Mitigating TCP degradation over intermittent link failures using intermediate buffers

THESIS

M. Brent Reynolds, Civilian, ND-04, US NAVY

AFIT/GIA/ENG/06-09

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GIA/ENG/06-09

Mitigating TCP degradation over intermittent link failures using intermediate buffers

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

M. Brent Reynolds, B.A.

Civilian, ND-04, US NAVY

June 2006

AFIT/GIA/ENG/06-09

Mitigating TCP degradation over intermittent link failures using intermediate buffers

M. Brent Reynolds, B.A.
Civilian, ND-04, US NAVY

Approved:

Maj Scott Graham, PhD (Chairman)	date
Dr. Ken Hopkinson (Member)	date
Capt Rich Beckman, PhD (Member)	date

AFIT/GIA/ENG/06-09

Abstract

This thesis addresses the improvement of data transmission performance in a challenged network. It is well known that the popular Transmission Control Protocol degrades in environments where one or more of the links along the route is intermittently available. To avoid this degradation, this thesis proposes placing at least one node along the path of transmission to buffer and retransmit as needed to overcome the intermittent link. In the four-node, three-link testbed under particular conditions, file transmission time was reduced 20 fold in the case of an intermittent second link when the second node strategically buffers for retransmission opportunity.

Acknowledgements

This product would not be possible with the love and support of my wife and my three children. For me, they bravely transplanted their lives for two years.

This opportunity was possible because my managers, Mark Embree and Curt Johnson, had faith in me and long term vision.

I would like to finally acknowledge Maj Scott Graham and Dr. Kenneth Hopkinson for their constant availability, insights, and patience.

M. Brent Reynolds

Table of Contents

List of Figures

List of Tables

Mitigating TCP degradation over intermittent link failures using intermediate buffers

I. Introduction

This chapter introduces the work and places it in the context of a larger military construct referred to as Network Centric Operations. In Network Centric Operations, data must be relayed throughout numerous mobile communication nodes. By networking the various nodes together exchanges significantly more information, in real-time and near real-time than is currently possible. The focus of this thesis is improvement of reliable transmissions over a channel that is intermittently available.

Network Centric Operations is a vision of people, organizations, and tools working together where quality communication is demanded by the users and provided by the network systems. Users demand rapid and reliable communication because distributed information is needed to make real time decisions. By providing reliable and rapid communication, the system facilitates an increase in the rate of decision making, thereby creating or maintaining an advantage over competitors. In these operations, information flows in all organizational directions: up, down, across, and broadcast. Decision loops exist where information is repeatedly gathered, processed, and disseminated. More and more decisions are made in less and less time. Operations of all kinds are increasingly becoming network centric in order to be more efficient and more effective. Government, military, private organizations, and even individuals, are actively and passively becoming more network centric. To improve performance, we need more information, causing an increase in information demand. The response to the demand is to have more information. This ironically increases awareness of how additional information can assist us, causing yet another increase in demand for information. As people, businesses, and all levels of organization become more reliant on the flow of information, their operations begin to center around the network; hence we refer to this as network centric operations.

To be successful in this exchange, technologies must work together. Machines must be capable of communicating with each other over various protocols. Working together, communication hardware, software, protocols, and users interact to deliver information across space and time. Although directly connected wire networks offer the greatest reliability and bandwidth, the desire for mobility has produced a growing variety of physical media such as wireless LANs, cellular networks, satellite, and even free-space optical links. A single data packet may traverse all of these forms of media, as it runs from a source to a destination. Additionally, the data may be translated from one format to another as it traverses the network. Understanding the interactions between the communication protocols, the link characteristics, and attributes of these various technologies is essential to achieving reliable performance across a range of operating environments and deployment scenarios.

In addition to the requisite ability to interoperate, military tools must work whenever and wherever we need them. These tools are used in demanding locations and environments for very good reasons. These difficult places and conditions are found in post-disaster rescue and recovery operations, on the ground in a battlefield, or in the harsh climate of deep seas or outer space. Information needs to flow into, out of, and within the harsh area. To assure connectivity, systems must employ various aspects of diversity including spatial diversity, frequency diversity, temporal diversity, and equipment diversity.

The term Hybrid Communications refers to the utilization of diverse forms of communication. Multiple forms of communication may be used to meet the *requirements and goals* of the communication and to overcome any environmental *conditions*. Although a diversity of media may be able to meet requirements better than homogeneous media, the existence of diversity necessitates decision making. Decision making requires suitable algorithms and processing power; i.e. with only one form of communication media, there is no decision regarding the choice of media.

The *requirements and goals* of the communication may limit choices in the media, hardware, software, and protocols. For example the requirement that the message not be compromised requires encryption software. As more demands are placed onto the communication the fewer options are available. Similarly the conditions of the network and the node constrain options.

Further consideration is given to the network at the macro and micro level. Hybrid Communications enhances Network Centric Operations by incorporating requirements and conditions at both *global* and *local* levels. *Global requirements* include but are not limited to such things as network connectivity, stability, security, priority, cost, and utilization. *Global conditions* include traffic demand, backbone topology and link state, policy changes, or other large scale shifts. *Local requirements* involve things like mobility, or lack there of, hardware, power consumption, and man power. *Local conditions* such as weather, presence of adversaries, injuries, and malfunctions impact communication. Table 1.1 shows the matrix.

Investigating and testing the interaction between global and local requirements and conditions involves models, simulations, and testbeds. In the process of this thesis a testbed was implemented which provides a means to integrate various forms of wireless and wired communication. The testbed network provides physical nodes to

Table 1.1: This is the matrix of example requirements and conditions that exist at the global and local scope of the network.

	Local	**Global**
Requirements	mobility, hardware, power consumption, man power	network connectivity, stability, security, priority, cost, utilization
Conditions	weather, presence of adversaries, injuries, malfunctions	traffic demand, backbone topology and link state, policy changes

transmit data across the network given certain requirements. The network additionally provides a mechanism to alter or emulate changes in environmental conditions.

Successful communication is defined as the successful transmission of data from a specific source to a specific destination. The source and the destination could be in direct communication however, the more likely scenario is that one or more intermediate nodes relay transmissions between them (multi-hop). In the latter case, the point to point exchange of data between nodes along the path must succeed at each step for successful end to end communication. In a wireless communication media, the environmental conditions affect how well the point to point transmissions work. Hybrid Communications provides strategic and tactical mechanisms to facilitate overcoming these link level challenges.

For our purposes, the challenged link between nodes can be modeled as a momentary status of up or down, regardless of the cause. We refer to the momentary disruption of communication between two nodes as *link wink*. Link wink may be the result of any of a number of possibilities. An aircraft used as a hub of communication may come in and out of range of ground based nodes. A central node using a directional communication device may service many nodes by physically moving the transceiver to point from one node to another. One node will experience link wink as the transceiver on the central node temporarily services another node. Interference or jamming may result in link wink. When a link is unavailable as a result of congestion and/or starvation, we refer to this as logical link wink. Although not a result of physical factors, the loss of the link due to congestion or starvation is semantically identical to the loss of the link due to interference; with the possible exception that the duration of the wink or its pattern of occurrence may be different. In general, link wink represents the time in which data packets do not successfully move across the link.

