to share it, and how to age shared information[16]. At this time, CRC is investigating TCP state sharing and testing various algorithms and possible solutions with the help of the OPNET simulation tool.

6. CONCLUSIONS

TCP has several problems when running over satellite links. Large bandwidth-delay product and high bit error rate are the two major factors that affect most TCP mechanisms thus creating problems over satellite links.

Solutions to some of these problems are still not clear. Some problems have suggested solutions that have not been fully proven and tested, while other solutions that have become part of IETF standards are not yet widely implemented. For example, all TCP versions use the sliding window and slow start algorithms, but only the newly modified versions such as TCP Reno use the TCP mechanisms such as window scaling and larger initial window size described earlier.

The goal of all modifications and enhancements is to improve bandwidth efficiency and application response time, without negatively affecting the end-to-end reliability offered by TCP. For this reason, solutions such as TCP splitting and TCP spoofing are difficult to implement if one wishes to maintain end-to-end transport reliability. The coupling of congestion avoidance and data corruption also poses difficulties in maintaining end-to-end reliability.

More modifications and extensions to these enhancements are being examined and tested, but it will be some time before they are reliable enough to become standards. One such modification is sharing TCP state information, which can improve bandwidth efficiency while maintaining end-to-end reliability. Such modifications and extensions will play an important roll in using TCP over satellite networks.

7. RECOMMENDATIONS

In this document, many possible algorithms which can solve various problems that TCP connections encounter over satellite where examined. The enhancements recommended by the authors are increasing the initial advertised window size, window scaling such that the BDP is satisfied, and the SACK mechanism. These enhancements should be implemented and be part of all standard TCP stacks for connections over satellite links.

Finally, one algorithm that must also be investigated is the sharing of state information across TCP connections and using this information when initializing a new connection. This algorithm can improve bandwidth efficiency while ensuring the end-to-end reliability of TCP. The above mechanisms will have a profound effect on connections using TCP over long delay paths, and will improve the bandwidth efficiency of satellite networks.

8. REFERENCES

[1] Halsall, Fred, Data Communications, Computer Networks and Open Systems, Fourth Edition, New York: Addison-Wesley, 1996.

[2] Tanenbaum, Andrew S., Computer Networks, Third Edition, New Jersey: Prentice Hall, 1996.

[3] Comer, Douglas E., Internetworking with TCP/IP volume I, Third Edition, U.S.A.: Prentice-Hall, 1995.

[4] Black, Ulysses D., TCP/IP and Related Protocols, U.S.A.: McGraw-Hill, 1992.

[5] Feit, Sid., TCP/IP, Architecture, Protocols, and Implementation with IPV6, McGraw-Hill, 1996

[6] Fall, Kevin, and Floyd, Sally, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", ACM SIGCOMM '97, Computer Communication Review pg.5-21.

[7] Muhonen, John, and Durst, Robert C., "Performance of Transport Protocols over Satellite Communication Links", the MITRE Corporation, IEEE Communications, January 1998.

[8] Partridge, C., Shepard, T., "TCP/IP Performance over Satellite Links", IEEE Network September/October 1997.

[9] Allman, M., Hayes, C., Kruse, H., Ostermann, S., " TCP performance over Satellite links", Fifth International Conference on Telecommunication Systems, March 1997.

[10] Peterson, Larry L., Davie, Bruce S., "Computer Networks: A System Approach", Second Edition, Morgan Kaufman Publishers Inc., 1999

[11] Fair, C. E., "TCP Performance over ACTS", National Center for Atmospheric Research, Boulder Colorado, 1995. http://www.ntis.gov

[12] Farserotu, J., and Tu A., "TCP/IP Over Low Rate ATM-SATCOM Links", Shape Technical Centre, IEEE Communications, August 1996.

[13] Charalambous, P. Charalambos et al., "Experimental and Simulation Performance Results of TCP/IP over High-Speed ATM over ACTS", Kansas University Lawrence Dept. of Electrical Engineering and Computer Science, 1997.

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

[15] Stadler, J. Scott, and Gelman, Jay, "Performance Enhancement for TCP/IP on a Satellite Channel", MIT Lincoln Laboratory, IEEE Communications, January 1998.

[16] Metz, C., "TCP over Satellite, the Final Frontier", IEEE Internet Computing, January-February 1999, http://computer.org/internet.

[17] Farserotu, J., and Prasad, R., "A Survey of Future Broadband Multimedia Satellite Systems, Issues and Trends", IEEE Communication Magazine, June 2000.

[18] Ghani, N., Dixit, S., "TCP/IP Enhancements for Satellite Networks", IEEE Communications Magazine, July 1999.

[19] Durst, Robert C., Miller, Gregory J., Travis, Eric J., "TCP Extensions for Space Communications", Wireless Networks 3, 1997.

[20] Nishida, Yosh., "Smooth Slow-Start: Refining TCP slow-start for Large-Bandwidth with Long-Delay networks", IEEE 1998.

[21] Samaraweera, N., Faihurst, G., "Explicit loss indication and accurate RTO estimation for TCP error recovery using satellite links", IEE proc-Commun., Vol.144, No.1, February 1997.

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

[23] "TCP Extensions for Transactions", RFC 1644.

[24] Mathis, M., Mahdavi, J., Floyd, S., Romanow, A., "TCP Selective Acknowledgement Options", RFC 2018.

[25] Allman, M., Glover, D., Sanchez, L., "Enhancing TCP over Satellite Channels using Standard Mechanisms", RFC 2488, January 1999.

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

[27] Allman, M., et al., "Ongoing TCP Research Related to Satellites", RFC 2760, February 2000.

[28] Barakat, Chadi, Altman, Eitan, and Dabbous, Walid, "On TCP Performance in a Heterogeneous Network: A survey", IEEE Communications Magazine, January 2000.

[29] Parsa, Christina, and Garcia-Luna-Aceves, J.J., "Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media", 1999 IEEE.

[30] "A Survey of Congestion Control Techniques and Data Link Protocols in Satellite Networks", International Journal of Satellite Communications, 1995.

[31] Brakmo, Lawrence S., and Peterson, Larry L., "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet", IEEE JSAC, Oct. 1995.

[32] Allman, M., Durst, Robert C., and Travis, Eric, "Issues Related to TCP Performance in a Satellite Environment and Discussion of Possible Protocol-Oriented Mitigations", http://tcpsat.lerc.nasa.gov/tcpsat/meetings.html

[33] Mathis, M., Semke, J., Mahdavi, J., and Lahey, K., "The Rate-Halving Algorithm for TCP Congestion Control", June 1999, http://www.psc.edu/networking/ftp/papers/draft-ratehalving.txt

[34] Zhang, Yongguang, De Lucia, Dante, Ryu, Bo, and Dao, Son K., "Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential", http://www.isoc.org/inet97/proceedings/F5/F5_1.HTM

[35] Mathis, Matthew, and Mahdavi, Jamshid, "Forward Acknowledgment: Refining TCP Congestion Control", SIGCOMM Symposium on Communication Architectures and Protocols, August 1996.

[36] Border, J., Kojo, M., Griner, J., Montenegro, G., "Performance Enhancing Proxies", Internet Draft draf-ietf-pilc-pep-03.txt, June 25, 1999.

[37] Hayes, Christopher, "Analyzing the Performance of New TCP Extensions over Satellite Links", Master Thesis, August 1997.