Interactions Between TCP and Link Layer Protocols on Mobile Satellite Links

by

Edward A. Faulkner IV

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author ....
Department of Electrical Engineering and Computer Science
September 9, 2004

Certified by ....
John V. Guttag
Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Interactions Between TCP and Link Layer Protocols on Mobile Satellite Links

by

Edward A. Faulkner IV

Submitted to the Department of Electrical Engineering and Computer Science
on September 9, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

High speed satellite-based data networks are highly desirable for a variety of mobile applications. In order to inter-operate with existing networks, satellite-based systems must support TCP/IP traffic. However, TCP performs poorly on land-mobile satellite channels because of large propagation delays and frequent correlated packet loss.

The use of a link layer ARQ protocol can improve TCP performance by insulating TCP from data loss. However, the interactions between TCP and link layer protocols are not well understood. This thesis describes an emulation-based study of a mobile satellite link that explores the interactions between TCP and link layer ARQ.

Thesis Supervisor: Andrew Worthen
Title: Technical Staff, MIT Lincoln Lab

Thesis Supervisor: John V. Guttag
Title: Professor

Acknowledgments

Thanks to Andrew Worthen, Jeff Schodorf, and John Choi at Lincoln Laboratory for their constant technical and professional guidance, to Professor John Guttag for his scholarly guidance, and to my parents, who first inspired my curiosity.

Dedicated to my wife Eileen, for her patience and support.

Contents

Chapter 1

Introduction

1.1 Why TCP Over Satellite?

The demand for fast connections to wide-area networks continues to grow. Huge investments have been made in wired network infrastructure, such as fiber optic cables. However, most of the world is still far from a convenient data port. The problem is most acute in sparsely populated areas. Satellites provide an attractive alternative, because a single geosynchronous satellite can provide coverage to a vast area, over 1/3 of the Earth's surface.

For military applications, satellites have additional advantages. Satellites are relatively secure, since there is no terrestrial infrastructure to defend. They can provide connectivity to highly mobile terminals, and can be prepositioned before operations begin, enabling rapid deployment.

In order for satellite networks to inter-operate with existing networks (i.e., The Internet), they must support TCP traffic. Transmission Control Protocol (TCP) is ubiquitous. It underlies most Internet technologies, including the world wide web, email, and FTP. Replacing TCP would be infeasible, because of the massive installed base of hardware and software, and because a wholly new protocol could conflict with TCP on shared networks[1]. Therefore, it is desirable to design satellite systems that can carry TCP traffic efficiently. The next section introduces the basic design of

[1]This is a consequence of TCP's congestion control algorithm, which will be discussed later.

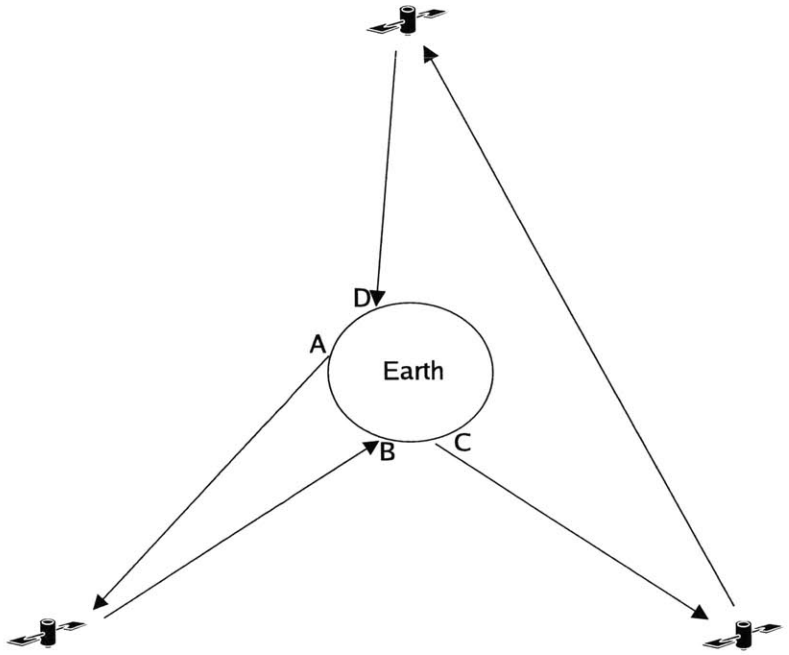


Figure 1-1: How communications satellites work. A single satellite may carry messages across long distances, such as from A to B. Using multiple satellites, messages can be carried anywhere in the world, such as from C to D.

satellite communication systems, and how they differ from terrestrial networks.

1.2 The Satellite Channel

A geosynchronous communications satellite is essentially a radio[2] placed into orbit around the Earth. By choosing just the right altitude, its orbital period can be made to match the planet's rotation, so that from the ground the satellite appears to hang stationary at the same spot in the sky. This allows communication across long distances, because messages can be relayed through the satellite to receivers beyond the horizon. See Fig. 1-1.

There are two kinds of communications satellites: transponding and processing. Conceptually, a transponding satellite simply bounces radio signals from sender to receiver, without any knowledge of the signal's contents. In contrast, a processing

[2]Most existing systems operate at radio frequencies, but some new and future systems can operate at optical frequencies.

satellite demodulates the incoming radio signal and generates a new signal that is transmitted to the receiver. Transponding satellites are simpler, while processing satellites are capable of digital signal regeneration.

The communication channel between a ground-based terminal and a geosynchronous satellite differs in several important ways from a typical terrestrial link. Geosynchronous satellites orbit at an altitude of 35,786 km, which introduces a propagation delay of about 120ms up to the satellite, and another 120ms down to the remote user. In contrast, typical terrestrial links experience propagation delays 4 to 6 orders of magnitude smaller. [3] Data rates on both uplinks and downlinks range from many megabits per second for large stationary terminals to rates as low as 2400 bits per second for small mobile terminals.

Unlike wired networks, satellite networks suffer frequent data corruption. Typical bit error rates for satellite links are between 10^{-5} and 10^{-8}. Mobile satellite receivers can experience error rates as high as 10^{-4} [29]. In contrast, the specification for 100BASE-T Ethernet lists the worst case bit error rate as 10^{-10}. The errors on satellite links tend to be bursty, which makes recovering from them more difficult. Mobile receivers experience frequent deep fades whenever their line of sight to the satellite is blocked, resulting in long sequences of lost packets. Such blockage events mean that any assumptions of independent packet loss are invalid; errors are highly correlated.

Another disadvantage of the satellite channel is that it is often asymmetric. Vehicle-mounted mobile transmitters have limited peak power output caused by the limits of power amplifier design[4]. This can result in an uplink much slower than the downlink. Such bandwidth asymmetry can negatively impact protocols like TCP that depend on receiver feedback. The next section describes TCP in more detail.

[3] One way to significantly reduce propagation delay is to use low Earth orbit (LEO) satellites. However, a geosynchronous (GEO) satellite provides vastly larger coverage than a LEO satellite, and the rapid movement of LEO satellites makes antenna pointing, data routing, and network planning significantly more complicated. This research will focus exclusively on GEO satellites.

[4] Space-borne transmitters can use travelling wave tube amplifiers, which are large, fragile, and expensive. Ground vehicle transmitters use solid-state amplifiers which are more rugged.

1.3 Introduction to TCP

TCP provides two key functions within any data network:

- Reliable Delivery: TCP attempts to ensure that all data a sender transmits ultimately arrives intact and in order at the receiver.
- Congestion Control: TCP prevents network congestion collapse by pacing senders, so that they do not overwhelm any part of the network.

TCP achieves these goals using a window-based automatic repeat request (ARQ) protocol. The sender splits the data into segments, labels each segment with a sequence number, and transmits the segments to a receiver. The receiver sends an *acknowledgment* (ACK) for each segment successfully received. Each ACK contains the first sequence number not yet received, implicitly acknowledging all preceding segments. Based on this feedback, the sender can detect lost segments and perform retransmissions. For more fine-grained feedback, many TCP hosts implement the *selective acknowledgment* (SACK) option, which allows arbitrary blocks of segments to be acknowledged. If the transmitter receives no feedback, its retransmission timer will eventually expire causing it to try again. Retransmissions are described in greater detail below.

To efficiently utilize available bandwidth without contributing to congestion, each TCP sender maintains a *congestion window*. The congestion window is the set of segments that have been transmitted but not yet acknowledged. The size of the congestion window equals the number of unacknowledged bytes that the sender may have in the network at any given time. As segments are acknowledged by the receiver they are removed from the congestion window, making room for new segments. The ideal congestion window size is equal to the bandwidth-delay product of the link, because it allows the channel to be completely utilized without causing congestion.

For a given link, a larger congestion window corresponds to a higher average data rate. When a new connection is established, the congestion window is initialized to a conservative value and the sender enters *slow start* mode. During slow start,

the congestion window size is increased exponentially as segments are successfully transmitted and acknowledged. When a segment is lost, the TCP sender assumes that congestion has occurred, and responds by decreasing its congestion window by one half and entering *congestion avoidance* mode. It is important to note that this assumption is not always correct; packet losses are not always caused by congestion, especially on wireless links. During congestion avoidance, the congestion window is increased linearly when segments are transmitted successfully and decreased by one half when segments are lost.

The size of the congestion window is limited by the size of the *receive window*. The receive window is advertised by the receiver and indicates how many bytes the receiver is prepared to handle. This prevents a fast sender from overwhelming a slower receiver. The maximum receive window supported by TCP is 2^{16} bytes. On high bandwidth-delay links (e.g. high-speed fiber, satellite links), this is often insufficient to achieve full utilization. The *window scaling* option allows windows up to 2^{30} bytes.

If multiple segments are lost, TCP is forced to timeout and wait before retransmitting the segments, in order to allow congestion to clear. Choosing the correct length of the *retransmission timeout* (RTO) is important for performance: too short, and unnecessary retransmission will contribute to network congestion; too long, and we leave the link sitting idle. Ideally, RTO should equal the round trip time (RTT), so that lost segments are retransmitted as quickly as possible. However, RTT can vary significantly over time and maintaining an accurate estimate is difficult. TCP maintains an smoothed average of measured round trip time, SRTT, and an estimate of the deviation in round trip time, D, which are used to calculate RTO:

$$Err = RTT - SRTT$$

$$SRTT = SRTT + g \cdot Err$$

$$D = D + h \cdot (|Err| - D)$$

$$RTO = SRTT + 4D$$

where g and h are tunable constants, typically $g = 0.125$ and $h = 0.25$. This timeout calculation has important implications for TCP performance on lossy links.

1.4 The Challenge of TCP over Satellite

The fact that TCP is ill-equipped for wireless (and especially satellite) links has been well-understood for some time [38]. The dominant problem is TCP's congestion control algorithm. TCP's congestion control algorithm works extremely well on traditional networks, where packet losses are almost always a symptom of congestion. However, on satellite links the dominant cause of packet loss is data corruption. The TCP sender misinterprets this loss as congestion and enters congestion avoidance mode unnecessarily, leaving the link underutilized.

TCP is also negatively impacted by high delay. A fast satellite link has a large bandwidth-delay product and therefore requires a large congestion window in order to keep the link occupied. Unfortunately, the congestion window grows slowly because of the long round trip time, leaving the link underutilized for a long period of time (on the order of many seconds) at the start of a connection. It is also critical that the TCP window scaling option is enabled, otherwise the congestion window is never allowed to grow to a large enough size.

Furthermore, TCP is known to be biased against sessions with long round trip times. Short RTT sessions can acknowledge segments faster, which allows their congestion windows to grow faster, which allows them to claim more bandwidth. Therefore, sessions carried by satellite will tend to be penalized during network congestion in favor of terrestrial traffic. [20, 22, 31]

High delay and frequent packet loss are both bad for TCP, but in combination they are even worse. Delay causes TCP timeouts to be long, and packet loss causes TCP timeouts to be frequent. As shown in chapter 4, a high-latency TCP connection experiencing 30% packet loss can sit idle over 90% of the time. In addition, packet loss and delay work together to keep the congestion window small; much too small to keep the link utilized. An analytical study in [24] claims that TCP suffers significant

throughput deterioration when the product of the loss probability and the square of the bandwidth delay product exceeds unity. For a 256kbps satellite link with 0.5 second round trip time, this implies that a packet loss probability above $3.7 \cdot 10^{-9}$ is unacceptable. This assumes that losses are independent however, which is unrealistic for the mobile case. Assuming losses are correlated, the theoretical channel capacity increases, but TCP timeouts become more likely, so the net effect is not immediately clear.

Bandwidth asymmetry can also degrade TCP performance. TCP relies on a steady stream of acknowledgment's (ACKs) to clock its transmit algorithm and ensure correctness. These acknowledgments can become a bottleneck as ACKs are queued for transmission over the low-rate uplink [6].

1.5 Improving TCP Performance

Previous attempts to improve TCP performance over wireless links fall into three basic categories, as described by Balakrishnan [7]: end-to-end techniques, split-connection techniques, and link-layer techniques. End-to-end techniques try to modify TCP to better handle wireless packet loss. They include TCP options like selective acknowledgments, window scaling, explicit congestion notification, path MTU discovery, and others [1, 17, 25]. Many of these TCP options were originally developed for other domains, but also help in satellite networks [5]. The problem with end-to-end solutions is that they require the modification of any TCP host whose traffic will cross a satellite link. Even widely accepted TCP options may not be implemented by all hosts. This represents a huge installed base that would be expensive and slow to change.

Split-connection techniques attempt to completely hide the satellite link from TCP. TCP connections are terminated at a gateway before the satellite link, the data is carried across the link using a satellite-optimized protocol, and a new TCP connection is initiated from the far gateway to the destination (see Fig. 1.5). Many variations have been studied [21, 27–30, 33, 37]. Split-connection techniques show good performance, but have some serious problems. Normally, TCP has the beneficial prop-

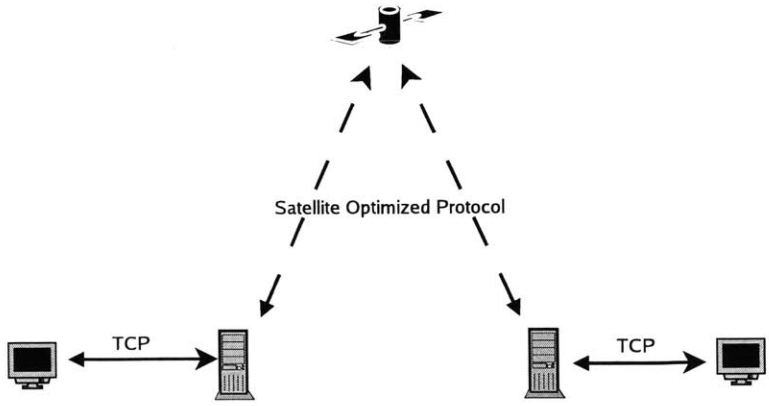


Figure 1-2: Split-connection strategies use two separate TCP connections between the end hosts and the gateways, with a satellite optimized protocol in the middle.

erty of *fate sharing*, which means that as long as a path exists between the endpoints, they will continue to operate correctly; only the failure of an endpoint will necessarily terminate the connection. Split-connection techniques violate the end-to-end semantics of TCP, which shifts the burden of reliability to the gateways, eliminating fate haring and creating a central point of failure. A crashed gateway could potentially crash every application running across the link. Furthermore, the violation of end-to-end semantics makes the use of IP layer encryption or authentication difficult or impossible.

Link-layer techniques attempt to improve performance by modifying the link layer. The link layer runs below the network layer (for example, IP) and above the physical layer (for example, Ethernet). Link layer techniques require no changes to TCP and preserve TCP's end-to-end semantics. There are two broad classes of link-layer techniques: true link layers, and TCP-aware link layers. True link layers, such as the Lincoln Laboratory Link Layer [39], respect the network layer abstraction and provide increased reliability through automatic repeat request (ARQ) and/or forward error correction (FEC) [13]. In [11], approximate analysis is used to show that link layer error recovery can effectively support TCP on lossy links; however this study does not consider correlated losses.

TCP-aware link layers, such as the Snoop protocol, actively monitor and modify

TCP header information to increase performance [3, 36]. They provide attractive performance benefits, but they cannot work with IP layer security (i.e. tunneled IPsec), because they rely on reading transport layer headers. This makes TCP-aware link layers unattractive for military applications, where packets are likely to be encrypted. This thesis focuses on true link layers, because of their general applicability and their amenability to encryption.

1.6 Related Work on Wireless Link Layers

Most of the literature concerning wireless link layer design is focused on terrestrial systems, because of the rapid growth of wireless LANs and mobile telephony. Much of this is applicable to the satellite case, with some extensions.