The cause of the link wink may be predictable. For example, the mobility patterns of nodes may be known. If the wink is predictable or periodic, we can exploit

the moments in which the link is available. If the wink is unpredictable, we must employ methods which produce good overall performance, without leading to system instability. Depending on these causes, their effects, and the requirements of the communication, a variety of options exist to deal with the situation. Traffic can be routed over other links. A different form of communication can be employed. The communication can be canceled entirely. Information could be sent to the source, deferring the decision to the source. An alternative route can be selected; simply bypassing the troublesome link. Of these many options, this thesis focuses on aggressive retransmission across the troublesome link.

Most reliable end-to-end communication today uses the well known Transport Control Protocol (TCP) over the Internet Protocol (IP). TCP is a reliable, in order delivery protocol. Using the presence or absence of acknowledgements as a feedback mechanism, it is optimized for a wired, well connected environment. Unfortunately, while its reliability mechanisms are desirable, its performance in unreliable environments is well known to be poor [1]. Imagine a large scale, ad hoc, wireless environment, such as a disaster area or a battlefield, employing a variety of wireless protocols, both directional and omni-directional. This dynamic environment does not match the assumptions of TCP. TCP assumes a high level of connectivity in its end to end transmission. Lost packets are assumed to be caused by congestion, to which TCP responds by reducing transmission attempts. While congestion is the most common form of packet loss in a wired environment, it is not in the wireless environment where a large percentage of failures are a result of bit errors over a harsh channel. The use of TCP in the wireless environment represents an incongruent matching of assumptions to conditions.

The way TCP responds to lost packets is called congestion control. The most congestion control assumes that loss is a sign of excessive traffic at a router along the path. To prevent congestion, this approach reacts by sending fewer packets less frequently. Naturally the throughput for the particular stream employing TCP drops. Presumably, other TCP streams are also affected by the packet loss, and hence they

too reduce transmissions, causing appropriate reductions in the congestion at the router. Various other algorithms have been developed to react differently, given different assumptions about packet loss. Some assume a wireless environment such that packet loss is not necessarily congestion. Others assume a more dedicated, reserved path such that congestion is unlikely. In all cases the congestion control must provide for back off in the face of congestion. This fairness keeps the network available. This fairness requirement causes TCP to under perform when packets are lost and congestion is not present.

This thesis shows that if the loss of packets caused by link wink is addressed near to the point of trouble, and not at the transmission source, overall performance does not degrade. If along the path, routers are equipped with additional processing and storage resources, the point of trouble is addressed locally. The additional resources allow for the passive buffering of the TCP flows. Once trouble is detected, i.e. the local node detects the lack of acknowledgements, the buffer is immediately retransmitted (as opposed to waiting for the source to detect the trouble and begin retransmissions). Throughput is near optimal (fully utilizing the link when it is available) because the intermediate buffer transmits as soon as the link is available. Traditional TCP is necessarily delayed as the source is expecting acknowledgments later, and the retransmissions require time to arrive at the intermediate buffer, at which point the link may have winked out again. TCPs mechanisms are forced to hope that the retransmission coincides with the links availability.

Further optimization is accomplished by notifying the source not to retransmit any of the packets that have been buffered, thereby freeing the upstream links to transmit other data. Three related factors contribute. First, packets that have traveled all the way to the buffering router are not needlessly consuming network resources by being retransmitted. Second, the source has the opportunity to send fresh packets. Third, congestion avoidance is not invoked at the source.

In summary, with an appropriate matching of assumptions, TCP can be made to perform much better in the environment of wireless network centric operations. We believe that strategic buffering, as we have defined it, holds great promise to serve as a mediating interface between incongruent assumptions and operating conditions. The remainder of this thesis flows as follows. Chapter Two outlines related work and background material necessary to properly place this work in context. Chapter Three presents our approach to implementing strategic buffering on challenged links and a mathematical model. Chapter Four presents analysis of our expected results compared with the actual results obtained on a physical testbed. Chapter Five concludes the document with a summary of results and recommendations for further research and development.

II. Background

2.1 Overview

The required knowledge to appreciate this thesis is an understanding of networks and their protocols, the Transmission Control Protocol (TCP), TCP's congestion control mechanisms, and networking testbeds. Additionally related items include delay tolerant networks and ad hoc wireless networks. This chapter first covers TCP and two specific papers most closely related to this thesis. This is followed by coverage of testbeds, network storage, TCP related issues, and finally delay tolerance.

This thesis focuses on data flow retransmission strategies in a challenged environment. In general the strategy provides a mechanism for a router somewhere in a data flow to retransmit packets in response to an unreliable link in the route, as opposed to the typical TCP approach of relying on the source node to retransmit packets. In order to accomplish this, an intermediate router necessarily "listens"to the traffic corresponding to a given TCP stream and reacts accordingly. The listening router under some conditions assumes the retransmission of the flow and suppresses the retransmission of the source. Before assuming this responsibility, which we refer to as *taking custody*, the router has been copying the data packets to a buffer in anticipation of possible retransmission. As acknowledgments return for these data packets, the opportunistically copied packets are removed from the buffer. The buffer approximates the packets in flight. This buffer is retransmitted repeatedly in bulk to overcome the troublesome link. With this approach in mind, we now address some fundamental principles and protocols necessary to appreciate its application.

2.2 Transmission Control Protocol (TCP)

TCP is the established mechanism for ensuring reliable, in order delivery of a stream of data packets. In general, data packets are sent from a particular source to a particular destination. The destination responds with an acknowledgment packet for each data packet is successfully receives. As acknowledgments are received by the source more data is sent. TCP attempts to achieve fairness by reacting to down-

stream congestion in two ways: sending less data and sending it less frequently. A TCP source determines the existence of congestion implicitly through the absence of acknowledgment packets. Essential to this thesis is TCP's inability to distinguish between packet loss due to congestion and packet loss due to other reasons such as a lossy link. While all of the intricacies of TCP are not discussed here, extensive detail is found in *TCP/IP Illustrated* by Wright and Stevens [30], and many other books, conference proceedings, and archival publications.

2.2.1 Congestion and Backoff. TCP uses acknowledgments, retransmission, window size, and timers, to provide reliable, in order transmission with some fairness (flow control). Because of its goals, assumptions, and trade-offs TCP does not perform well in a topologically challenged environment. With no ability to distinguish between congestion and other losses, TCP assumes unacknowledged packets are dropped due to congestion somewhere between the source and the destination. As TCP is presumably operating on many streams, delivering data from many different source-destination pairs, the collective result is that congestion will be relieved if all TCP flows coming into the congestion area reduce their respective load. The source reacts by incrementally shrinking the transmission window and incrementally waiting longer between retransmissions. In contrast, if packets are lost due to a link winking in and out, the rate of transmission should not slow down because the winking is neither caused by nor cured with a slower data rate. Overall throughput suffers significantly by reacting to lost packets in this fashion.

2.2.2 Window Size. Fundamental to TCP's performance is window size. The window size is the number of packets that TCP will send before waiting for an acknowledgment (ACK). TCP endeavors to have the optimal number of packets in flight such that the pipe is full of data packets in one direction and full of acknowledgments in the other. As an acknowledgment is received by the source the transmission window is increased by one (until some upper limit is reached). If an acknowledgment is not received within a calculated timeout based on estimated round

trip time, the size of the transmission window is cut in half. This is referred to as additive increase, multiplicative decrease [30] and leads to a saw tooth effect [30] as the window slowly increases to a point only to immediately and dramatically drop. In a challenged environment, the window size never achieves a suitable size and thus limits performance.

2.2.3 Retransmission. Retransmission is indisputably necessary for reliable transmission. TCP will therefore retransmit packets for which it does not receive explicit acknowledgment of reception (ACK) from the destination, within a suitable time period (referred to as back off, which is itself an adaptable parameter). Presumably, as packets are retransmitted, they will eventually reach the destination and be acknowledged. If retransmissions continue to go unacknowledged, however, the time between retransmissions attempts (back off) climbs; doubling every timeout up to a maximum of 64 seconds. Upon retransmission, TCP retransmits the entire window of unacknowledged packets. TCP also contains a gross timeout of just over nine minutes in which retransmission attempts cease entirely; although this is generally operating system and implementation specific. In a challenged environment, retransmission is not coordinated with the links being up. Because TCP is an end-to-end strategy, TCP has no information on the status of the links between the source and the destination. TCP transmits on its clock, not when the links are up. In a challenged environment performance suffers dramatically.

2.2.4 Fast Retransmit. In a reliable link, packets are occasionally corrupted, but this typically occurs in an individual packet as opposed to affecting the entire stream. In an attempt to avoid disruption due to a single packet loss, TCP includes a mechanism to detect the loss of a single or very few packets. Sequence numbers in TCP are incremented by the number of bytes sent rather than by the number of packets sent. Under normal circumstances, the receiver acknowledges the reception of a packet by sending an ACK with an acknowledgment number equal to the sent packet's sequence number plus its payload's length in bytes. This represents the next

sequence number the destination expects from the source. For example, packet 14 shows up and has 45 bytes in it. This means the next sequence number is 59. The ACK field in a return packet from the destination contains the number 59, indicating it got packet number 14 (and its 45 bytes), and *expects packet 59* to be the next one.

Fast Retransmit begins when the destination receives a packet out of order, e.g., the next expected packet did not show up but a subsequent packet did. For every packet received after the missing packet, an ACK is sent indicating the missing packet as the *next expected packet*. In the example, if packet 59 did not show up but several packets after it did, the receiver will send an ACK for 59 (the ACK for packet 14). This is interpreted by the sender to mean that packets are still getting through, but a particular packet (number 59) is missing. The sender, upon getting three of these duplicate acknowledgments, instantly retransmits the missing packet without decreasing the window size. This avoids paying the timeout penalty of waiting for 59 to become unacknowledged. The penalty instead is that the complexity of TCP has increased.

2.2.5 Send Window. So far the discussion has been about TCP's handling of flow control. In fact there are two transmission windows in TCP. The one discussed previously is technically the *congestion* window. In order to prevent the overflow of a receiver's buffer, the receiver specifies a requested window size in the packets it transmits back to the sender. This is the *send* window. This window tells the sender just how much the receiver can actually handle. This is limited to the buffer size the receiver wants to implement.

To prevent wasteful transmission, the sender will not have more than this window size in unacknowledged packets out at a given time. The receiver might need to keep this window small if it is constrained by resource limitations or stresses. This window is the largest amount of data the receiver will hold onto while waiting for a continuous set of bytes to deliver up the stack to the application.

TCP guarantees in order delivery. This means if a previous packet is missing the data it has received will not be passed up to the application out of order. The send window is generally fixed during the transmission. The maximum number of packets in flight is the minimum of the transmitter's congestion window and the transmitter's current estimate of the receivers send window.

2.2.6 Congestion Control Algorithms. The classic TCP congestion control algorithm was described above. Over the last several years, multiple variations of the TCP congestion control have been studied, and even adapted. Each of these varying flavors of congestion control serves a specific purpose. For example, since the maturation of TCP, long distance, high speed transmission lines such as 1 Gb and 10 Gb have become common place; creating a huge delay bandwidth product which results in TCP becoming a bottleneck to improved performance over such links. Additionally wireless communication has become ubiquitous. In both environments, TCP's classic congestion control algorithms under perform.

To assist in the development of other TCP congestion control algorithms, Linux has included a simplified mechanism for creating and deploying new congestion control algorithms. This simplified mechanism in Linux is reviewed in McDonald and Nelson [23]. These changes are recent additions to Linux and significant documentation does not exist. In essence, at critical moments in handling TCP retransmissions, Linux provides callback function hooks which allow a researcher control of the various aspects of TCP congestion control: send window size, congestion window size, slow start threshold, etc. It is contemplated that the various strategies will affect the performance of TCP in a challenged environment.

The Linux kernel v2.6.14.7 includes several congestion control algorithms. They are listed below. These descriptions come directly from McDonald and Nelson [23].

Reno is the implementation of Van Jacobsons research [11] and was the default congestion control scheme until recently.

Binary Increase Congestion Control (BIC) [31] aims to address issues on high performance networks, particularly around RTT unfairness, and uses a combination of additive increase and binary search to alter the size of the congestion window.

Vegas [4] is based on Reno and tries to track the sending rate through looking at variances to the RTT along with other enhancements.

Westwood [22] is an implementation that estimates the available bandwidth and is claimed to be suited to wireless use or other networks where loss may occur which does not mean congestion.

TCP-Hybla [5] is a congestion control mechanism that works with links such as satellite which have high RTT but also high bandwidth as some other congestion control mechanisms favour low RTT flows.

H-TCP [21], **Highspeed TCP** [9], and **Scalable TCP** [15] all aim to improve congestion control on high speed networks.

2.2.7 Challenges. Al Hanbali et. al. [10] is a survey of the issues, performance, and solutions in TCP for mobile ad hoc networks. Five categories of issues are described: high bit error rates, path asymmetry, network partitions, route failures, and power constraints. The *bit errors* come primarily from signal attenuation, Doppler shifts of mobile units, multi-path fading, and signal interference. *Path asymmetry* comes from bandwidth asymmetry, loss rate asymmetry, and route asymmetry. *Network partitioning* is primarily a problem caused by mobility and/or power control problems. *Routing failures* are again caused by node mobility; i.e., even though the network is connected the route may fail due to the changing topology. From the perspective of performance, simulations demonstrate that as the number of nodes increase, and thus the number of hops increase, the performance of TCP decreases rapidly.