Many previous studies of wireless link-layer design have not considered the high-latency case [7, 12, 13, 41]. Others treat losses as uncorrelated or ignore losses completely, which is not realistic for land-mobile transmitters [4, 9, 23]. A recent paper by Briceno [10] examines TCP performance on a mobile satellite channel, including the use of link-layer performance enhancements, but does not delve deeply into link-layer design considerations.

Analytical studies are faced with two important challenges: how to model the wireless channel, and how to model TCP behavior. Because of the complexity, authors are often forced to either oversimplify the channel model, such as in [23], or oversimplify the traffic model (TCP behavior), such as in [34, 40].

The most widely used discrete model for wireless channel performance is the Gilbert model [18]. It captures the effect of burst errors by defining a two-state Markov model with "good" and "bad" states. This can correspond to the physical processes of shadowing and multi-path fading. The Gilbert model has been shown to approximate the land-mobile satellite channel quite well [26, 35].

Models for TCP behavior tend to be either too complicated to analyze or too simple to capture the salient behaviors. See for example [32]. The situation is complicated by the fact that several competing versions of TCP exist [5, 23].

In [42], selective repeat ARQ with a Gilbert error model on both the forward and reverse channels is studied analytically. The paper shows that what they call *time diversity* can effectively combat imperfect feedback. In this context, time diversity refers to waiting longer than one round trip time before assuming that lack of feedback implies loss of data. This can improve throughput efficiency by preventing retransmission in the case where packets are received successfully, but the corresponding acknowledgments are lost.

Forward error correction (FEC) techniques for wireless links are explored in [8]. FEC alone cannot compensate for the long blockage events that characterize the land-mobile satellite channel without introducing a large constant delay. Such a delay would have a strong negative impact on TCP. Furthermore, FEC would require a constant low rate code, whereas ARQ allows the use of a high rate code during periods without blockage.

It is also possible to combine FEC and ARQ; this technique is called Hybrid ARQ (HARQ). HARQ schemes for wireless links are explored in [14, 19]. Such schemes fall into three classes. Type I HARQ uses FEC to encode each packet against errors, and then performs ARQ to ensure correct reception of all packets. This strategy can be useful for land-mobile satellite links. Type II and Type III HARQ attempt to use information from erroneously received packets to aid the decoding of future packets. These techniques are probably not useful for the land-mobile satellite channel, because lost packets tend to be completely lost because of deep fading, with almost no useful signal. Therefore, type II/III HARQ schemes are not worth the added complexity.

Chapter 2

Link Layer Protocol Design

2.1 Assumptions

The goal of this thesis is to design, implement, and test an experimental link layer protocol that can effectively carry TCP over a mobile satellite system. The protocol is intended to work with the existing Milstar platform. Therefore, we can make several assumptions:

- The underlying physical layer is a circuit-switched network. Therefore, we should make full use of the channel at all times - any time we do not use is wasted, because no other sender can utilize it anyway.
- The underlying physical layer preserves frame order.
- The transmitter is vehicle-mounted and not seriously power-constrained.
- Encryption will be enabled at the IP layer. This requires the use of a true link layer, as opposed to a transport-aware link layer.
- The channel will undergo frequent correlated packet loss.
- The uplink may be significantly slower than the downlink.

2.2 Design Considerations

There are two well-known strategies for building reliable link layers: automatic repeat request (ARQ) and forward error correction (FEC). ARQ detects and retransmits damaged or missing data. FEC encodes redundancy into the data to allow error recovery.

Satellite transceivers typically perform FEC as part of their coding scheme, which corrects relatively isolated bit errors. However, the land-mobile satellite channel typically experiences large bursts of errors (up to many seconds), which cannot be corrected. ARQ is the best way to cope with these highly correlated losses. To correct both isolated and burst errors efficiently, we can run FEC and ARQ together in a Type I Hybrid ARQ (HARQ) scheme. An added benefit of Type I HARQ is that it achieves clean separation between the FEC and ARQ protocol layers. This work focuses solely on the ARQ layer.

There are several important trade-offs to consider:

- The link layer should attempt to hide errors from TCP in order to avoid triggering TCP's congestion control. However, the link layer needs to give up before a TCP timeout occurs, triggering TCP-level retransmissions. TCP often benefits from timely link layer delivery with sufficient reliability, rather than perfect reliability with increased delays [16].

- The link layer should try to ensure in-order delivery, because out-of-order delivery can trigger TCP retransmissions. However, performing packet ordering at the link layer can cause large variations in TCP's estimate of round trip time, which can result in long timeouts.

- The receiver should provide detailed feedback to the sender, to ensure that lost packets are retransmitted in a timely fashion without unnecessary retransmissions. However, the return path may be constrained due to bandwidth asymmetry, so the receiver needs to minimize its feedback bandwidth.

- The amount of buffering at the link layer should be large enough to prevent excessive packet drops, but small enough to avoid added latency.

Another design decision is the size of link layer data units. By fragmenting IP packets into smaller units, the link layer can experience a lower packet loss rate. Smaller data units also allow finer-grained error recovery, which minimizes wasted retransmission bandwidth. However, if the data units are too small the overhead costs associated with headers goes up.

It is also important to consider the link layer's blockage mitigation strategy. When channel blockage triggers a TCP timeout, TCP will wait for perceived congestion to clear, and then perform retransmission at exponentially increasing intervals. This can cause the channel to sit idle for several seconds after blockage has cleared. This significantly reduces throughput, especially if blockages are so frequent that TCP cannot detect the brief open periods between blockages.

A properly designed link layer can mitigate this problem by detecting the end of blockage and delivering data to TCP, which will trigger a rapid recovery. Scott and Map [36] describe a link layer protocol that achieves this goal. However, their protocol is TCP-aware and maintains per-connection state; it does not satisfy our goal of designing a true link layer. We can achieve a similar (or better) result by taking advantage of the circuit-switched nature of our network and transmitting continuously. By transmitting continuously, even during blockage, the link layer will take advantage of re-connections immediately.

A true link layer only needs to maintain per-link state, as opposed to per-connection state. This scales much better in large systems, and it allows us to dedicate more resources per protocol instance. We can therefore use relatively large buffers if necessary. By buffering a large window of recently sent traffic, the sender can achieve a high bulk-throughput efficiency despite imperfect feedback from the receiver. Zorzi and Rao [42] demonstrate such a strategy. However, their system explicitly retransmits after a fixed timeout period, based on the round trip time. Because of the continuous transmissions and large windows we have available, we can eliminate the need for explicit retransmission timeouts, as described in section 2.3. This simplifies

the link layer and eliminates the need to estimate the round trip time, which would be difficult because RTT varies considerably as ACKs are subjected to queuing delay.

2.3 Protocol Design

The Comm-On-The-Move Link Layer (COTMLL) was designed at Lincoln Laboratory by Andrew Worthen and implemented and tested by the author. It is a point-to-point selective repeat windowed ARQ protocol designed to carry TCP/IP traffic[1] over frequently-blocked, asymmetric links. The protocol instance is clocked from the layer below; it generates a frame whenever the underlying link requests data. Therefore, COTMLL runs at a constant average rate intended to completely utilize the underlying link.

COTMLL runs on top of an Encrypted Frame Layer (EFL). The EFL provides an interface between the frame-oriented link layer and the serial physical layer, while encrypting/decrypting the data with a block cipher (Advanced Encryption Standard [2]). It detects frame boundaries within the serial stream, verifies frame integrity with a checksum, and resynchronizes when frame boundaries are obscured by channel errors. Figure 2.3 depicts the relationships between TCP, IP, COTMLL, and EFL. It is important to note that the frame layer version used in emulation does not perform encryption/decryption, because the decryptor is implemented more efficiently in specialized hardware.

As TCP/IP packets are accepted from the higher layers, COTMLL fragments them into smaller frames[2]. This decouples the TCP/IP packet size from the frame size, which is helpful because the packet size and frame size are under competing design pressures. Larger packets are more efficient because they obviate the need for IP fragmentation (which is highly inefficient) and amortize the cost of headers across more data. Smaller frames are more efficient because they are less likely to undergo

[1] COTMLL can carry any IP traffic, however it is optimized to support TCP/IP.

[2] "Packet" refers to the TCP/IP transfer unit, whereas "frame" refers to the link layer transfer unit. Unlike Ethernet, COTMLL does not have a one-to-one mapping between packets and frames. Instead COTMLL has a one-to-many mapping.

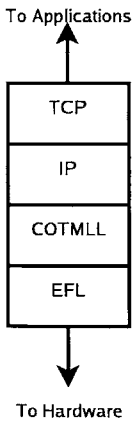


Figure 2-1: The relationships between four protocol layers. TCP is the transport protocol, responsible for end to end reliability and flow control. IP is the network protocol, responsible for wide-area packet routing. COTMLL is the link layer reliability protocol, providing increased point-to-point reliability. EFL is the link layer framing and security protocol, providing frame serialization and encryption.

loss, they allow more fine-grained control of data retransmission, and they can be transmitted completely in a reasonable length of time even on slow links. Packet fragmentation allows COTMLL to support the standard Ethernet maximum transfer unit (12000 bits) while running with much smaller frames (up to 3968 bits). Frames are sent at maximum size whenever possible, but no padding is performed if less data are available.

The frame size is chosen to make a good trade-off between frame overhead and frame loss rate. The COTMLL trailer is 32 bits, and the EFL header is 40 bits, so the total frame overhead is 72 bits. If we select a frame size of 3968 bits (which is thirty-one 128 bit code blocks), the result is an overhead of 1.8%. Assuming an independent random bit error rate of 10^{-5}, this results in a frame error rate of 3.9% (compared to 11% if we performed no fragmentation). A dynamically chosen frame size could conceivably achieve lower overhead during good channel conditions, but the overhead is so low (1.8%) that it does not justify the additional complexity.

Each frame is appended with a four byte trailer. The format of the COTMLL trailer is given in Fig. 2.3. A trailer is used rather than a header in order to enable just-in-time acknowledgments, an optimization for slow links described below. There

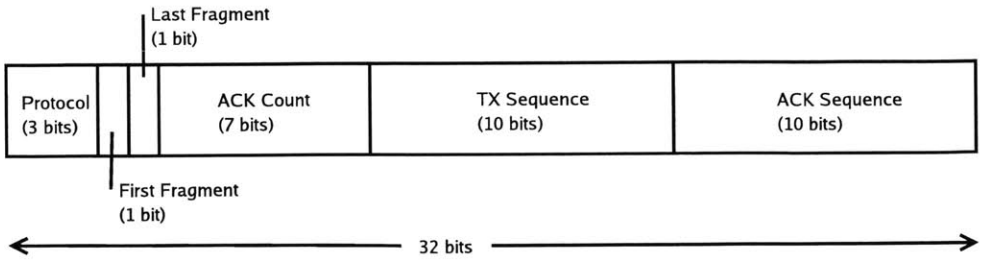


Figure 2-2: The COTMLL trailer format.

is no need for a header, because frame boundaries and frame size are detected by the EFL; COTMLL receives full frames. The trailer fields are as follows:

PROTOCOL Identifies the sub-protocol type of the frame. Defined sub-protocols include:

000 The payload is a control message.

001 The payload is part of an IP packet that should be retransmitted if necessary.

010 The payload is part of an IP packet that should not be retransmitted.

FIRST FRAGMENT Indicates that this frame is the first fragment of an IP packet.

LAST FRAGMENT Indicates that this frame is the last fragment of an IP packet.

ACK COUNT If this field is 0, then the frame contains a *cumulative acknowledgment*. Otherwise, this frame contains a *block acknowledgment*, and this field is the number of sequential frames being acknowledged.

TX SEQUENCE Indicates the sequence number of the frame. Sequence numbers are assigned sequentially, modulus 1024. Unlike TCP, COTMLL sequence numbers count frames, not bytes.

ACK SEQUENCE In a cumulative acknowledgment, this field contains the sequence number of the first unreceived packet. In a block acknowledgment, this field contains the sequence number of the first packet being acknowledged.

The trailer format is designed to enable piggybacked acknowledgments; acknowledgments for frames flowing in one direction are carried within frame trailers flowing the opposite direction. There are two different kinds of acknowledgments. A *cumulative acknowledgment* acknowledges all frames up to a given sequence number. This is signified by setting ACK SEQUENCE to the first unreceived sequence number, and ACK COUNT to zero.

The second kind of acknowledgment is a *block acknowledgment.* A block acknowledgment acknowledges the receipt of a continuous block of packets not located at the start of the window. To signify a block acknowledgment, ACK COUNT is set to the length of the block, and ACK SEQUENCE is set to the first sequence number in the block. Block acknowledgments are only sent for the first received block in the window, therefore they implicitly indicate that all unacknowledged packets before the block have been lost. Whenever a gap exists in the receive window, cumulative and block acknowledgments are sent in an alternating fashion. When no gap exists, only cumulative acknowledgments are sent.

Whenever a COTMLL sender is asked to send a frame, it attempts each of the following actions in order until one succeeds:

1. If any previously sent frames have been flagged for immediate retransmission, retransmit the first one in the window.

2. If there is free space in the transmission window and new data waiting to be sent, transmit a new frame.

3. If there are any previously sent frames that are unacknowledged, select the least recently sent one for retransmission.

4. If there is nothing else to send, generate a bare acknowledgment message.

Frames are flagged for immediate retransmission based on received block acknowledgments. The receipt of a block acknowledgment indicates that all preceding frames in the window are still outstanding. These outstanding frames require retransmission,

unless they have already been retransmitted more recently[3] than the acknowledged block. Such definitely lost frames are flagged to be retransmitted at the highest priority, in order to keep the window moving. This block acknowledgment scheme is optimized for the blockage channel, because we expect large gaps in the receive window, as opposed to many small gaps.

Bare acknowledgment messages are defined using the control message sub-protocol. They contain no data, but are useful because they convey the current receive state of the host. Bare acknowledgments are necessary when data is flowing in only one direction and piggybacked acknowledgments are not possible. In practice, this is rare because TCP generates its own acknowledgment stream, causing a two-way flow.

The state required for COTMLL can be organized into four data structures. The first is the sliding transmission window which maintains copies of all previously sent frames, along with flags indicating which have been acknowledged and which require immediate retransmission. The second is an ordering of the unacknowledged frames that indicates in which order frames were most recently sent. The third is the sliding receive window, which only needs to contain one bit per frame to indicate whether a frame has been received. Finally, the fourth data structure is a queue to hold received frames before they can be assembled into full packets.

[3]This requires that we maintain an ordering of all outstanding packets, based on the order they were most recently sent. This can be efficiently maintained as a linked list.

Chapter 3

Experimental Methods

This chapter describes the nature of the experiments conducted. All experiments were carried out within our networking emulation software architecture, which is described below. Each experiment was designed by choosing a set of channel parameters, a set of protocol parameters, a traffic model, and a set of metrics.

3.1 Software Architecture

I developed a networking emulation test bed at Lincoln Laboratory dubbed the Experimental Protocol Stack (XPS). Using XPS, one can implement experimental link layer protocols and integrate them directly with the Linux kernel TCP/IP stack. This has three important advantages:

- We can route arbitrary network traffic across the experimental link in real time.
- We are running a production TCP/IP stack, so we can be sure that we are faithfully modeling TCP behavior; perhaps not ideal TCP behavior, but TCP as it actually performs in the wild.
- XPS supports both emulated and actual hardware interfaces. So the same experimental code can be run in the lab and on real satellite links.