In Al Hanbali et. al. [10], the proposed solutions to TCP's difficulties are categorized into one of two types: Cross Layer and Layered. The Cross Layer proposals are further broken into three categories: TCP and network cross layer, TCP and physical cross layer, and network and physical cross layer. In the TCP and network cross layer, the solutions generally propose suspending the TCP session if there is a route failure. Each solution uses additional control packets to distinguish between congestion failure and route failure. TCP-BuS suggests buffering the packets in the intermediate nodes while the route is failed. Lastly, Split TCP [18] uses proxies along the route to provide point to point ACKs in addition to end to end acknowledgments. The network and physical cross layer focuses on routing and signal strength [10]. The Preemptive routing in ad hoc networks [10] attempts to predict that a route will fail by looking at the signal power of the packet receipt notification. If the value is below a threshold and new route will be discovered and hopefully used before the original route fails. Another solution again predicts trouble by checking the power of the next hop's signal, but in this solution a notification is sent to the sender so that transmission and stop and a new route discovered (Proactive Link Management [10]). Additionally though, the "ailing" node increases its transmission power (Reactive Link Management).

In Al Hanbali et. al. [10], the Layered proposals are solutions limited to a single layer of the stack, either the TCP layer or the link layer. In the TCP layer, one technique suggests not using the exponential back off for route failures. Another assumes that out of order events demonstrate a route failure and thus congestion control can be disabled for a specific time period. Another demonstrates that setting the TCP window size to four packets and using a delayed ACK and cumulatively ACKing every other packet increases performance 15%-32% [10]. This reduces the number of ACKs in flight and makes TCP increase the window size more slowly [10]. Finally, another uses a dynamic delayed ACK that increases as the transmission continues (successfully) over time [10]. The link layer approaches primarily rely on

using some form of Random Early Discard at the link layer to reduce contention or increase fairness.

Since Hybrid Communication Testbed is wireless, it faces the same issues facing TCP in an ad hoc wireless network. Some of these issues can be alleviated using directional communication. However, TCP still presents a significant challenge. Although it contemplates a Cross Layer solution by suggesting information from the other layers could be used to make good buffering decisions, this thesis is specifically a Layered solution.

2.3 TCP Bulk Repeat

The bulk retransmission of a TCP buffer is introduced in TCP Bulk Repeat [32]. This work examines the performance of retransmitting an entire buffer in response to packet loss. Yang et. al. [32] assumes a difficult environment where the probability of packet loss is as high as 20-30%. Three observations in TCP are discussed. First, multiple losses occur in a transmission window. Second, when the error rate is high there are non-optimal back offs in the Retransmission Timeout. And third, in the high error situation the slow start threshold and the congestion window are much smaller than the optimal value. This retransmission occurs in response to a lost packet indicated by repeated acknowledgments. Instead of simply sending the single lost packet, the entire window of packets, up to some size, is resent.

This mechanism is similar to this thesis in that it retransmits a buffer of packets in response to loss. As discussed later in Chapter Three, this thesis faces similar issues as pointed out in TCP Bulk Repeat and include determining the amount of the buffer to retransmit and the amount of time to wait between retransmissions. Too much data, too quickly has the potential to overwhelm the channel. Too little may not overcome the channel's difficulties.

The TCP Bulk Repeat differs from this thesis in two related ways. Foremost their implementation exists in the source of the transmitted data. This thesis places

the retransmission in an intermediate router much closer to the destination and to the difficult link. Since the logic of the retransmission is in the source, TCP Bulk Repeat uses information in the source such as windows sizes, acknowledgment receipt, and other internal TCP state information. Because this thesis provides retransmission downstream, the TCP state information in the source is not fully available and is only approximated in the router.

2.4 Snoop TCP

Snoop TCP [2] introduces the snoop agent. This agent resides in a base station (access point) for a wireless network. The agent monitors TCP packets as they pass through the station in both directions. From this monitoring a cache of unacknowledged packets are stored in the base station. Repeated, duplicate acknowledgments indicate packet loss. The snoop agent transmits the missing packet from its cache, if available, and suppresses the duplicate acknowledgments from reaching the source. This prevents the source from invoking congestion control and fast retransmission.

This solution is similar to this thesis in that it has an agent located in an intermediate node that retransmits packets. It is further similar by keeping per-connection state and only keeping unacknowledged packets. It differs in that it only reacts by sending the indicated missing packets. This thesis sends all of the unacknowledged packets when it is time to retransmit.

2.5 Hybrid Communications Laboratory

This thesis is part of a larger effort called Hybrid Communications, which addresses the use of multiple communication media interacting in harmony for an entire network. To study these issues, we are developing a hybrid communications testbed capability, which will allow various forms of wireless, free space optical, and wired communications interfaces to interact. While this thesis' effort focused on the effects of link wink, which were simulated in a wired medium, much work was accomplished

for the purpose of enabling future research in hybrid communications. That research is out of scope in this thesis but is mentioned for completion.

2.6 Testbeds

This thesis is the first research project utilizing the Hybrid Communication Testbed. During the course of this thesis the testbed was initially researched, prototyped, and developed. This section discusses the several wireless testbeds previously documented. Their purposes and implementations are discussed by comparing and contrasting them to the Hybrid Communication Testbed.

Wireless testbeds exist in a variety of settings. These testbeds test a variety of topics including protocol development, power reduction, transmission interference, and effects of mobility. The primary goal of wireless testbeds is to verify the findings suggested by theory and demonstrated in simulations. Testbeds validate (or invalidate) theory and simulation by injecting real asynchronicity, real interference, real equipment, and real spatial restrictions. A few of these testbeds are covered here and are compared to the Hybrid Communication testbed.

At the CSRC at NIST (the Computer Security Resource Center at National Institute of Standards and Technology), a wireless testbed has been created for their MANET and Sensor Network Security Project. Their objectives are to test node mobility, facilitate configuration management, and to recreate an environment [14]. Their software tools are written in C and Linux shell scripts. Logical networks are created and manipulated by configuring the nodes' Linux IP tables. Additionally, mobility is emulated by varying power levels with a software tool. Each node (Linux Intrinsync CerfCubes) communicates with other nodes using 802.11 wireless connections. The testbed is administered using a central server that communicates with each node via a wired Ethernet connection.

The Hybrid Communication Testbed similarly utilizes a centralized server to administer the network and its topology of wireless nodes. The Hybrid Communi-

cations Testbed however utilizes more wireless platforms that 802.11. The Hybrid Communication Testbed is not currently concerned about node mobility, although it is a future consideration.

In Nordstrom et. al. [24] the APE (Ad hoc Protocol Evaluation) testbed presents a methodology for an analyzing the stochastic issues innate to wireless networks. Their goals are to reduce the number of stochastic factors and the variance for these factors. This testbed utilizes Linux laptops using 802.11. Software modules include clock synchronization, a routing daemon, a traffic generator, a scenario interpreter, and a traffic recorder. Data gathered at nodes focuses on connections to other nodes and the strength of each connection over time. Two metrics, Link Change Metric and Virtual Mobility Metric, provide a framework for assessing stochastic variance. Link Change counts connection establishment and loss between two nodes over time. The Virtual Mobility tracks the signal strength of between two nodes overtime. These metrics provide a basis to assert that two experimental trials had statistically similar topologies over the course of the experiment. As nodes move throughout the testbed, the strength and number of links change over time.