See Fig. 3-1 for an example test bed configuration using an emulated channel, and Fig. 3-2 for an example with real hardware. XPS is implemented as a Linux

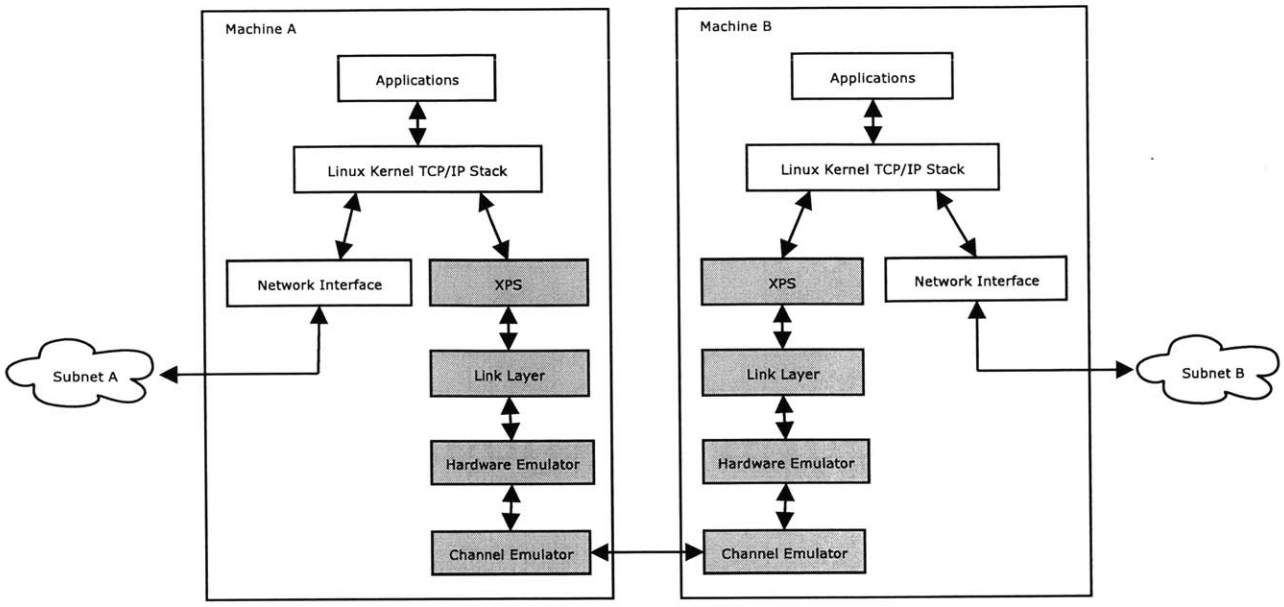


Figure 3-1: An example test bed configuration using an emulated satellite channel. The shaded modules indicate our custom software. In this case, the two machines are acting as gateways to establish a satellite route between two IP subnets. The channel emulator modules use a "hidden" Ethernet connection to pass data, while adding appropriate delay and loss characteristics.

kernel module. Using the XPS interface, users write modules (such as the link layer and channel emulator modules in the example) that can be swapped in and out of the XPS stack. This facilitates the development and comparison of multiple different protocols, channel models, etc.

3.2 Channel Modeling

The land-mobile satellite channel suffers from strong variations in signal strength due to multipath fading and signal shadowing. For EHF satellite signals with highly directional receivers, multipath fading is negligible and the dominant effect is signal shadowing.

Signal shadowing is often modeled as a two state Markov process, with a "good" state that describes the channel with a clear line of sight, and a "bad" state that describes the channel behavior during shadowing. One such two-state model is the

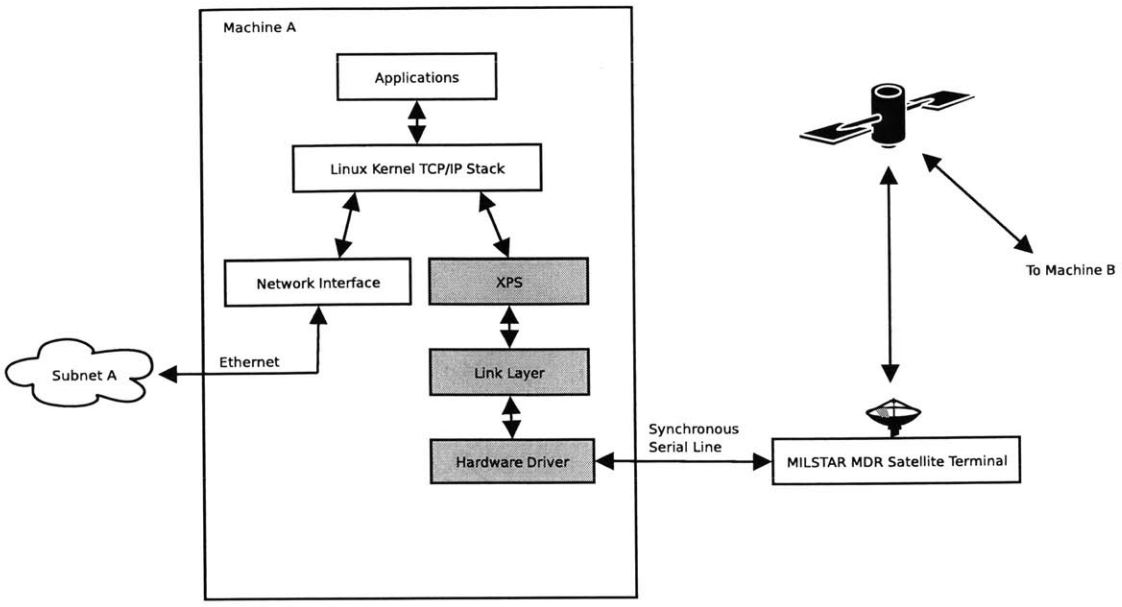


Figure 3-2: An example test bed configuration using a real satellite link. The hardware emulator from the previous example has been replaced by a hardware driver. Because the XPS system defines standard module interfaces, the same link layer code can run on both configurations.

Gilbert model [18]. According to the Gilbert model, during the good state the channel is error free, and during the bad state each bit experiences errors independently with fixed probability. The Gilbert-Elliot model is an extension which also includes a fixed bit error probability during the good state [15]. Based on field measurements, the Gilbert and Gilbert-Elliot models have been shown to be imperfect but useful for modeling mobile satellite links [35].

More recent unpublished work at Lincoln Laboratory has shown that higher-order Markov models can more faithfully match measured channel conditions. However, it is unrealistic that any single model can predict the effects of terrain, vegetation, weather, and movement under all conditions. We therefore focus on a two-state Gilbert-Elliot model, because it is simple yet captures the phenomenon of interest: channel blockage.

The Gilbert-Elliot model is characterized by four parameters: the bit error probability while blocked (b), the bit error probability while connected (i.e. not blocked) (c), the mean blockage duration (d), and probability of blockage, or blockage fraction

Parameter	Symbol
Bit Error Probability During Blockage	b
Bit Error Probability During Connection	c
Probability of Blockage (aka Blockage Fraction)	f
Mean Blockage Duration	d

Table 3.1: Summary of Channel Parameters

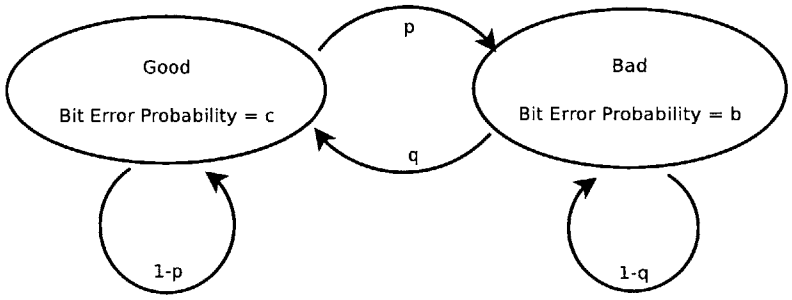


Figure 3-3: The two-state Gilbert-Elliot channel model.

(f). Table 3.2 summarizes these parameters.

The model is implemented as a discrete Markov process as depicted in figure 3.2. The transition probabilities p and q are completely determined by the parameters d and f:

$$q = \frac{1}{d} \qquad\qquad p = \frac{f}{(1-f)d} \tag{3.1}$$

Some authors treat the transition probabilities as parameters and do not define blockage duration d and blockage fraction f. I have chosen to do so because these parameters have a more intuitive physical meaning. In addition, f is a very good approximation for the channel entropy, H, as long as d is significantly longer than the transmission time of a single bit. This is useful because we can find the channel capacity $C = 1 - H \approx 1 - f$. See Appendix A for a derivation. Therefore, channel capacity is approximately equal to the proportion of time the channel is not blocked.

Field measurements indicate that blockage events are typically severe enough to cause complete loss of signal. We therefore assume that the bit error probability

during blockage is 1/2, that is $b = 0.5$. This serves as a realistic approximation of empirical results, and it represents the worst possible case in terms of blockage severity. In contrast, the bit error probability during connection c can vary considerably based on conditions and its effect is explored in the next chapter, along with the effects of d and f.

Our software implementation of the error model allows error states to be controlled independently on each direction of the link. However, in all experiments considered here, we assume that errors are perfectly correlated between directions, because blockage will apply to both the uplink and downlink signals.

In addition to the error model, the channel must account for delays inherent in a satellite network. One-way propagation time to the satellite is approximately 120ms. In existing systems such as MILSTAR, coding techniques built into the hardware, such as interleavers, can introduce substantial additional delay. For this emulation work, an estimate of 250ms one way latency from terminal to terminal was selected. This delay does not vary significantly in time and is modelled as a constant.

3.3 Traffic Generation

Our experimental setup can carry traffic from any existing IP network application. However, in order to carefully control the traffic characteristics we employ several idealized traffic generation models. The simplest is a bulk-transfer application. Such an application transfers a large fixed amount of data at the highest rate TCP will accept. This is similar to downloading a large file via TCP or HTTP.

The second application is a rate-controlled source. This application generates data at a fixed rate and attempts to transmit it via a TCP connection. This scenario is similar to live streaming media, such as video teleconferencing or voice over IP, and it allows us to accurately measure the latency characteristics of the link. It also provides a model for quality of service provisioning, because we can decrease the offered load and measure the resulting improvement in latency.

The third application is a transaction system. It measures interactive application

performance by opening many short-lived TCP connections. This is similar to web browsing or database access.

Finally, multiple instances of each application can be run concurrently, in order to model environments where different users and traffic types compete for resources. For example, running many concurrent TCP connections may achieve higher utilization than a single TCP connection (because the sum of the congestion windows starts larger), even though the available bandwidth is the same. Or for another example, the presence of bulk-transfer traffic may significantly degrade transaction performance.

3.4 Measurements

3.4.1 Types of Measurements

The traffic generation applications have built-in measurement capabilities. Each measures TCP throughput over time by counting bytes received. In addition, each application embeds timestamps in the TCP data stream in order to sample the end-to-end latency. These end-to-end measurements provide insight into actual application performance as described in the previous section.

In order to understand the behavior of both TCP and the link layer protocol at a more detailed level, XPS has an integrated lightweight logging facility that can record every detail of protocol behavior. For example, the logs record the sequence number of each TCP packet that passes through the link, the sequence number of every link layer frame, all link layer acknowledgment messages, and salient changes to internal link layer state variables. We can therefore detect every TCP retransmission and watch as the link layer recovers from each individual blockage event.

3.4.2 Steady-state Performance Measurements on Random Channels

One difficulty in generating meaningful aggregate performance measurements is the long memory associated with the system. Random blockage events, particularly if

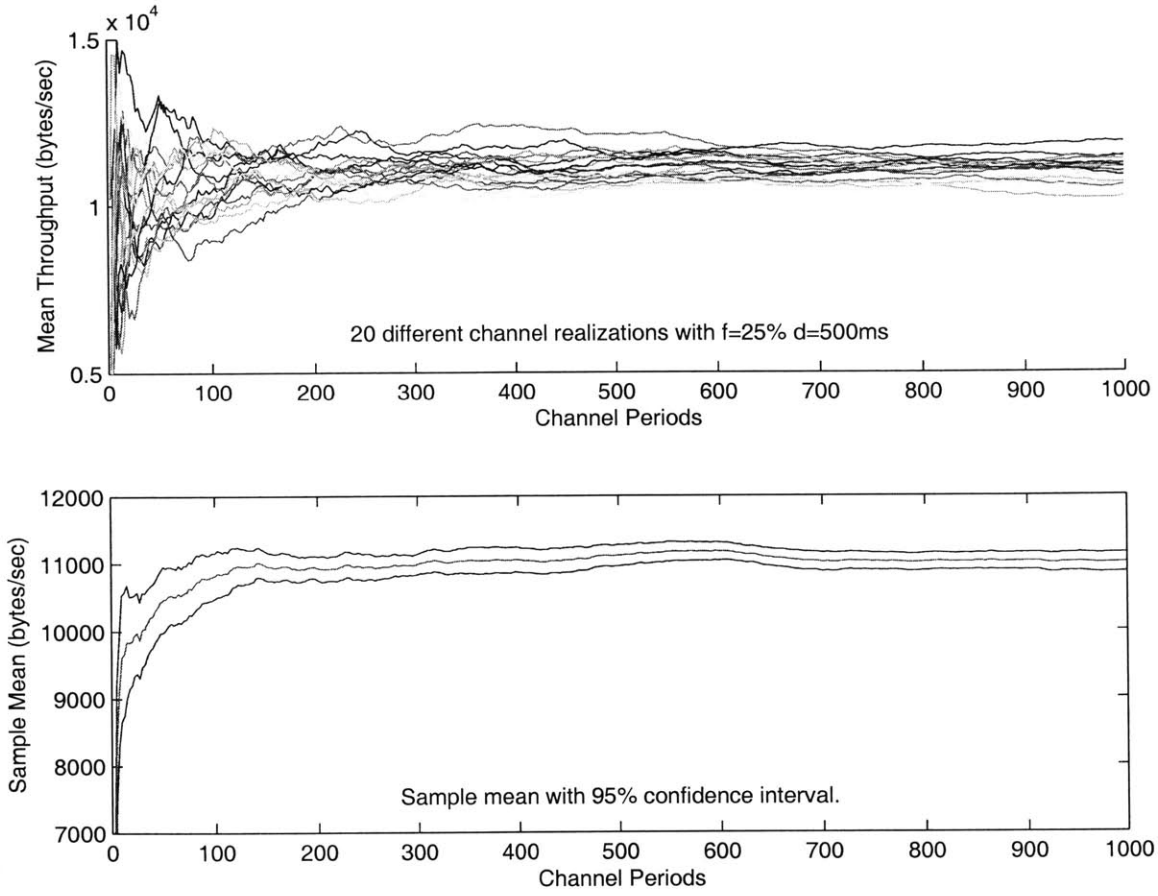


Figure 3-4: The top graph shows the mean throughput estimate over time for 20 different channel realizations, each with the same parameters. The length of the experiment is measured in channel periods. A channel period is defined as the expected amount of time required for the channel to make one full cycle through the states (in this case, 2 seconds). The bottom graph summarizes the results by presenting the sample mean of the 20 measurements, along with a 95% confidence interval (assuming a normal distribution). Note that in order to measure steady-state performance, we must wait approximately 150 channel periods for startup transients and TCP slow start effects to die out.

they occur near the start of a connection, can sometimes have long-lasting effects on performance. Fig. 3.4.2 shows how the mean throughput measurement varies as the length of the experiment increases. Initially, there is very high variation because performance is dominated by random early blockage events. As the experiments get longer, the variation decays and reaches a constant level which does not decay further. Notice that even in this steady state, individual channel performance varies considerably over long periods of time. We therefore rely on repeated measurements and use the sample mean as an estimate of steady-state performance. The figure indicates that in this example, we can achieve a reasonably steady-state performance estimate after 150 channel periods.

3.4.3 Comparing Different Scenarios

In order to generate meaningful comparisons between different scenarios, we generate a single realization of a random channel and use it repeatedly while varying other parameters. This ensure that not only channel statistics, but precise channel behavior, are held constant across measurements. Similarly, in order to generalize the difference in behavior between two scenarios, we generate a set of channel realizations and run both scenarios against this constant set.

Another important aspect of comparison is measurement normalization. In these results I use two different normalization methods. *Normalized throughput* is actual throughput achieved divided by the channel rate. For example, a connection that achieves 128kbps over a 256kbps link has 0.5 normalized throughput. *Throughput per channel capacity* is actual throughput achieved divided by channel capacity. Due to blockage, channel capacity may be less than nominal channel rate. For example, a connection that achieves 128kbps on a 256kbps link that experiences 50% blockage has a throughput per channel capacity of 1.0.

Chapter 4

Results

This chapter is organized into several classes of experiments. First, we measure the performance of TCP with link layer ARQ, hereafter referred to as TCP-LL, and compare it to the performance of TCP without any link layer ARQ. Second, we explore TCP-LL's sensitivity to channel parameters. Third, we measure the performance effects of the protocol design parameters.

4.1 Performance of TCP-LL vs TCP

The first set of experiments characterizes the bulk throughput performance of TCP-LL in its baseline configuration, and compares it to the performance of TCP alone. First we compare the two scenarios on a simple channel, and then on a more realistic blockage channel. TCP-LL's baseline configuration has a link layer window size of 512 frames and no packet reordering.

4.1.1 Binary Symmetric Channel

Figure 4-1 summarizes the throughput performance of a TCP bulk transfer on a binary symmetric channel. The link layer clearly provides an order of magnitude improvement in error protection on this kind of channel. This simple channel is a limiting case of the full blockage channel, as probability of blockage approaches zero.

The blockage channel parameter c corresponds to the bit error rate of the binary symmetric channel. Unblocked BERs between 10^{-7} and 10^{-4} are frequently seen in the field due to weather and other interference, so the fact that the link layer outperforms TCP in this range is relevant.

The performance difference can be explained by approximate analysis. In both scenarios, the packet error rate p_p is related to the bit error rate p_b by

$$p_p = 1 - (1 - p_b)^n$$

where n is the packet size in bits. For TCP $n = 12000$, whereas for TCP-LL, $n = 4000$ because of fragmentation. Therefore, TCP-LL experiences a lower packet error rate. This factor alone, however, is not enough to explain the difference in performance.

TCP throughput on lossy channels is dominated by the occurrence of timeouts. The bandwidth-delay product in this example is 16kB, and the TCP segment size is 1500 bytes. Therefore, at steady-state, the TCP window will contain approximately 11 packets. If more than one packet is lost, a timeout will occur. Therefore, the probability of timeout is

$$p_{timeout} = 1 - p_p^{11-1}$$

We can use this timeout probability as an estimate of TCP's normalized throughput.

In contrast, TCP-LL will not necessarily suffer timeouts because of multiple packet losses within a window. So we can approximate its normalized throughput with its packet loss rate. Using these approximations, we can plot predicted throughput for each scenario, as in Fig. 4-2, and confirm that it matches the observed data. In particular, the steep drops at 10^{-7} and 10^{-6} are accurately predicted.

The performance advantage of the link layer can be seen in more detail in Fig. 4-3, which depicts traces of TCP behavior with and without link layer error recovery. Both face the same channel realization, which was generated as a binary symmetric channel with bit error rate 10^{-5}. TCP experiences several packet losses early in the connection which prevent the congestion window from growing. Furthermore, repeated losses and timeouts cause the round trip time estimate to grow, as seen in

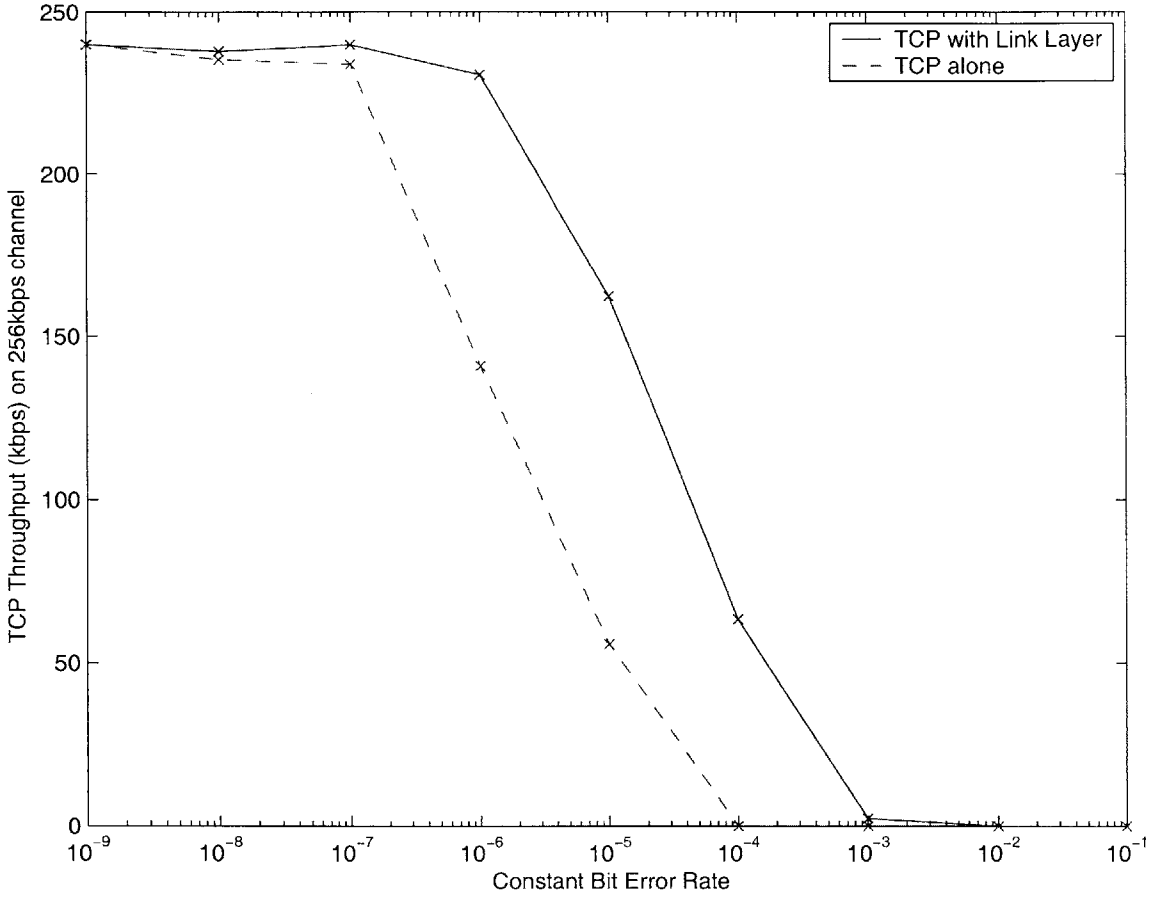


Figure 4-1: Comparison of TCP and TCP-LL performance on binary symmetric channels.

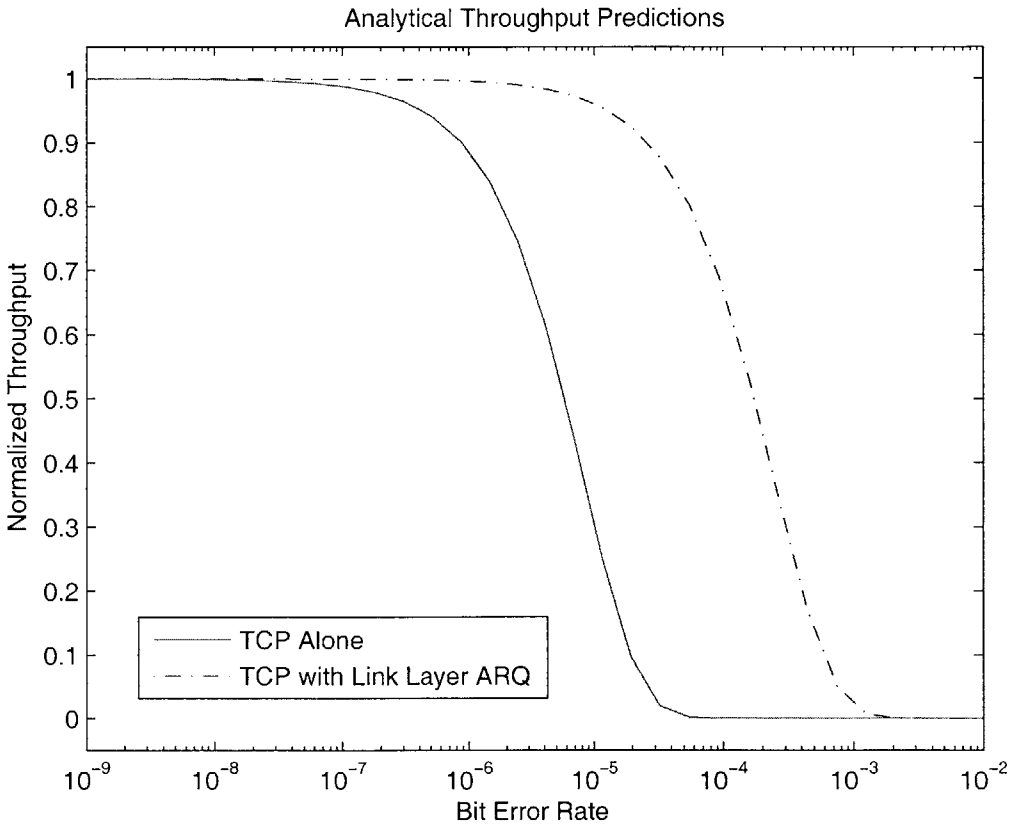


Figure 4-2: Predicted TCP and TCP-LL performance on binary symmetric channels.

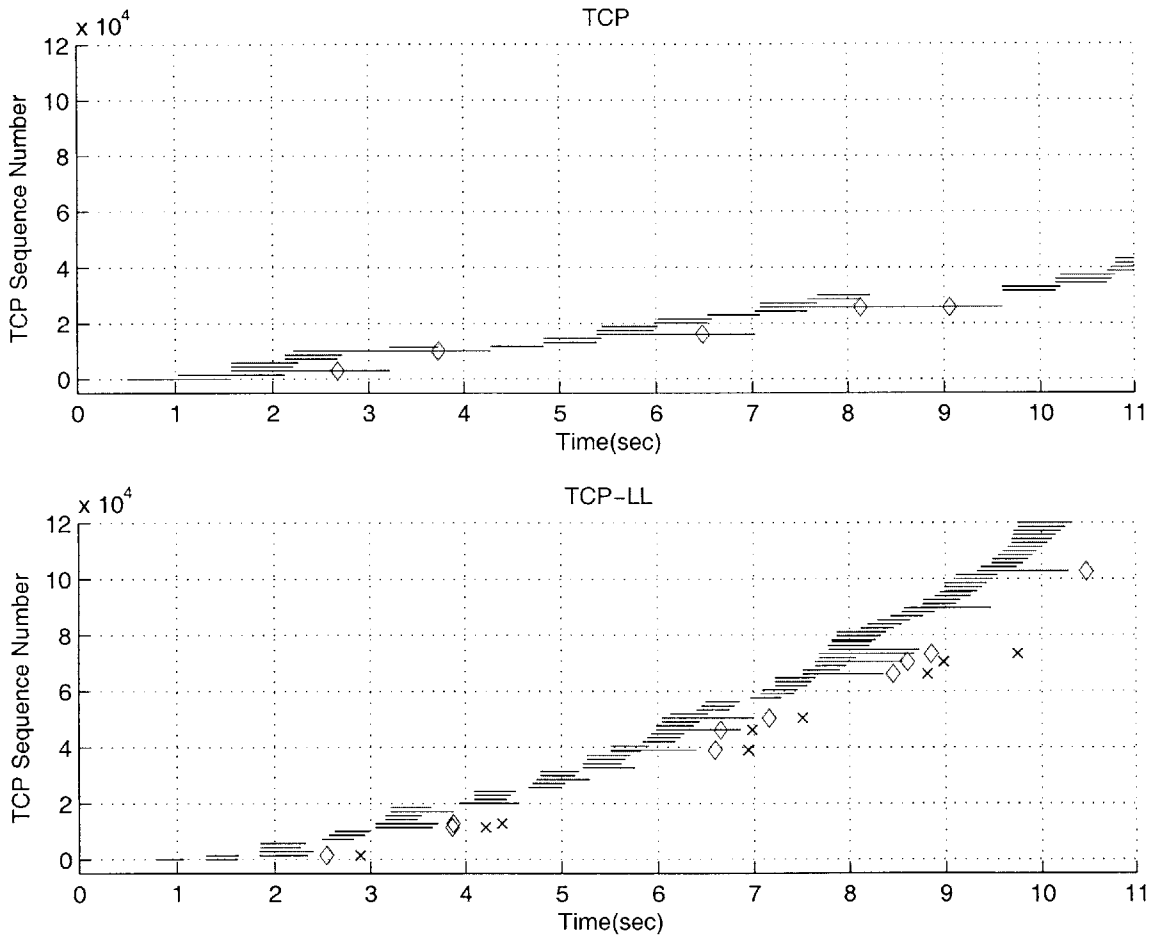


Figure 4-3: Traces of TCP-LL and TCP behavior in identical channel conditions. Lines connect the initial transmission of a packet to its initial reception. Diamonds indicate retransmissions, while X's indicate multiple receptions. The channel has a BER of 10^{-5}.

the widely spaced retransmissions after 7 seconds. This makes timeouts very long, and the connection is effectively stalled.

The situation is different for the connection with link layer ARQ. Early in the connection, several TCP timeouts still occur, due to the high channel latency and frame losses, which cause TCP to enter congestion avoidance. However, the timed-out packets are eventually received, albeit later than expected, due to link layer retransmission. The receiver then generates a series of TCP ACKs which arrive at the sender and cause its congestion window to grow, allowing new packets to be transmitted.

4.1.2 Blockage Channel

Next we compare TCP performance with and without link layer error control on more realistic blockage channels. Fig. 4-4 is the result of averaging 25 runs per data point. It shows the effect of increasing blockage fraction (f) on TCP throughput. Beyond a blockage fraction of 0.4, TCP stalls out completely, whereas TCP-LL performance continues to degrade gracefully as the channel capacity decreases.

Note also that a blockage fraction of, for example, 25% is different than a BER of 25% from the previous section. In both cases, one quarter of the bits experience errors, but in the case of the blockage channel, those errors are localized in time, allowing much greater channel capacity. In the measurements shown, the mean blockage duration (d) is varied between 200ms and 5000ms, but as described in the next section, this parameter has a relatively small effect on throughput as long as it stays significantly higher than a frame duration.

In order to understand why TCP-LL performs so much better on the blockage channel, we can run both protocols on a simplified blockage channel, in which blockage events are fixed-length and periodic. Figure 4-5 depicts TCP and TCP-LL throughput behavior on such a channel, with 5 second long blockages. Initially, TCP-LL under performs TCP slightly due to its increased frame header overhead. However, as soon as the first blockage event occurs, TCP-LL jumps ahead and each blockage event places it farther ahead of TCP. Note that the first TCP-LL blockage recovery is

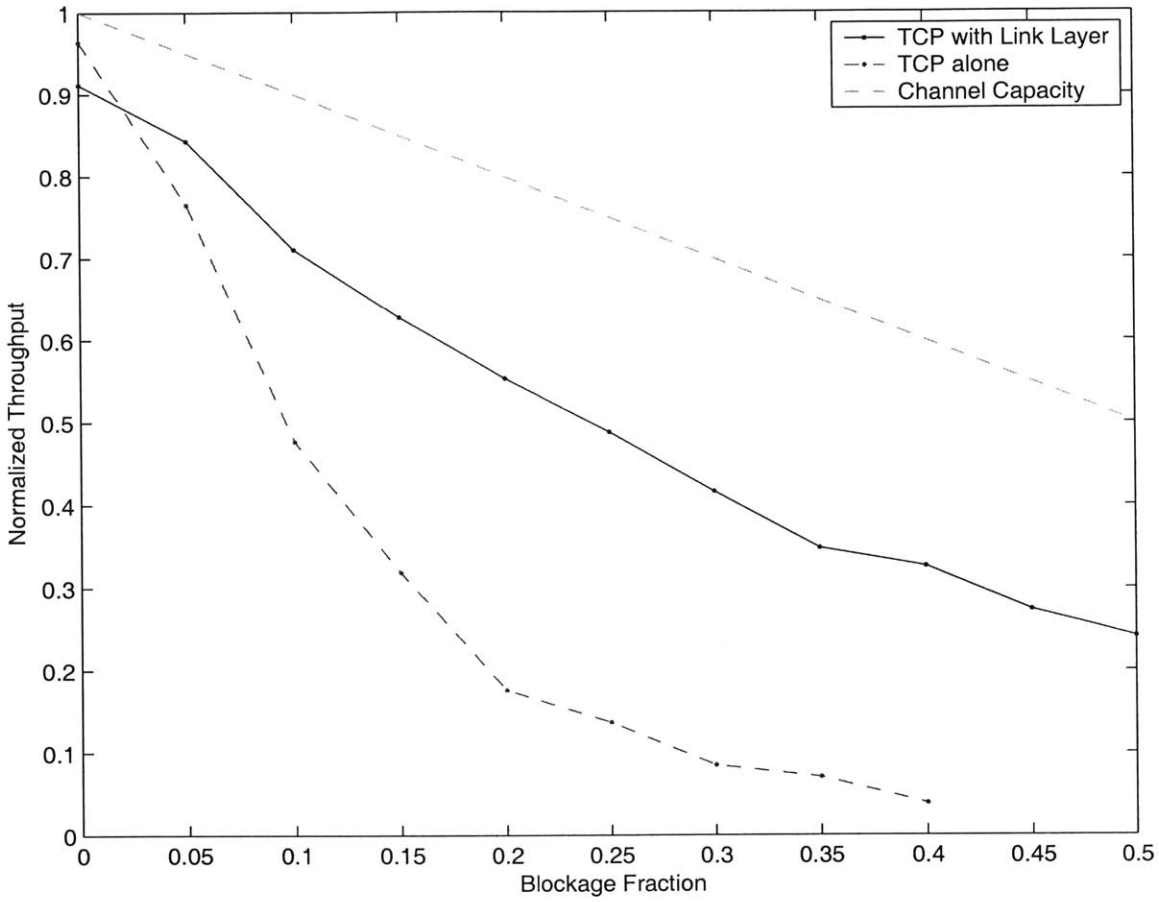


Figure 4-4: Comparison of TCP performance with and without link layer ARQ on blockage channels with increasing probability of blockage. Note that this is the baseline configuration - TCP-LL performance can be improved further through careful tuning.