The Hybrid Communication Testbed again utilizes a variety of wireless communication platforms, not just 802.11. Similarly though, as the Hybrid Communication Testbed is advanced it will intentionally vary topology. The changing topology derives from intentionally changing a directional link or using a different communication medium. Similar metrics, like Link Change and Virtual Mobility, and others are input into route and topology calculation of the central server. This functionality will exist in the future and is not used in this thesis; the links remain static.

Multi-radio nodes provide some unique issues in testbed and real world environments. In Robinson et. al. [26], multi-hop, ad hoc wireless networks are investigated specifically using nodes with multiple 802.11 radios such that one receives and one transmits. Each node is a Linux PC workstation using PCI bus NICs. Instead of using a backbone of wireless access points wired together, the purpose of the testbed

is to emulate a multi-hop wireless backbone. In Robinson et. al. [26], the investigation demonstrates simultaneous activation of multiple radios in the same node leads to degradation due to crosstalk, leakage, and inadequate antenna separation. The experimentation shows that separating the antennas greatly increases performance. Specifically, their routers can not have more than two radios and require a minimum antenna separation of 35db.

The Hybrid Communication Testbed provides a wireless backbone with specific focus on directional heterogeneous platforms. Robinson et. al [26] shows the difficulty of only using 802.11 omni-directional technology in a backbone architecture.

TAPs, Transit Access Points, is described in [13] and has a home at Rice University. TAPs provide a multi-hop wireless backbone for broadband networking. A TAP is stationary 802.11 access point. Each TAP communicates with other TAPs with directional 802.11, forming the backbone. A limited number of these TAPs provide gateways to the outside, wired world. In the paper, ten premises of the TAPs architecture are presented focusing on cost-efficiency, performance, scalability, fairness, and the need for new routing and scheduling protocols.

The Hybrid Communication Testbed is very similar to the TAPs architecture. The Hybrid Communication Testbed can have multiple links between each node in the backbone (potentially of different wireless platforms). These multiple links can be used for different types of transmissions or for redundancy or for bandwidth aggregation.

At Microsoft Research, the Mesh Connectivity Layer (MCL) [7] has been created to exist between layer 2 and 3 of the OSI model. This software creates a virtual network card that sits on top of other network cards in a single PC. The mesh network is an overlay network lying on top of the Ethernet data link layer. Each node is assigned a virtual MAC address. By default Link Quality Source Routing (LQSR), derived from Dynamic Source Routing (DSR), routes packets in the mesh network. In [7], particular interest is paid to routing in a mesh network of multi-hop, multi-radio

nodes. LQSR relies on a metric to establish the "quality" of the links. Two particular metrics are tested: WCETT (Weighted Cumulative Expected Transmission Time) and ETX (Expected Transmissions). In short the results show that WCETT out performs ETX which outperforms a shortest path routing algorithm. Each of twenty three nodes are PC workstations running the Microsoft Windows XP operating system and two 802.11 wireless NICs. Interestingly, they point out that the Windows OS will not support two identical wireless NICs at the same time so each is a different make. They point out that TCP degrades in performance in long multi-hop paths due to high round trip times and the high probability of packet loss. Additionally, they point out that the multiple radios interfere with each other even when using distant channels and even using 802.11a, 802.11b, and 802.11g in various combinations.

The Hybrid Communications Testbed has two commonalities with the MCL testbed. This virtual mesh network has each node uniquely identified in an overlay network and each node has more than one NIC. The two testbeds differ on two significant issues: routing and topology. The Hybrid Communication Testbed is not using a distributed routing algorithm at this time. Currently a centralized, and more proactive, means of routing is used. Furthermore, determining the quality of a route will incorporate more information than a single metric. From the perspective of topology, the MCL testbed assumes that the topology of the 23 node network is relatively fixed, while the Hybrid Communication Testbed assumes a dynamic topology.

2.7 Network Data Storage

In a topologically dynamic environment, data in transit encounters, with a probability greater than zero, a break in the topology between the source and the destination. In the face of this difficulty the node currently holding the data can (1) panic, drop the data, and potentially alert the failure back to the sender, (2) desperately forward the data to another node with the hope that the new node will have a better chance delivery, or (3) store the data in a buffer until the topology is such that it can be forwarded and potentially alert the sender. The last option requires sufficient and

persistent storage in the network infrastructure. Furthermore, this option must consider the handling of end to end transmission timeouts. A review of some persistent network storage follows.

In Beck et. al. [3], Logistical Networking is outlined as a means of storing data throughout the network. End to end considerations are applied to data storage. Traditional storage, such as a hard disk in a server is called local storage. Local storage is tightly coupled to the process that is accessing it. It has predictable delay, high accuracy, and high availability. In contrast, network storage must provide availability, integrity, confidentiality, unbounded file size, and unbounded duration of storage. These considerations must be addressed with end to end principals. A storage stack, similar to the network stack, is discussed that ranges from the physical devices through a few logical abstractions up to the application. Just above the physical storage and local access layers, the Internet Backplane Protocol is defined to allocate and manage the storage on network storage depots. Above this layer is the exNode, a layer above the IBP, and is analogous to the iNode structure in the UNIX file system. It points to depots on the network instead of locations on a disk.

The above approach focuses on formal storage systems, such as a distributed file system. It acknowledges the important requirements of storing data on a network that must be considered if the storage is a file system or the storage is data in transit. The concept of attaching storage to network infrastructure, such as routers, is essentially similar. The storage buffers in the Hybrid Communication testbed are locally accessed by the node in need of buffering.

A mobile file system named Coda, initially described in Satyanarayanan et. al. [27] and highlighted in Satyanarayanan [28], provides a means for mobile workstations to disconnect from a network while continuing to work with the local file system. Coda caches files on the workstation as they are accessed or predicted to be accessed. Once the workstation disconnects the file operations work on the cached objects. Finally when the workstation reconnects the cached files are reintegrated with the

corresponding files on the network, and vise versa. Also highlighted in [28], Odyssey offers a Resource Negotiation API for applications. Applications request an amount of a resource and a window of tolerance. The API returns the amount available and further notifies the application of changes in the availability. Odyssey optionally delivers the resources in different levels of fidelity, e.g., degrees to which the copy of the data presented matches the copy on the server. Coda is an application transparent solution to the problem of disconnectedness. Odyssey is an application-aware solution to the issue of disconnectedness or limited connectivity.

Coda is similar to our strategic buffering in that Coda assumes that connectivity is not continuous. Similarly Coda will buffer changes over time until the channel is available. The changes do not have to be forwarded all at once. Large files can be updated by transmitting only the changes to the document, similar to database transactions; see Lee [20]. Coda operates at, or just below, the application layer of the OSI model defining connectivity in the end to end context, while our strategic buffering operates at the transport and network layers defining connectivity one link at a time. Coda is concerned with consistency. Odyssey and strategic buffering share the need to define from the application layer what the tolerance is for resource constraints. Strategic buffering defines how much data must be transferred in what time frame. Furthermore, strategic buffering accelerates the transmission as resources become available.

Kangaroo [29] off loads the transmission and retrieval of large data files in a super computing environment. It off loads the work from a super computer, much in the same way an operating system and a hard drive controller off load the work from the CPU. Kangaroo offers this to every node in the Kangaroo network, so data can be hopped from one node to the next until it reaches its final destination. It uses a fixed routing table in each node to accomplish routing. In Rajamani [25], a multi-route Kangaroo is contemplated and evaluated so that Kangaroo works around failure points. Strategic buffering behaves similar to Kangaroo in that data is fed

from one node to another until the destination is reached. Strategic buffering does not do routing itself. It uses a provided, static routing table in this thesis.

2.8 Big Picture

Kramer et. al. [19] is a broad review of technologies needing improvement to facilitate the future of high powered computing, called Deep Computing. The environment discussed includes large scale grid computing, supercomputers, and large sized data sets. Included in the discussion, of interest to this thesis, are the needs of routers and of transport protocols. For routers, the mismatch of Maximum Transmission Units (MTUs) across networks causes inefficiency. One solution suggests Layer 7 routers that repackage MTUs; however, production of these is noted to be unlikely. The other suggested solution is to turn the computing nodes into high speed routers, so they can be placed directly onto the highest speed connection that has the largest MTU. In regards to transport protocols, current TCP is inadequate. TCP aims to fully utilize the network path and be fair to other traffic. TCP, by its design, is prevented from utilizing the available high bandwidth delay product. In the example given, one Gb path with a 100 ms round trip and an MTU of 1500 bytes needs the error rate to be less than $2 \cdot 10^{-8}$ (one packet every 555 round trips) in order to fully utilize the allotted capacity. Without any congestion, this is less than the random error rates of the equipment used to support the network path. Two solutions to TCP's issues are discussed. Using multiple streams increases throughput but, as noted, is unfair and requires software to disassemble and reassemble the streams. Additionally, reserved bandwidth, or a virtual circuit, limits congestion and guarantees bandwidth for the TCP stream. However, the paper points out that this does not remove the minimum error rate described above for high bandwidth reservations.

While the Hybrid Communication Testbed is not specifically concerned with massive parallel computing and its capacity concerns, it faces similar troubles on a smaller scale; routers and the transmission control protocol must change. Using a variety of physical media to communicate presents the issue of varying MTU sizes.

Smaller MTUs require more transmissions raising the possibility of message transmission failures. Furthermore, as previously discussed, TCP works well in the environment it currently excels in: continuously connected, low bit error rate, and medium to small bandwidths; in other words: wired general purpose internets.

2.9 Delay Tolerant Networking

Fall [8] presents an architecture addressing the difficulties of data transmission in a challenged network. A challenged network is defined as a network with worse latency, bandwidth limitations, error probability, node longevity, or path stability than typical networks, such as wire networks or the Internet. This work is a broad survey of the aspects that must be considered in such networks. One of the points is the issues facing TCP in such an environment. TCP's original specification defines the maximum segment length to be two minutes. For a network that could be disconnected for any length of time, this causes obvious difficulties. At the IP layer, there is no mechanism for fragment retransmission. IP also has a time to live field (maximum 255) that indicates the number of seconds (or hops) a particular packet can live. In regards to routing, Fall [8] points out that conventional routing protocols (RIP, BGP, etc) determine paths based on available connectivity. In the face of regular disconnection these protocols will not function sufficiently. In terms of application development, applications assume a connected, low latency environment. Four issues discussed are short application timeouts, lack of automatic failover, application execution is assumed to be much longer than transaction duration, and application protocols are "chatty". Several other points are made about SMTP, the Postal Service, naming conventions, and internetworking unchallenged networks with challenged networks.

A more detailed discussion about routing in a delay tolerant network (DTN) in Jain et. al. [12] addresses specifics of path discovery and maintenance with varying levels of network knowledge. In a DTN an end to end path may never exist. The delivery eventually occurs over time. If a DTN has paths that vary predictably over time, a proactive approach to routing might be useful and would likely involve a set of

fixed routes that are indexed by time. Otherwise a reactive scheme is more attractive. To this end, the concept of (network) Knowledge Oracles is introduced. These are nodes throughout the network that collect information about the past, present, and future state of the network. The Contact Oracle answers questions about contacts (connections) between any two nodes at any point in time. The Contacts Summary Oracle answer questions about the aggregated statistics of contacts. The Queuing Oracle gives information of the instantaneous buffer occupancies at any node at any time. The Traffic Demand Oracle answers questions about requests for service in the networks. Discussion is given about routing with none, some, or all of these oracles and performance of various algorithms are analyzed.

The Hybrid Communication Testbed's initial vision utilizes a centralized routing and topology server that is similar to the oracles discussed above. This is not implemented in this thesis. Current work in the Hybrid Communications Laboratory is addressing the ability to predict various network state information, thereby enabling the oracles presented above. Additionally, the testbed incorporates directional communication. The direction may change due to a prescribed topological shift. If packets were traveling along the path when the shift occurred, the packets are buffered at the node until the topology shifts backs upon which they are transmitted. The DTN solution relies on a bundle transport protocol that resides at the application layer. This Hybrid Communication Testbed operates at lower layers in the stack.

2.10 Summary

This chapter provides background specific to the testbed implementation and to the experiment at hand. Additional related topics and similar research has been addressed. The next chapter builds upon this knowledge by detailing the specifics of strategic buffering and a model of its performance.

III. Modeling and Implementing Strategic Buffering in the Hybrid Communication Laboratory

3.1 Overview

This thesis investigates the effects of TCP retransmissions in a challenged environment and strategies to overcome them. This chapter discusses the environment, assumptions, mathematical model, testbed implementation, and methodology used to analyze TCP performance with and without strategic buffering. The environment is a wireless networking testbed, intended for this work and flexible enough for future work. The testbed consists of multiple nodes connected using specifically configured IP networks. In the testbed, links between nodes can be disabled to simulate transmission difficulty. Intermediate nodes strategically buffer and retransmit packets to overcome the transmission difficulty. In order for this buffering and retransmission strategy to work with TCP, modifications are made to the TCP implementations running in the testbed. The testing investigates the effects of buffering versus not buffering under varying conditions of transmission difficulty: the probability of trouble and the length of trouble.

The last part of the chapter describes the implementation of this strategy in a testbed environment. The configuration of the various hardware and software tools is presented.

3.2 Concept Definitions

This section defines the various concepts involved in strategic buffering.