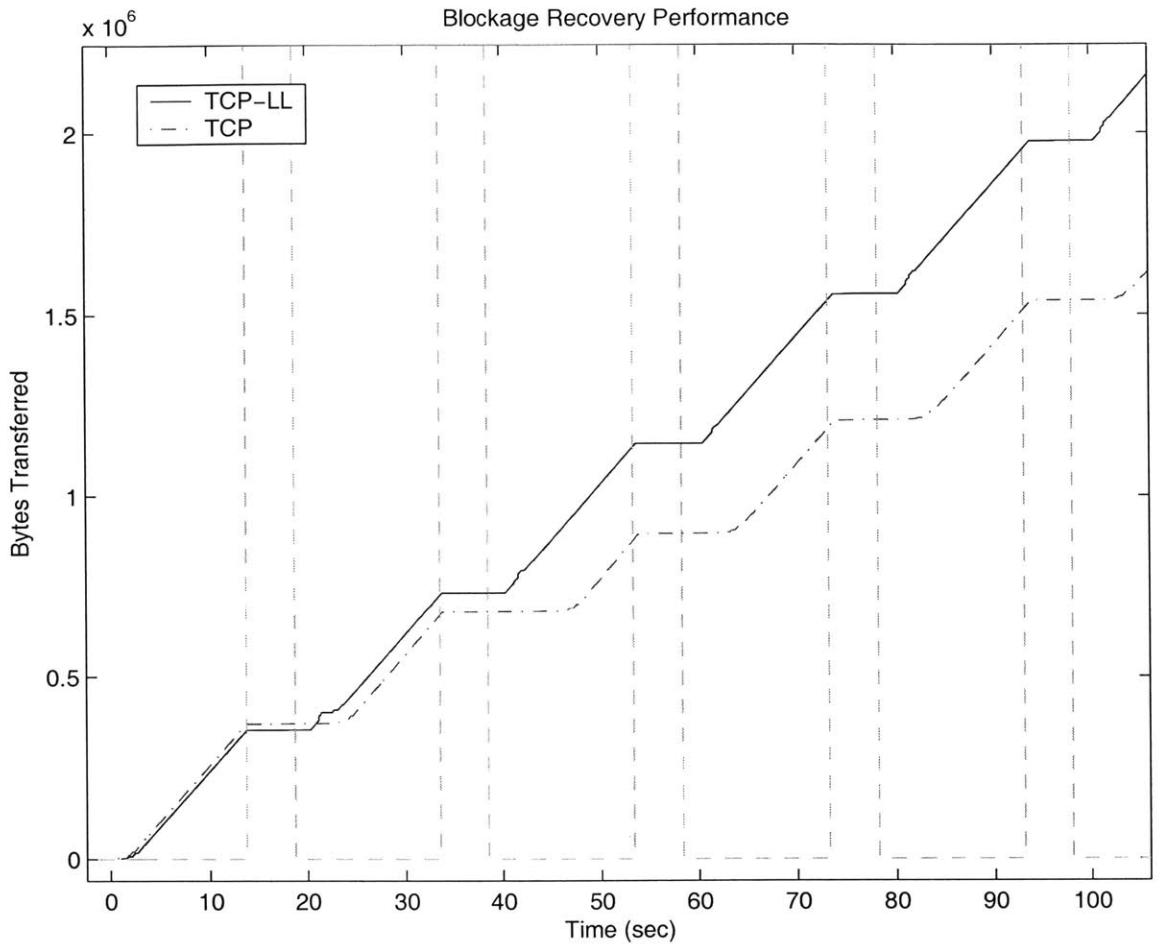


Figure 4-5: TCP and TCP-LL blockage recovery behavior. The dashed lines indicate the start and end of blockage events. For a closeup of the first blockage event, see Fig. 4-6.

slower than subsequent recoveries. This is because the TCP round trip time (RTT) estimate has not yet adapted to the added latency of link layer ARQ, and TCP triggers a full go-back-N retransmission. Once the RTT estimate has adapted, TCP-LL recovers faster because TCP waits for the link layer to perform retransmission, without performing retransmissions of its own.

Figure 4-6 is a closer view of the first blockage event. TCP-LL begins to recover from blockage about twice as fast as TCP, because the link layer retransmissions rapidly deliver queued data as soon as the link becomes unblocked. The sudden delivery of many packets (which include TCP ACKs) triggers new TCP transmissions,

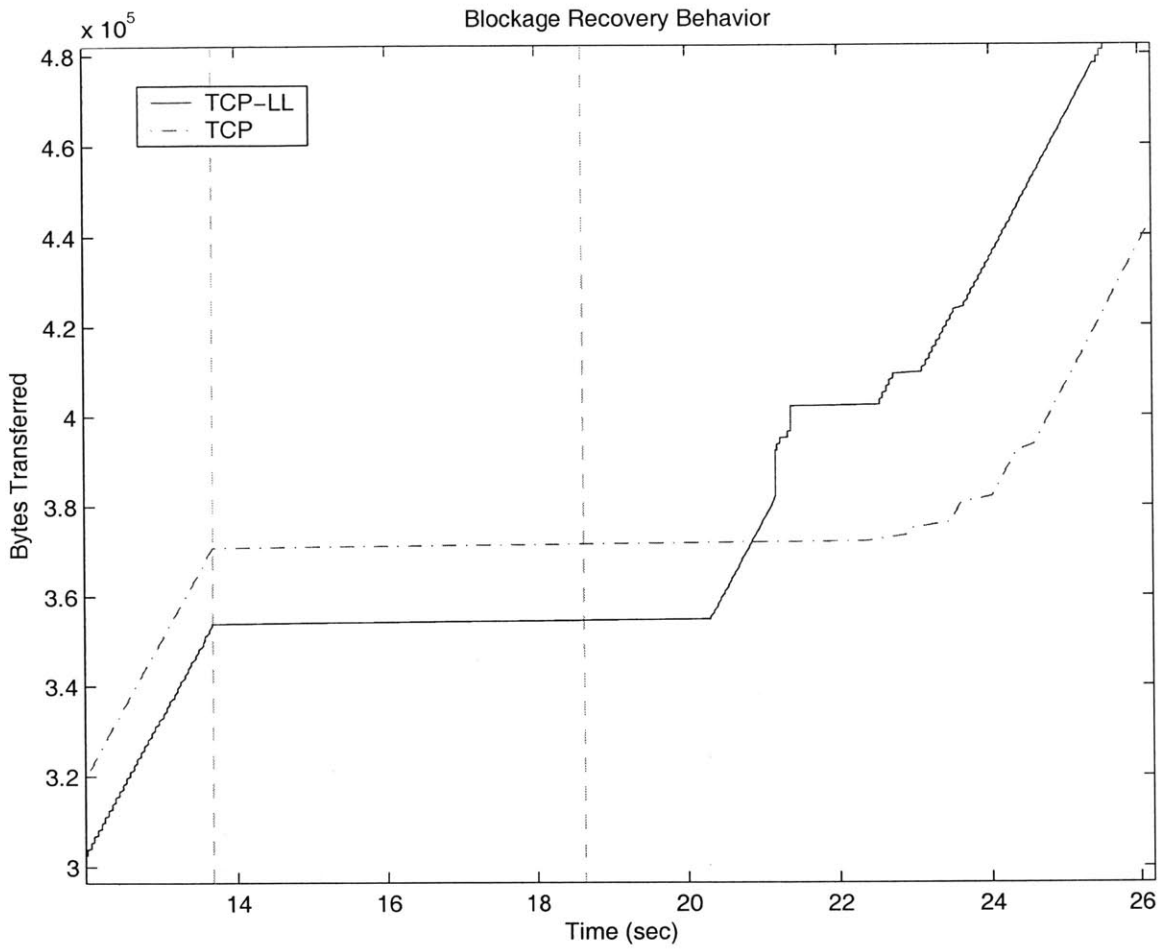


Figure 4-6: Closeup of first blockage event from figure 4-5.

kick starting the connection.

In order to understand the retransmission behavior of both the link layer and TCP, we can examine the order of sequence numbers sent and received. Fig. 4-7 contains sequence number traces of both layers during a blockage event. When blockage occurs, TCP stops sending new packets because it receives no acknowledgements. The link layer below it, however, continues to cycle through the window of outstanding frames. When the blockage clears, the link layer delivers queued packets (which contain ACKs), which in turn trigger new TCP transmissions. Gaps in the receive window are remedied through link layer block acknowledgements, and the connection continues as before blockage.

In this example, TCP times out and probes the link with several single packet

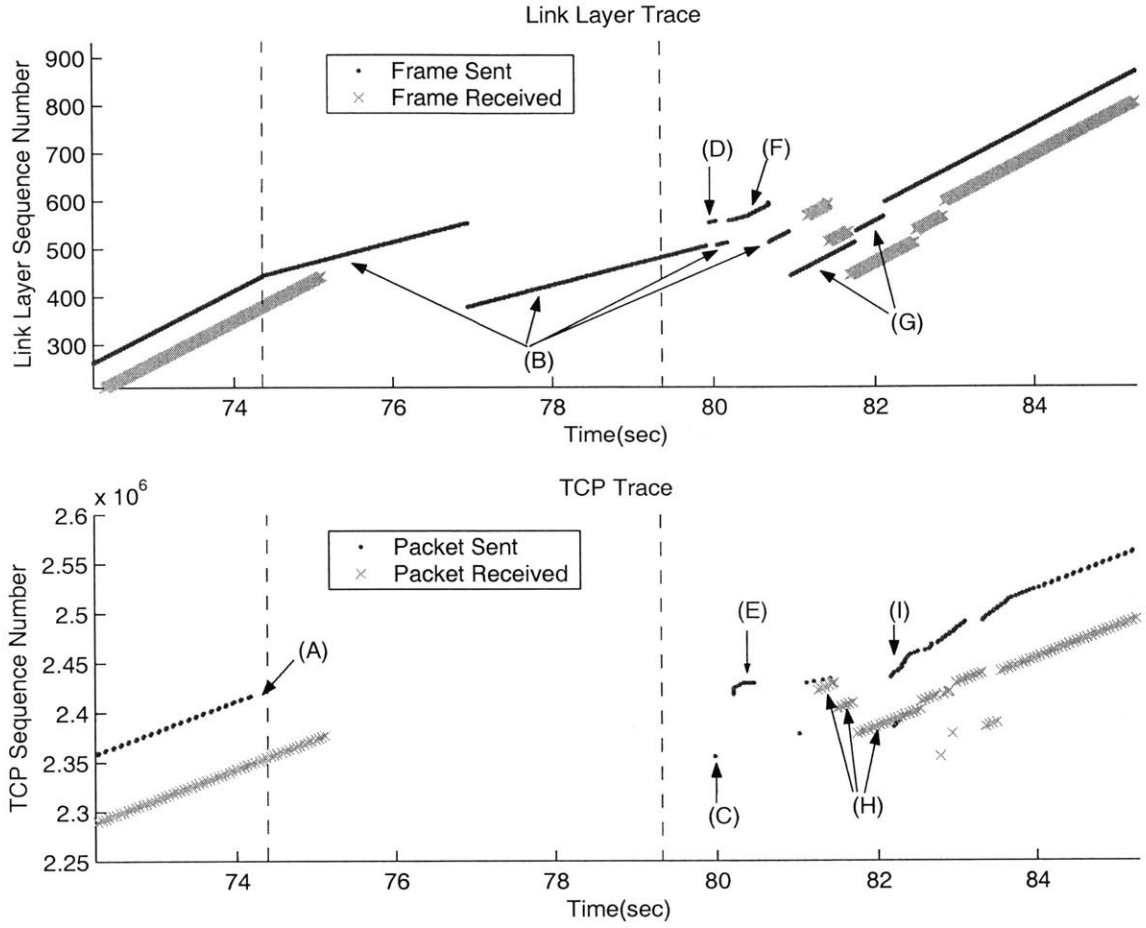


Figure 4-7: Traces of link layer and TCP sequence numbers during a TCP-LL blockage recovery. (A) TCP stops transmitting because it is no longer receiving ACKs. (B) The link layer cycles through its window retransmitting outstanding frames. (C) TCP probes the link with a one packet retransmission. (D) The TCP retransmission appears as new data to the link layer, which sends it immediately and then resumes cycling. (E) The delivery of queued ACKs by the link layer (running on the reverse link, so not depicted here) causes TCP to send new data, (F) which is immediately sent by the link layer. (G) Link layer block acknowledgements trigger the retransmission of missing frames. (H) The receipt of missing packets causes TCP to generate new ACKs. (I) The receipt of new ACKs causes TCP to start flowing again.

retransmissions. However, note that the total amount of data retransmitted by TCP is small relative to the amount of data effectively retransmitted by the link layer. On shorter blockages, the link layer does even better, and TCP does not have a chance to time out and retransmit. TCP's exponential back-off algorithm ensures that little bandwidth is wasted on unnecessary TCP retransmissions that the link layer can handle more quickly.

4.2 Sensitivity to Channel Parameters

This section explores the effects that the channel parameters have on throughput and latency. We conclude that blockage fraction f is the dominant factor in determining throughput in all practical situations, whereas f and blockage duration d both contribute to latency, and d's effect on latency is load dependent. Finally, we describe the effects of bandwidth asymmetry.

4.2.1 Effects on Throughput

Figure 4-8 shows the variation in TCP-LL throughput as both blockage fraction (f) and blockage duration (d) are varied. Fig. 4-9 is a similar plot using TCP instead of TCP-LL. The key result here is that blockage fraction is the dominant parameter, whether we use link layer ARQ or not. Blockage duration, at least within the range 200-5000ms, has little impact on throughput. However, it is clear that blockage duration must have an effect within *some* range, because as blockage duration approaches zero, the blockage channel should approach the behavior of a binary symmetric channel with BER $= \frac{f}{2}$. By exploring shorter and shorter time scales, this effect can be detected, as in fig. 4-10. From this we can conclude that mean blockage duration has a minor impact on throughput, *unless* it approaches the length of a single frame.

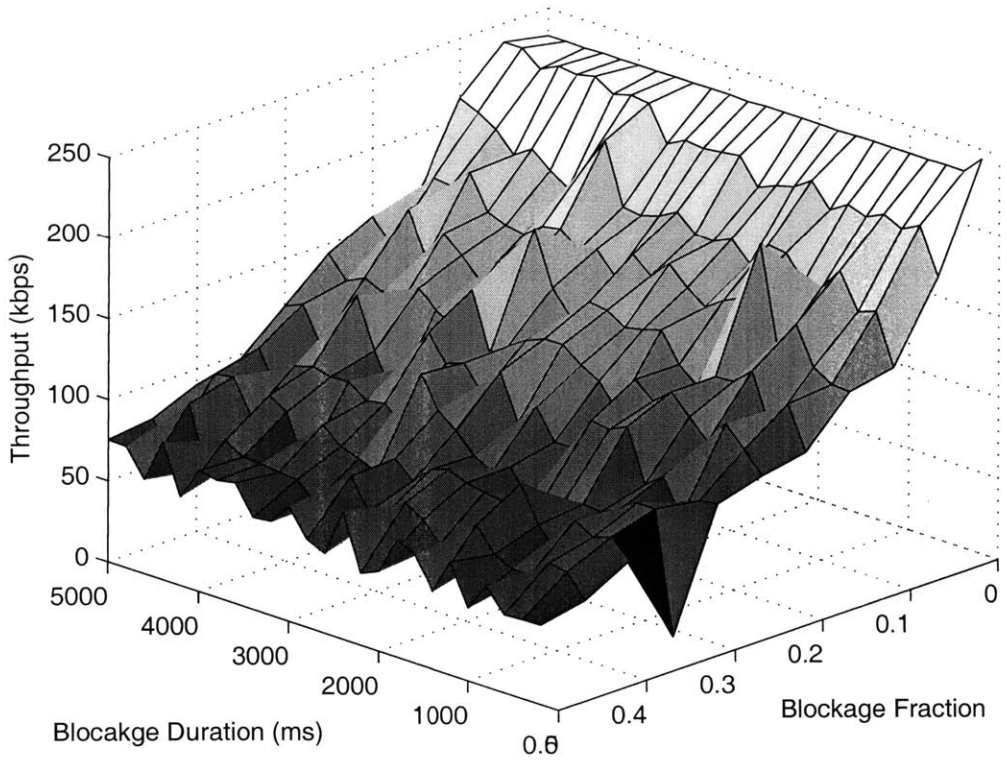


Figure 4-8: TCP-LL throughput performance on a variety of blockage channels. Note that blockage fraction is the dominant parameter, whereas mean blockage duration has no readily discernible impact within this range. No averaging was performed on these measurements.

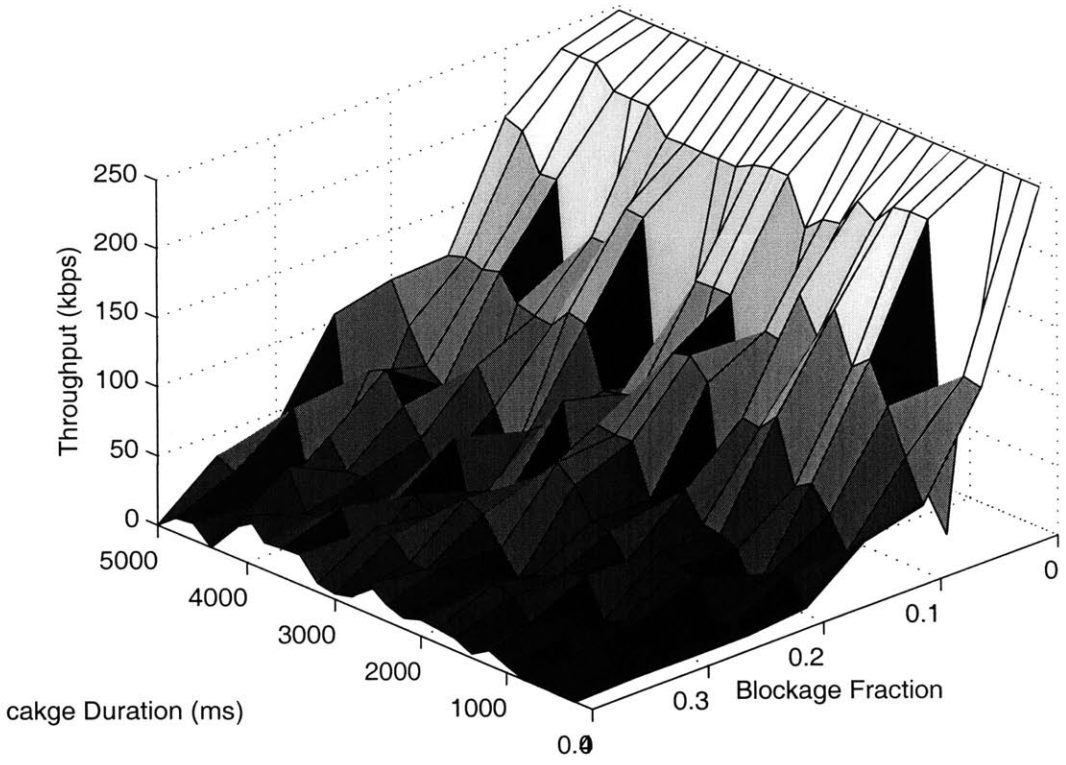


Figure 4-9: TCP throughput performance on a variety of blockage channels. TCP performance falls off much faster than TCP-LL with increasing f. Note that without the link layer, the measurements are noisier because of higher variability in throughput.

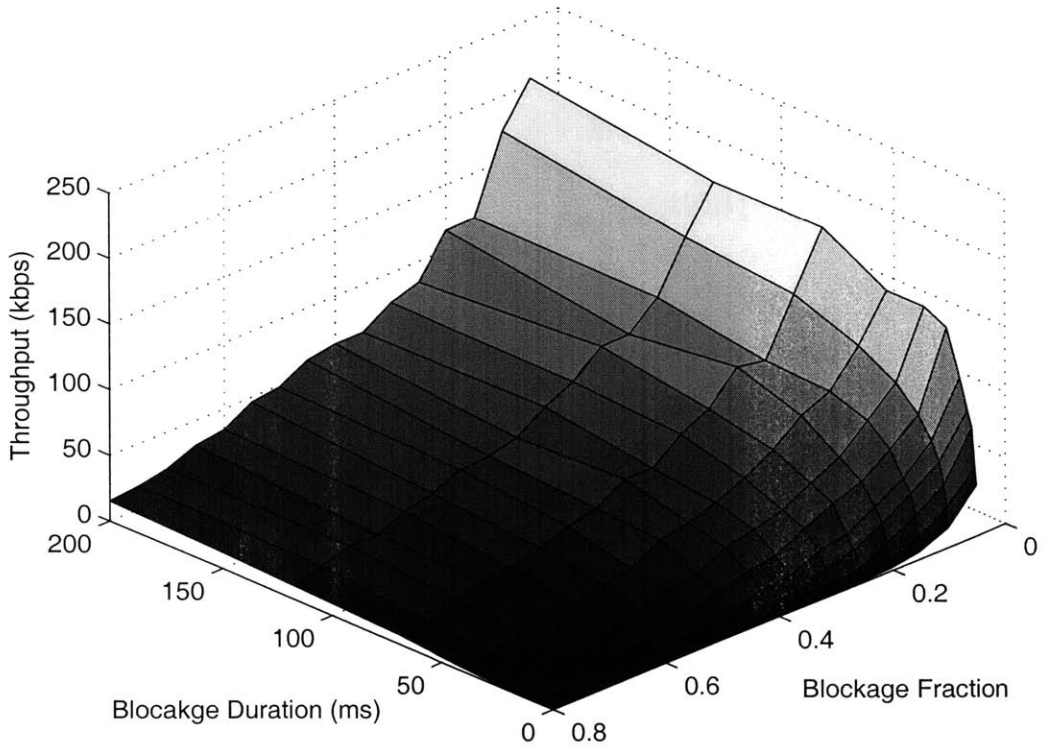


Figure 4-10: TCP with link layer throughput performance on channels with short mean blockage durations. The frame size is approximately 4000 bits, which corresponds approximately to the knee of the curve.

4.2.2 Effects on Latency

Link layer ARQ achieves greater throughput at the cost of increased latency. The throughput measurements in the previous sections were taken by running at the highest throughput possible. To study the relationship between throughput and latency, we run a rate-controlled source and measure the mean latency. Note that the tests are of fixed length (5 minutes) and therefore the latency is artificially bounded. If load exceeds real capacity, latency will grow without bound as the length of the test increases. Also note that we are measuring application level latency, which includes queuing delays at both the link layer and transport layer.

Figure 4-11 shows this relationship as the blockage fraction varies. Mean blockage duration is held constant at 1 second. The chart demonstrates that mean latency increases approximately exponentially as load increases. The offered load is normalized by the channel capacity, so that more highly blocked channels are given correspondingly lighter loads. However, highly blocked channels still experience higher latency. This shows that the system makes less efficient use of capacity at higher blockage fractions.

Figure 4-12 shows load/latency curves as mean blockage duration varies, while blockage fraction is held constant at 10%. At low offered load, channels with shorter blockages experience lower latency. However, as load increases to between 60% and 70%, the situation inverts and channels with longer blockages achieve lower latency.

To explain this inversion, we must consider the dominant sources of latency in each regime. At low load, most packets experience a single blockage event while they are waiting in the transmission queue, because each non-blocked period is sufficient to eliminate backlog. In contrast, at high load most packets experience many blockage events while they are waiting in the transmission queue, and each blockage event incurs a fixed overhead to get the link flowing again. Therefore, the low-load behavior is dominated by the duration of the blockages, while the high-load behavior is dominated by the number of blockages (and channels with lower d experience more blockages, since f is held constant).

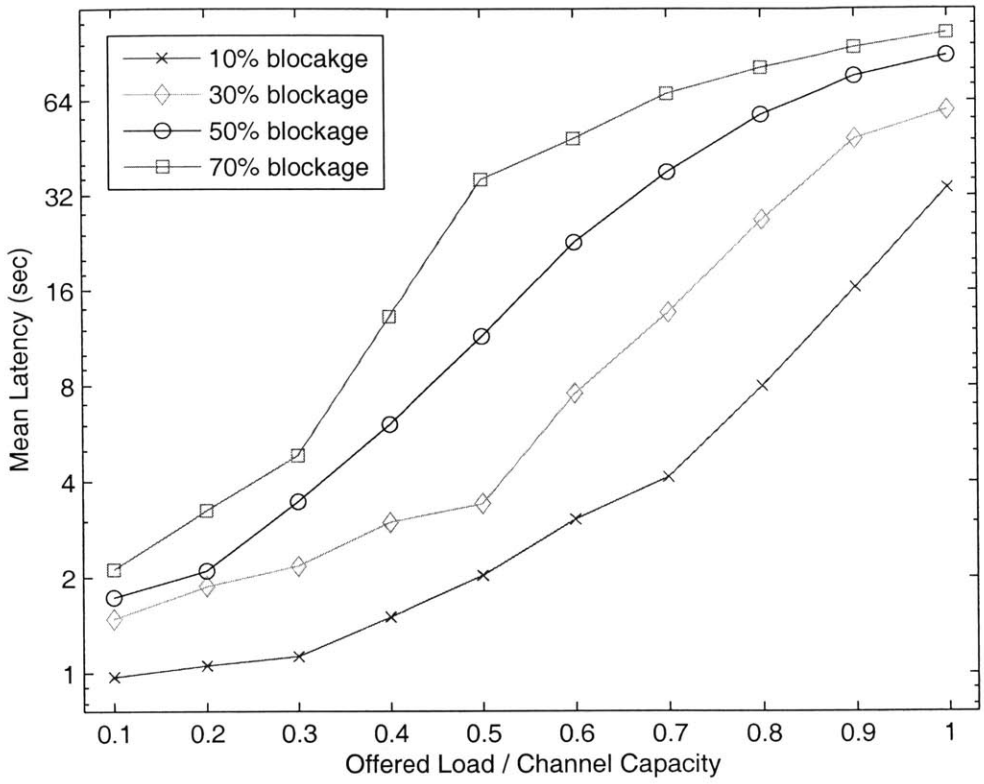


Figure 4-11: The horizontal axis measures the ratio between offered load and channel capacity. Note that the channel capacity for a highly blocked channel is inherently lower than the capacity of a less blocked channel, but this effect is normalized out. Also note that the vertical axis is on a logarithmic scale, implying that the latency grows exponentially as load increases. Each data point is the mean of 8 measurements. Mean blockage duration is held constant at 1 second.

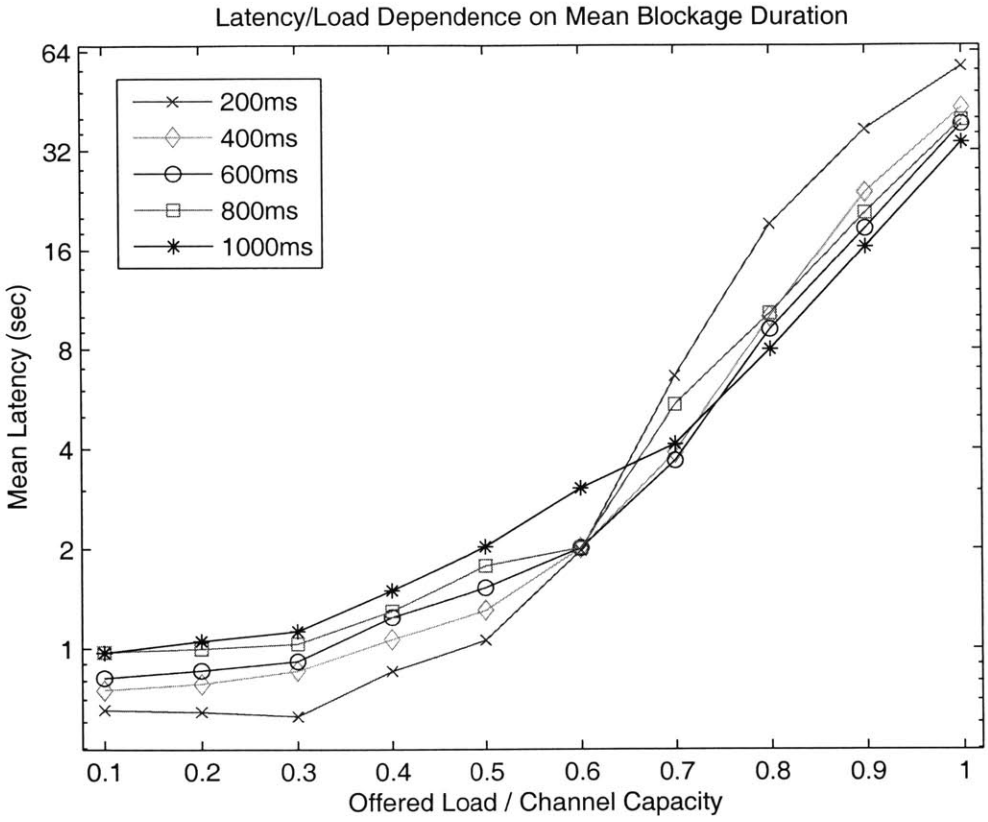


Figure 4-12: At low load, quickly varying channels achieve lower latency. However, at high load, slowly varying channels achieve lower latency. Blockage fraction is held constant at 10%

For a more detailed picture of latency behavior, see Fig. 4-13. It compares the blockage recovery behavior of two connections on the same simple periodic blockage channel. One connection is run at full channel capacity, the other at 50% channel capacity. From the figure it is clear that TCP is able to utilize excess capacity to catch up from blockage, if excess capacity is available, by increasing its rate. However, the increase is clearly not instantaneous, which explains the latency overhead associated with each blockage recovery. This overhead is also why a connection cannot run at 100% of capacity with bounded latency.

4.2.3 Bandwidth Asymmetry Effects

Another aspect of the channel that can seriously impact performance is bandwidth asymmetry. In order to measure the effect of bandwidth asymmetry, we vary the rate of one direction of the link (hereafter referred to as the uplink) while holding the rate of the other direction (the downlink) constant. We measure the throughput achievable on the downlink.

TCP acknowledgement messages cost 62 bytes, because the IP header is 20 bytes, the TCP header is 32 bytes, and the link layer frame header is 10 bytes. Assuming the underlying hardware MTU is 1500 bytes (standard Ethernet), each data packet can carry 1438 bytes. Therefore, 1438 bytes received requires 62 bytes for acknowledgement. This is a symmetry factor of $62/1438 = 4.3\%$. We therefore expect TCP performance to be unaffected until the symmetry factor falls below this level.

Figure 4-14 shows the asymmetry experiment results. On a channel with no blockage, TCP performs as expected by maintaining full throughput as long as symmetry stays above 4.3%. However, on the three different blockage channels tested, performance degrades much more quickly. Unlike the clear channel, the blocked channels have no hard-edged limit on asymmetry, because their performance is inherently stochastic.

Consider a symmetric link carrying TCP data on the downlink and ACKs on the uplink. The load per channel capacity on the uplink will necessarily be low, which therefore provides relatively low latency service as described in the previous section.

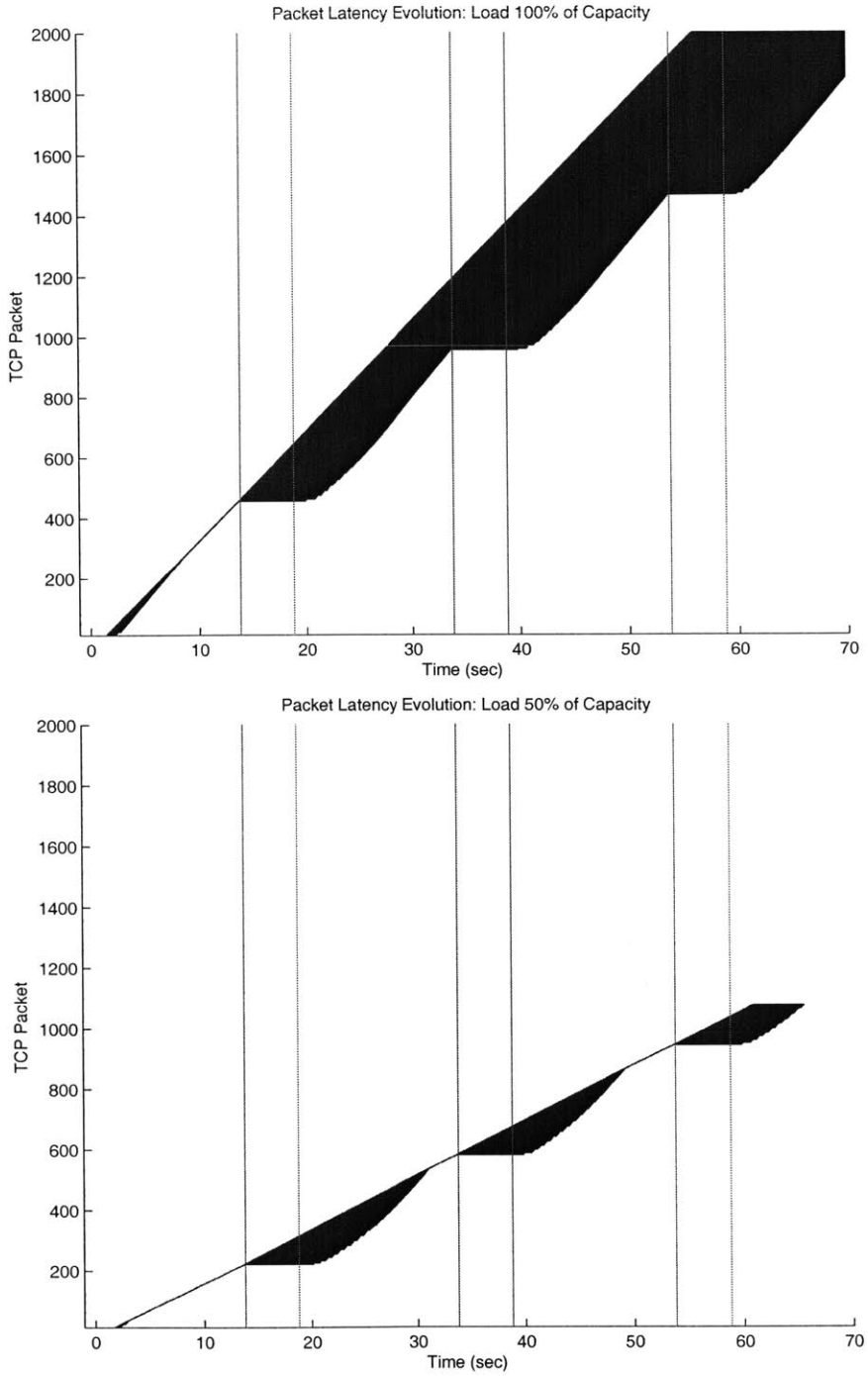


Figure 4-13: A periodic blockage channel loaded at 100% and 50% of channel capacity. Each packet is represented by a horizontal line running from its transmission time to its reception time. In the first example, latency grows large because there is no excess capacity to catch up from blockages. In the second example, the TCP congestion window ramps up after blockage to utilize excess capacity and eliminate backlog.