3.2.1 Link Wink. Link wink is a term used to describe the dynamic nature of a link's status. Assuming the link can be available, link wink is the availability of the link. A link connected and available all of the time is not experiencing link wink. A link that is never available is not experiencing link wink. A link's availability that is intermittent is experiencing link wink. For analysis in this work, link wink is defined as a probability of an outage and a duration of that outage. An example

of this is a directional laser link that slews periodically between networks in a time sharing fashion spending one minute dedicated to each. A plane circling a battle field or a disaster area comes in range with some predictability. Lossy RF links create link wink on smaller time scales; potentially down to the packet and bit level. Congestion at a router is a simple example of link wink; at times the link appears to be down. Similar to congestion, higher priority traffic of sufficient load creates link wink for a lower priority flow.

Link wink is modeled in the testbed by dropping packets on the troubled link. If the link is determined to be down (according to a random variable) the packets are dropped. If the link is up the packets are allowed through. For a given interval, i, such as 100ms, the status of the link is determined to be down with a probability of p every 100ms; otherwise it is up with a probability of 1 - p. The status is determined every 100ms, in this case. The link wink space is defined as (p, i). Winks arrive according to a Poisson distribution and have a geometric duration (exponential in the limit). To achieve link wink in the testbed, the winker (code to create link wink) is placed on one side of the link. In one direction it destroys packets before they leave the node. In the other direction the packets are destroyed as they arrive at the node. This allows control of the link wink in both directions to exist in a single node.

3.2.2 Strategic Buffering. To address the difficulties of TCP (discussed in Chapter Two) in an environment where the links are winking in and out, adjustments must be made to the retransmission strategies of TCP. The goals are to increase throughput and decrease retransmissions from the source. To gain this advantage, resources must be available at intermediate routers such as processing capacity and storage capacity. It is desirable to avoid overhead penalties when links are working successfully. The solution described in this thesis uses buffers and retransmissions in routers located in the path of the TCP conversation. Specifically this solution involves buffering strategy, retransmission strategy, TCP state management, intermediate acknowledgment, custody considerations, and leader election considerations.

A TCP flow is identified by a four-tuple: source IP address, source port, destination IP address, and destination port. In order to have the packets available for retransmission at the intermediate routers, packets for each flow (in each direction) are copied to a buffer as they pass through the router, as shown in Figure 3.1. As ACKs return from the destination, copied packets are removed from the buffer. Under normal circumstances, i.e., little or no link wink, packets are simply copied and removed from the buffers with no intermediate intervention.

If it is determined that links are experiencing transmission difficulties, and custody for this flow must be taken, the packets needing to be retransmitted are immediately available from the buffer. Ideally, this retransmission occurs at the router nearest the point of difficulty, thereby avoid wasteful upstream retransmissions. Along the path of a TCP flow each router has the buffer of unacknowledged packets available in the event any particular link goes bad. The buffer approximates the actual window of unacknowledged packets the source has sent. An unsubstantiated, but logical, observation is that on average the buffers in the routers closest to the source will be larger than the buffers in the routers closest to the destination simply because the buffers closer to the destination will see ACKs sooner than buffers closer to the source.

If difficulty is encountered, the router takes custody of retransmissions. Ideally the router immediately before the troubled link takes custody and retransmits, as needed, to optimize the throughput over the troubled link. Subsequently, the source node is notified that custody has been taken, which, at a minimum, tells the

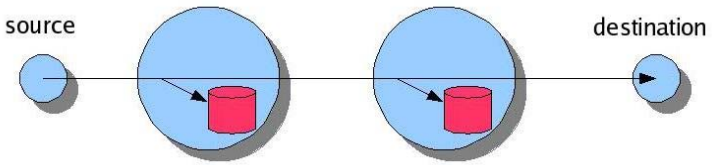

Figure 3.1: Two nodes buffering between the source and the destination.

source not to invoke its congestion control mechanisms, i.e., slow down transmissions. Furthermore, this notification tells upstream buffering routers (between it and the source) that custody has been taken downstream and the packets acknowledged by the downstream intermediate buffer are to be removed from their buffers.

This method combines the facets of TCP Bulk Retransmission and Snoop TCP described in Chapter Two. TCP Bulk Retransmission focuses on retransmitting from the source, not from an intermediate node. Snoop TCP applies retransmission in an intermediate node but only to packets that have been lost to avoid fast retransmission in the source.

3.2.3 Custody. Once packets are buffered and retransmission is to occur, responsibility for the successful retransmission of these packets must be taken by the buffering node. The taking of this responsibility by an intermediate node is referred to as taking custody. Packets that have made it to the node taking custody are not retransmitted by the source (through explicit notification), which saves upstream network resources. Not only is bandwidth made available, but for a troubled network, packets may have already traveled a great distance and encountered significant trouble. There is no reason to send them again. It is left for future consideration on how to deal with the node that has pledged responsibility for the retransmission of these packets, and subsequently disappears.

Ideally the source continues to transmit fresh packets to the buffering node. This allows the buffering node to have a sufficient amount of packets to optimize retransmission over the troublesome link. For example, if a link is up and down for three seconds at a time, performance will be optimized if enough packets are in the buffer to transmit the entire three seconds the link is up.

The blue arrow in Figure 3.2 denotes the notification to the source that custody has been taken and communicates what packets have already been received, e.g. in the buffer. This notification informs the source not to retransmit what is the buffer. It further notifies the other buffering nodes to flush their buffers of the packets for

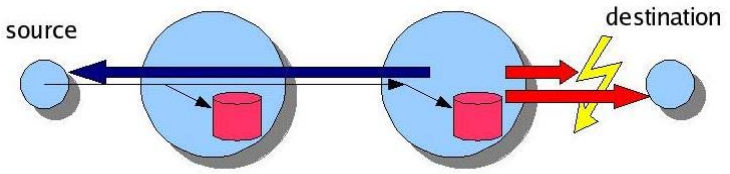

Figure 3.2: Nodes taking custody in the presence of link wink.

which custody has been taken. The red arrows indicate the repeated retransmissions across the winking link.

A significant challenge is determining when (under what conditions) custody is required. The wink of the link is either predictable or unpredictable. The perfect solution to the unpredictable outage would be explicit notification from the link layer that the link is in trouble. If the downtime is predictable, custody is predictable. Knowledge of the anticipated outages must be delivered to buffering router. This explicit notification comes from the global network. For example, a directional link may be temporarily ordered to communicate with another node. During this outage packets could be buffered in preparation of the directional link returning to its normal target. Another example is the predictable movement of nodes. As a node moves back and forth from network to network traffic is buffered in preparation of the node's return.

In the absence of these explicit notifications, either from the global network or from the link layer below, the TCP flow is the only source of information available to the buffering node. The flow of data packets and the return of their acknowledgments is the only indication the router has of trouble. The absence of acknowledgments can indicate a lost destination, a lost route, congestion, a lossy link just outside the router, or a lossy link further down the route, etc. Some conditions require retransmission and others do not. Furthermore, once custody has been taken, conditions for terminating custody must be defined in the absence of explicit notification. The topics of implicit and explicit notification of custody are left for future research.