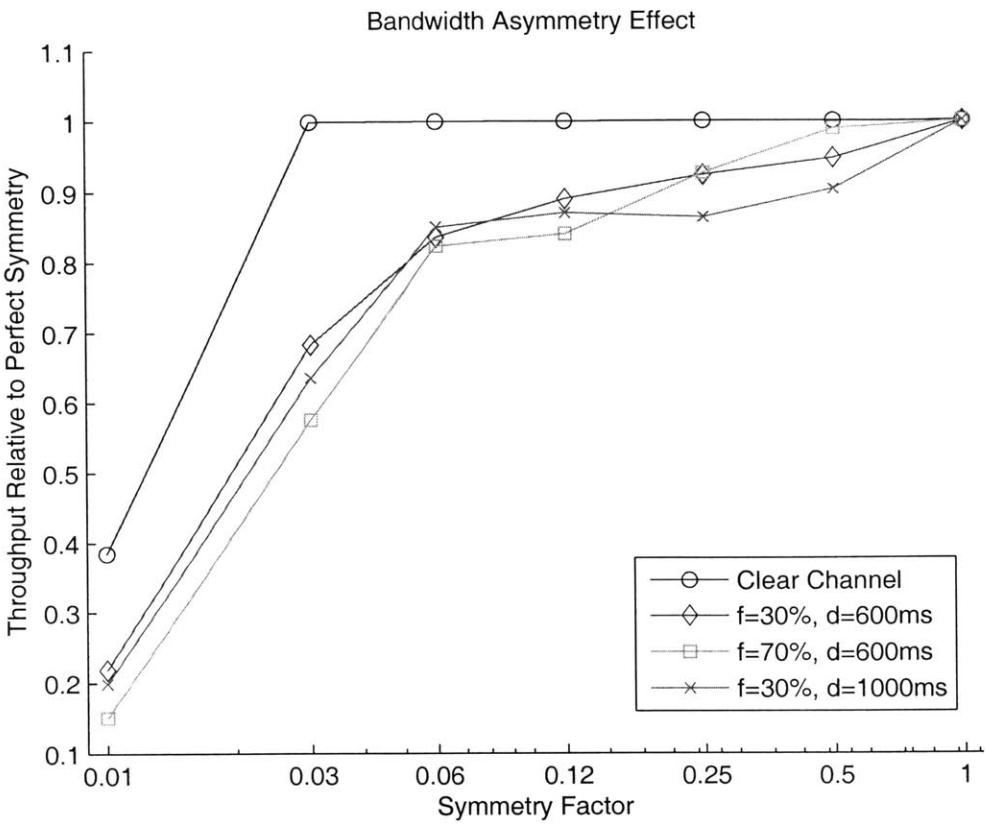


Figure 4-14: The effect of bandwidth asymmetry. The symmetry factor is the ratio between uplink and downlink bandwidth. The experiment measures downlink throughput. Each data point is the mean of 20 measurements. Each curve is normalized by its throughput at perfect symmetry, which allows us to compare the way throughput changes with asymmetry, rather than absolute throughput (which is highly dependent on f).

However, if the rate of the uplink is decreased its load per channel capacity increases, causing a corresponding increase in the latency. Therefore, TCP ACKs experience a higher latency, which causes the TCP congestion window to grow more slowly, which impedes blockage recovery. This explains why the blocked channels are more adversely effected by asymmmetry.

4.3 Sensitivity to Protocol Parameters

This section explores the effects of design parameters on link layer performance. Specifically, we attempt to optimize the link layer window size and the packet reordering strategy.

4.3.1 Link Layer Window Size

The effect of link layer window size is a complex one. We would expect the optimal window size to depend on the bandwidth-delay product experienced by TCP. However, the window size is also one of the major factors that determines the bandwidth-delay product experienced by TCP. Without a more detailed analytical understanding of TCP, it is not clear how to choose an optimal size, except by experiment.

Figure 4-15 shows how throughput depends on link layer window size for two channels with 30% blockage. The data rate is 256kbps and the one way latency is 250ms. Both channels achieve maximum throughput with a window size of about 1.4 seconds, corresponding to 46kB. From these data, it appears that blockage duration has no effect on the optimal window size. Fig. 4-16 shows a similar graph for a channel with 70% blockage. In this case, the optimum window size has decreased to 0.6 seconds, or 20kB. Therefore it appears that a higher blockage fraction requires a smaller window. This might seem counterintuitive, since we would expect a more highly blocked channel to require larger buffers. The key factor here is TCP behavior.

The difference in optimal window size between the 30% and 70% blocked channels can be accounted for by the different channel capacities. When the channel is 30% blocked, the optimal window size of 46kB (or 1.4 seconds at channel rate) takes an average of

$$\frac{1.4}{1-30\%} = 2.0$$

2 seconds to transmit successfully, due to blockage. Similarly, when the channel is 70% blocked, the optimal window size of 20kB takes an average of

$$\frac{0.6}{1-70\%} = 2.0$$

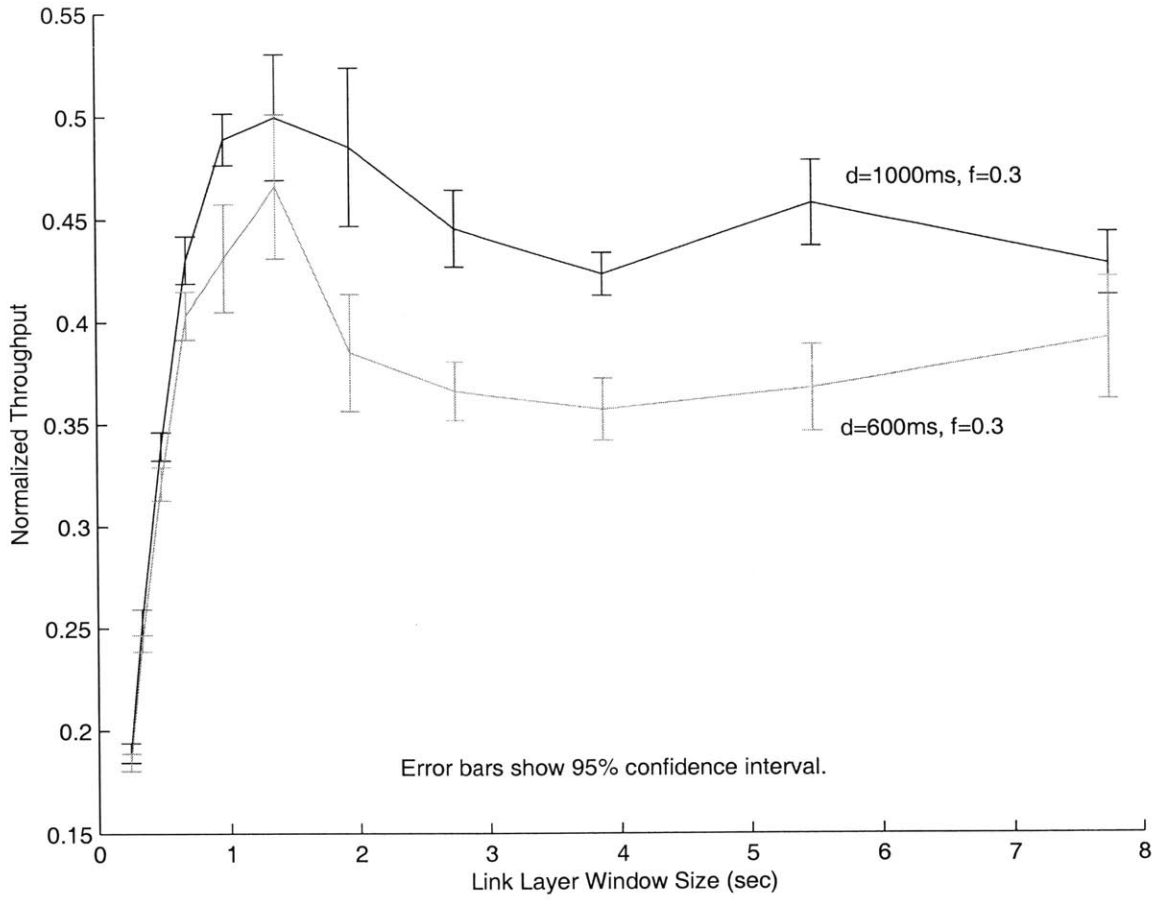


Figure 4-15: The effect of link layer window size. Both channels have the same blockage fraction, and both exhibit the same optimal window size. The channel with a shorter blockage duration has a sharper maximum. Window size is measured in seconds of transmission buffered. The channel rate is 256kbps and the frame size is 496 bytes, so 1 second corresponds to about 66 frames.

2 seconds to transmit. In both cases, throughput is maximized when the link layer queuing delay is 2 seconds.

To see if this value depends on channel latency, the experiment was repeated with double the channel latency. While overall throughput was lower (as expected. due to the increased effect of slow start), the optimal window size did not change. It is not clear what factors influence this optimal size, and further experiments are warranted.

Figure 4-15 also contains a clarification of a previous result. In section 4.2.1, we claimed that blockage duration d has a negligible effect on throughput. That result was found using the baseline configuration, which sets the link layer window size to

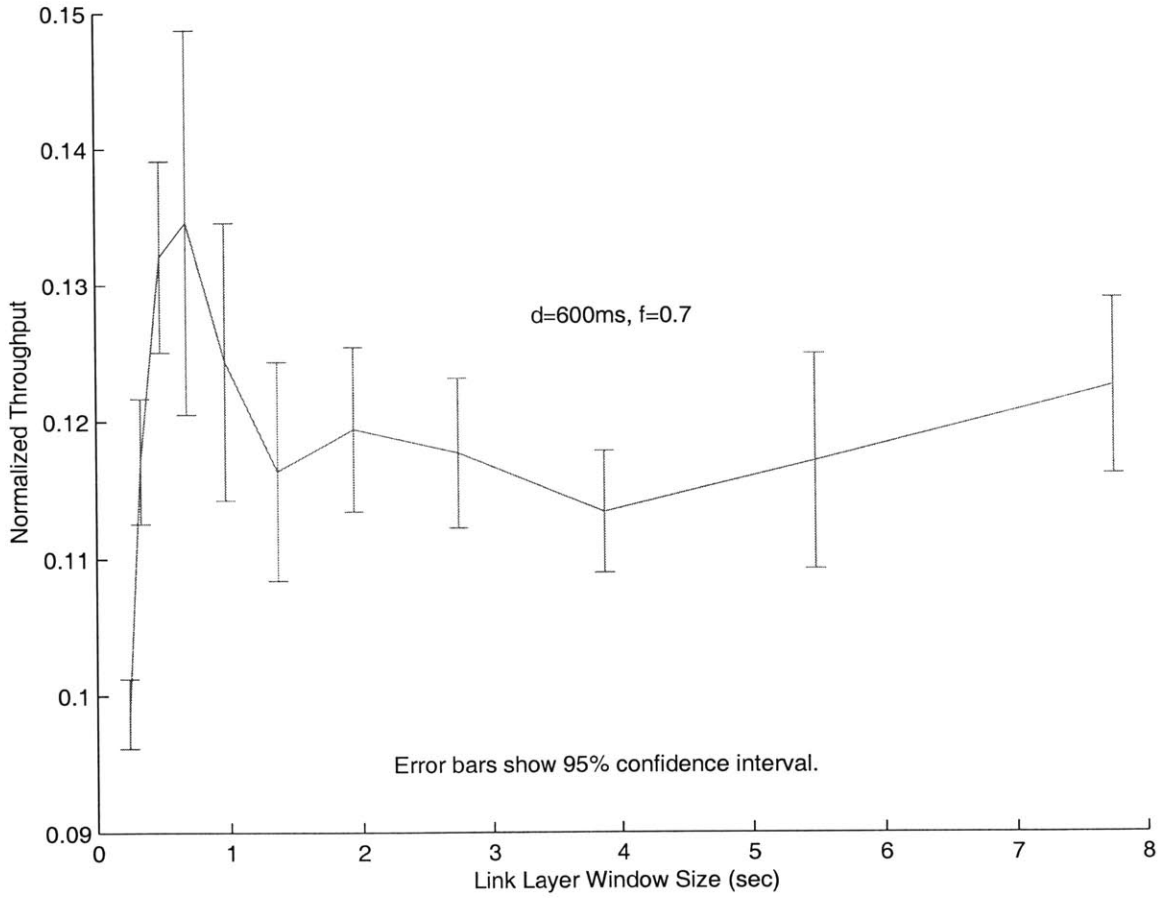


Figure 4-16: The effect of link layer window size at 70% blockage. Unlike Fig. 4-15, we only depict the $d = 600$ channel because the $d = 1000$ channel overlays it very closely.

512 frames, which in this context is 8 seconds. While it is true that blockage duration has a small effect on a window this large, Fig. 4-15 shows that this result is not true in general. For some window sizes, especially 2-5 seconds, blockage duration *does* have a measurable effect on throughput, although it is still much smaller in absolute terms than the effect of blockage fraction.

4.3.2 Packet Reordering

When TCP receives several packets out of order, its fast retransmit algorithm is activated. When using link layer ARQ, this causes unnecessary retransmissions. It is therefore desirable for the link layer to preserve packet order, by actively ordering packets that arrive out of order. However, preserving packet order can cause much higher latency and latency variation as later packets must wait for the reception of earlier packets before being delivered to TCP. This added latency could negatively impact TCP throughput.

Figure 4-17 shows results of an experiment to determine which effect is dominant. In all channels tested, the link layer provided better throughput when it guaranteed in-order delivery. Therefore we can conclude that TCP fast retransmissions are the dominant effect.

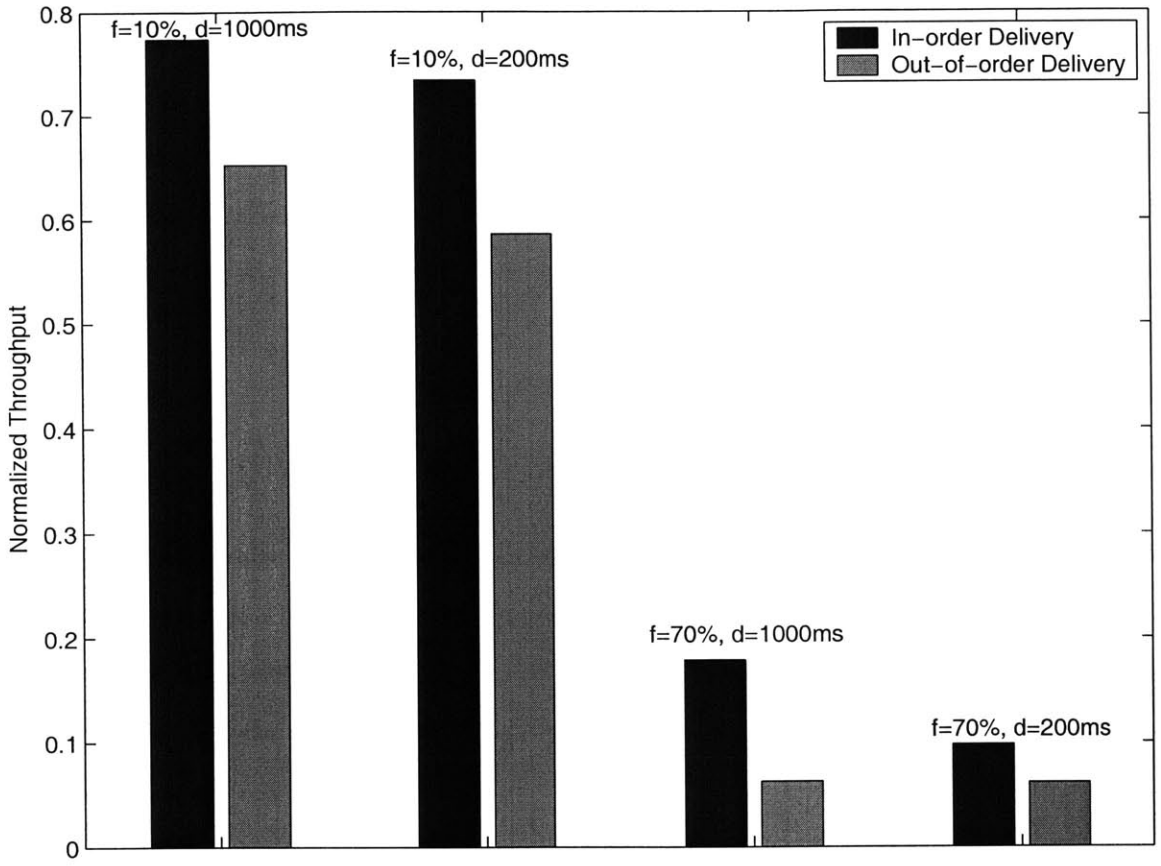


Figure 4-17: TCP throughput on several channels. For each channel, the left bar indicates throughput with TCP-LL providing in-order delivery, and the right bar indicates throughput with TCP-LL not necessarily providing in-order delivery. Clearly, packet ordering improves throughput.

Chapter 5

Summary and Key Results

5.1 Summary

This thesis describes a link-layer Automatic Repeat Request (ARQ) protocol designed to carry TCP traffic on vehicle-mounted satellite links. The protocol was implemented at Lincoln Laboratory and tested in emulation. It is currently being tested on the air, but results are not yet available.

In Chapter 1, we outlined the mobile satellite communications problem and the motivation for studying TCP performance. We argued that ARQ is the most promising technique for improving TCP performance, based on our reliability and security requirements, and we surveyed previous wireless link layer designs.

In Chapter 2, we described our protocol design in detail. We stated the design assumptions, which included a circuit-switched physical layer, in-order frame delivery at the physical layer, IP layer encryption, frequent channel blockages, and possible bandwidth asymmetry. Key design decisions were highlighted, including the choice of link layer buffer size, whether or not to preserve frame order, and the appropriate level of persistence.

In Chapter 3, we described our experimental methods. This included the XPS software architecture for kernel-mode protocol testing, the choice of the Gilbert-Elliot model for channel errors, the selection of channel parameters, and the choice of network traffic models. We also described the performance metrics we deemed salient

and the strategies we used to record and interpret them.

In Chapter 4, we presented experimental results. First, we compared the throughput performance of TCP with and without link layer ARQ on binary symmetric channels, and then on Gilbert-Elliot blockage channels. We concluded that employing ARQ provides substantial improvements under all realistic conditions. Next, we studied the effects of the channel parameters on TCP throughput and latency, and concluded that blockage fraction (the proportion of time the channel is blocked) is the dominant factor, while mean blockage duration has a much smaller effect. We also showed that bandwidth asymmetry has a stronger impact on blockage channels than on clean channels. Finally, we studied the effects of protocol design decisions, including the choice of link layer buffer size and the decision to allow or disallow packet reordering.

5.2 Key Results

1. We demonstrated that blockage fraction is the dominant channel parameter, as opposed to blockage duration. Blockage duration only begins to have an effect if it approaches the duration of individual packets, which is unlikely to occur on land-mobile links. This suggests that network designers should focus on minimizing blockage fraction.

2. We provided latency vs load data that can serve as a planning guide for efficiently provisioning networks with desired latency and bandwidth. For example, in order to achieve 2 second TCP latency with a throughput of 50kb in a 50% blocked environment, the designer must provision at least 250kb link capacity.

3. We showed that bandwidth asymmetry limitations are stricter on blocked channels than on clean channels. Since bandwidth asymmetry is common in wireless environments, this effect can be important to network designers.

4. We demonstrated empirically that more highly blocked channels have smaller optimal buffer sizes. In all scenarios studied here, the optimal buffer size pro-

vided approximately 2 seconds of queuing delay at the link layer, but future experiments are needed to clarify exactly why. We do not have an adequate explanation for this phenomenon.

5. We showed that providing in-order delivery at the link layer improves TCP throughput. Further experiments would be useful to understand the interactions between multiple TCP sessions.

6. We observed that bandwidth wasted by needless TCP retransmissions is relatively small and unimportant. TCP's congestion control algorithm is designed to avoid adding to congestion, and TCP is therefore conservative about retransmitting without positive feedback from the receiver. TCP adapts to the presence of added link layer latency and allows the link layer to handle retransmissions, without making competing retransmission.

7. Based on the previous point, we believe that a limited persistence link layer would not provide significant benefit over an unlimited persistence link layer. When a link becomes blocked, all the TCP senders using it will stop transmitting. Yet in order to restart the link quickly, the link layer must have a large amount of TCP traffic to deliver. Therefore, the link layer should retain previous TCP packets and not drop them. If instead the link layer allows such packets to drop after several failed delivery attempts, a long blockage would cause the link layer window to become empty. When the link later becomes available it will sit idle until TCP retransmissions probed the link and discover it open. This wastes significant time and eliminates one of the key benefits of link layer ARQ: fast recovery.

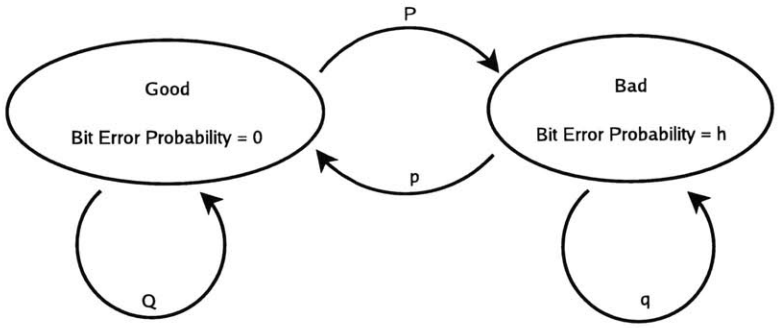
As described above, this thesis provides several useful results that should inform future system designs. These results apply to current "Milstar on the Move" efforts at Lincoln Laboratory, and should be useful to link designers in general. The most important extension to this work would be to relax the assumption of a circuit-switched physical layer. This would open several new and important questions concerning media access control and distributed network planning.

The overarching recommendation of this work is that system designers should consider new and creative ways of avoiding signal blockage. For example, sharing many links within a group of vehicles with a fast local network could provide drastically improved satellite connectivity in some environments.

Appendix A

Channel Capacity Analysis

The capacity of a burst-noise channel is derived by Gilbert in [18]. He defines the two-state Markov error channel like this:



where p,P,q, and Q are transition probabilities, and h is the probability of a correct bit in the bad state. The good state experiences no bit errors. Gilbert's analysis provides the following set of equations for calculating the channel capacity.

Channel capacity C depends on channel entropy H:

$$C = 1 - H$$

Channel entropy is found by summing the entropies for every possible sequence of bit errors. For this purpose we define a sequence to be a 1 (a bit error) followed by any number of zeros. We define $h(10^K)$ as the entropy of the sequence with K zeros . The entropy of each sequence is weighted by the probability of the sequence occurring, $P(10^K)$:

$$H = \sum_{K=0}^{\infty} P(10^K) h(10^K)$$

If we define u(K) to be the probability of K successive zeros, we can rewrite $P(10^K)$:

$$P(10^K) = P(1)u(K)$$

$P(1)$ is the probability of being in the bad state times the probability of getting a bit error given the system is in the bad state:

$$P(1) = (1-b)\frac{P}{p+P}$$

A recursive solution for $u(K)$ is

$$u(K) = (Q+hq)u(K-1) + h \cdot (p-Q)u(K-2)$$

$$u(0) = 1$$

$$u(1) = p + hq$$

The sequence entropy is given by:

$$h(10^K) = -\frac{u(K+1)}{u(K)} \log_2 \frac{u(K+1)}{u(K)} - \left[1 - \frac{u(K+1)}{u(K)}\right] log_2 \left[1 - \frac{u(K+1)}{u(K)}\right]$$

This completes Gilbert's solution for channel capacity. We are interested in the limiting case where the blockage duration is very long relative to the length of the individual bits. As blockage duration approaches infinity (holding blockage fraction constant), P and p approach 0, so Q and q approach 1. Therefore, in the limit:

$$u(K) = h^K$$

Another simplifying assumption we can make is that the bad state is as bad as possible, with $h = 0.5$. This causes the sequence entropy to simplify greatly:

$$h(10^K) = 1$$

The channel entropy now simplifies to:

$$H = \frac{1}{2}\frac{P}{p+P}\sum_{K=0}^{\infty}\left(\frac{1}{2}\right)^K = \frac{P}{p+P}$$

So the channel capacity is:

$$C = 1 - \frac{P}{p+P}$$

Using the definition of blockage fraction given in Chapter 3, this can be rewritten:

$$C = 1 - f$$

Therefore, blockage fraction has a very clear relationship to channel capacity. In practice this approximation holds very well.

Bibliography

[1] RFC 2760: Ongoing TCP research related to satellites, Feb 2000.

[2] Advanced Encryption Standard. Technical Report FIPS PUB. 197, National Institute of Standards and Technology, 2001.

[3] E. Amir, H. Balakrishnan, S. Seshan, and R. Katz. Efficient TCP over networks with wireless links. In *Hot Topics in Operating Systems, 1995*, pages 35–40. IEEE, May 1995.

[4] F. Anjum and R. Jain. Performance of TCP over lossy upstream and downstream links with link-level retransmission. In *IEEE International Conference on Networks*, pages 3–7. IEEE, 2000.

[5] F. Anjum and L. Tassiulas. Comparative study of various TCP versions over a wireless link with correlated losses. *IEEE/ACM Transactions on Networking*, 11(3), June 2003.

[6] H. Balakrishnan and V. Padmanabhan. How network asymmetry affects TCP. *IEEE Communications Magazine*, pages 60–67, April 2001.

[7] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE Transactions on Networking*, 5(6), December 1997.

[8] H. Bischl and E. Lutz. Packet error rate in the non-interleaved Rayleigh channel. *IEEE Transactions on Communications*, 43(2), Feb 1995.

[9] I. Bisio and M. Marchese. Analytical expression and performance evaluation of TCP packet loss probability over geostationary satellite. *IEEE Communications Letters*, 8(4), Apr 2004.

[10] G. Briceno, D. J. Shyy, and J. Wu. A study of TCP performance for mobile satcom system over blocking conditions. In *Military Communications Conference*. IEEE, October 2003.

[11] H. Chaskar, T. Lakshman, and U. Madhow. TCP over wireless with link level error control: analysis and design methodology. *IEEE/ACM Transactions on Networking*, 7(5), Oct 1999.

[12] C. F. Chiasseri and M. Meo. Improving TCP over wireless through adaptive link layer setting. In *Global Telecommunications Conference*, volume 3, pages 1766–1770. IEEE, 2001.

[13] A. Chokalingam, M. Zorzi, and V. Tralli. Wireless TCP performance with link layer FEC/ARQ. In *IEEE International Conference on Communications, 1999*, volume 2, pages 1212–1216. IEEE, June 1999.

[14] D. Cygan and E. Lutz. A concatenated two-stage adaptive (CTSA) error control scheme for data transmission in time-varying channels. *IEEE Transactions on Communications*, 43(2), Feb 1995.

[15] E. O. Elliot. Estimates of error rates for codes on burst noise channels. *Bell Systems Technical Journal*, pages 1977–1997, September 1963.

[16] G. Fairhurst and L. Wood. RFC 3366: Advice to link designers on link automatic repeat request (ARQ), Aug 2002.

[17] N. Ghani and S. Dixit. TCP/IP enhancements for satellite networks. *IEEE Communications Magazine*, July 1997.

[18] E. N. Gilbert. Capacity of a burst noise channel. *Bell Systems Technical Journal*, pages 1253–1265, September 1960.

[19] J. Hagenauer and E. Lutz. Forward error correction coding for fading compensation in mobile satellite channels. *IEEE Journal in Selected Areas of Communications*, 5(2), Feb 1987.

[20] T. Hamann and J. Walrand. A new fair window algorithm for ECN capable TCP. In *IEEE INFOCOM 2000*, pages 1528–1536. IEEE.

[21] T. Henderson and R. Katz. Transport protocols for internet-compatible satellite networks. *IEEE Journal on Selected Areas in Communications*, 17(2):326–344, February 1999.

[22] T. Henderson, E. Sahouria, S. McCanne, and R. Katz. On improving fairness of TCP congestion avoidance. In *IEEE Globecomm 97*.

[23] A. Kumar. Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM Transactions on Networking*, 6(4), Aug 1998.

[24] T. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3), Jun 1997.

[25] G. Leerujikul and K. M. Ahmed. TCP over satellite link with SACK enhancements. In *IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, volume 2, pages 350–353, August 2001.

[26] E. Lutz, D. Cygan, M. Dippold, F. Dolainsky, and W. Papke. The land mobile satellite communication channel - recording, statistics, and channel model. *IEEE Transactions on Vehicular Technology*, 40(2), May 1991.

[27] M. Meyer, J. Sachs, and M. Holzke. Performance evaluation of a TCP proxy in WCDMA networks. *IEEE Wireless Communications*, October 2003.

[28] I. Minei and R. Cohen. High-speed internet access through unidirectional geostationary satellite channels. *IEEE Journal on Selected Areas in Communications*, 17(2), February 1999.

[29] J. Mineweaser, J. Stadler, S. Tsao, and M. Flanagan. Improving TCP/IP performance for the land mobile satellite channel. In *Military Communications Conference, 2001*, volume 1, pages 711–718. IEEE, October 2001.

[30] Y. Miyake, T. Hasegawa, and T. Kato. Acceleration of TCP throughput over satellite-based internet access using TCP gateway. In *Fifth IEEE Symposium on Computers and Communications*, pages 245–253, July 2000.

[31] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5), Oct 2000.

[32] J. Padhye, V. Firoui, D. Towsley, and J. Kursoe. Modelling TCP throughput: A simple model and its empirical validation. In *SIGCOMM 98*, 1998.

[33] S. Philopoulos and K. Ferens. Proxy-based connection-splitting architectures for improving TCP performance over satellite. In *IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1430–1435. IEEE, 2002.

[34] M. Rossi, L. Badia, , and M. Zorzi. Accurate approximation of ARQ packet delay statistics over Markov channels with finite round-trip delay. In *IEEE WCNC*, 2003.

[35] J. Schodorf. EHF satellite communications on the move: Experimental results. Technical Report 1087, MIT Lincoln Laboratory, August 2003.

[36] J. Scott and G. Mapp. Link-layer based TCP optimization for disconnecting networks. In *ACM SIGCOMM*, volume 33, pages 31–42, Oct 2003.

[37] P. Spagnolo, T. Henderson, and J. Kim.

[38] C.T. Spracklen. Digital communication procotols in the satellite environment. In *IEE Colloquium on 7 April, 1997.*

[39] J. S. Stadler. A link layer protocol for efficient transmission of TCP/IP via satellite. In *MILCOM 97 Proceedings*, volume 2, pages 723 -727. IEEE, November 1997.

[40] W. Turin and M. Zorzi. Performance analysis of delay-constrained communications over diverse burse-error channels. In *Vehicular Technology Conference 99*, volume 3, pages 1305–1309. IEEE, 1999.

[41] J. Wong and V. Leung. Improving end-to-end performance of TCP using link-layer retransmissions over mobile internetworks.

[42] M. Zorzi and R. Rao. Throughput of selective-repeat ARQ with time diversity in markov channels with unreliable feedback. *Wireless Networks*, 2(1):63–75, 1996.