A subtle note about the assumption of custody is a concern about leader election. If the flow of traffic is the only mechanism determining custody, all of the routers between the source and the trouble will use the same criteria and react in a synchronized fashion with perhaps all assuming custody simultaneously. This anomaly is not handled in this thesis. One consideration to handle this is to use the Time To Live (TTL) field in the packets. This field is decremented for each hop along the path. The value for the TTL in any given packet at any given router can be anything but is monotonically decreasing. It is contemplated that this field should be considered in the timeout value used to determine if custody is taken so that routers further from the source will react sufficiently quicker than routers closer to the source.

The experiments performed in this thesis assume only one winking link and one node taking custody. The first experiment demonstrates the implication of link wink without custodial retransmission. The second experiment analyzes how custodial retransmission performs in the same winking environment. In this second experiment, the node has custody the entire time. The conditions under which to take custody are left for future investigation.

As mentioned in Chapter Two, various congestion control algorithms change how packets are sent in the absence of ACKs. As mentioned above, the buffer must have packets to send when the link is available to perform optimally. When experiencing packet loss, congestion control algorithms are designed to back off. However, in the case of winking links, the buffers need more packets as links degrade. To test this situation, a special purpose congestion control algorithm, called Unfair, is defined. Its modus operandi is to have a fixed congestion window size of 2 GB. The intention is to not slow down in the absence of ACKs. It is noted in McDonald and Nelson [23] that all congestion control algorithms in Linux only manipulate certain variables, not the TCP timers. See Wright and Stevens [30] for more.

3.2.4 Retransmission. As stated previously, a router assuming custody begins retransmitting packets stored in its buffers over the troubled link. Consideration

must be given to two facets of this retransmission strategy. First the size of the router's retransmission window must be determined. Many parameters are available to determine the size the window: size of the buffer, defined or perceived capacity of the link, rate of data arrival, other flow usage of the link, other buffer capacities, flow priority, receiver's suggested window size, link behavior, round trip time, and explicit external notifications. This thesis leaves these calculations to future research and assumes the entire buffer or a set limit (which ever is smaller) is retransmitted.

The second facet of retransmission is the interval. Regular routers send packets out as instantly as possible without regard for the downstream consequences. TCP retransmits with a shrinking window as conditions downstream appear to get worse. The consideration here: should this intermediate, strategic retransmission behave like TCP or like an IP router?

The router may completely consume the outgoing link's bandwidth. If the link is only intermittently available 10% of the time, and the link is not shared, then "shouting" might be the proper choice. The extreme scenario involves a buffer with one data packet and a high capacity link that is winking. This packet could be constantly retransmitted filling the 100Mbit channel with one packet.

The link's medium is an important consideration. Shared links must be taken into consideration. Moreover, bandwidth must be available for returning ACKs to get through in a timely manner. It is further contemplated that if a link is bad enough to warrant custody on one side, it is likely the other side of the bad link has taken custody of flows traveling in the other direction. If both sides engage in shouting on a shared medium, performance may suffer. This thesis leaves this for future research as well. This thesis assumes retransmission of all buffers in custody every 10 milliseconds; transmit, wait, transmit, wait, etc.

3.2.5 Delay Tolerant Traffic. A network generally supports various classes of traffic. The various classes of traffic have varying levels of priority. Traffic such as command and control or real time system control requires the highest priority

generally available. Other traffic such as personal emails or backing up files does not require immediate attention from the network resources. This lower priority of traffic does require reliable, *eventual* delivery. It is thus considered *delay tolerant traffic*.

The overall performance of this low priority traffic is improved by increasing network utilization. Logically, if a network's overall utilization is not entirely consumed, then there is available bandwidth. Unused bandwidth is lost forever. If a continuous supply of low priority traffic is available, the otherwise lost, unused bandwidth provides an opportunity to send traffic. Using previously unused bandwidth increases performance.

A continuous supply of low priority traffic is required in order to take advantage of the bandwidth as it becomes available. In order to accomplish this, the apparent disadvantage of lower priority actually provides the solution. By definition, the lowest priority traffic yields to all other priorities. Then on a unshared medium, by design, the lowest priority traffic is able to be unfair in its transmission scheme. A low priority TCP flow is not required to back off in the face of congestion with higher priority flows. By design the low priority traffic is in a near constant state of congestion. Instead of backing off, the lowest priority traffic should increase its output.

This surplus of low priority traffic must have some place to go. Strategic buffering provides holding tanks for the low priority traffic. The nodes honor the high priority traffic by always sending it when it arrives. However, in the absence of high priority traffic, the reserve of low priority traffic is sent. The low priority traffic is aggressively sent in order to fill the next strategic buffer. It can be sent aggressively because it always yields to the other traffic.

From the perspective of the lowest priority traffic, the unavailability of a link due to channel failure or due to the presence of high priority are semantically equivalent. The presence of high priority traffic can therefore be modeled as if the link were unavailable, e.g. *Link Wink*. Link Wink can be used to model the high priority

network utilization. The generic utilization-delay graph looks like Figure 3.3. As utilization increases, delay increases.

In this instance, if the current high priority utilization is 0.6, the delay is 2.5. If additional traffic is injected to increase the utilization to 0.9, the delay increases to 10. If the strategic buffering used to insert the extra traffic to take advantage of the unused bandwidth, the delay increases slower. This is demonstrated in the two graphs 3.4 and 3.5.

Delay tolerant traffic was one of the original motivating factors to pursue this research thesis. The important thing to take away from this section is the similarity of a lossy link and the intermittent starvation of lower priority traffic. Both are modeled the same way. The mathematical model that follows in the next section provides insight into how TCP behaves in a lossy link and how TCP behaves with respect to a low priority flow in an environment where significant higher priority flows dominate the resources.

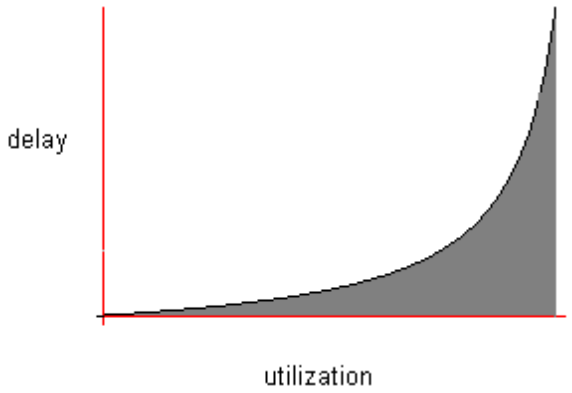

Figure 3.3: This graph demonstrates the exponential increase in delay as utilization approaches capacity.

It is contemplated that hording low priority traffic in a buffer provides a source of packets to send when the link is available. It is further contemplated this buffer of traffic can be transmitted when high priority traffic is not available to the extent that the link's utilization is maximized for optimization. From this is is asserted that (1) the low priority traffic will proceed through the network faster with strategic

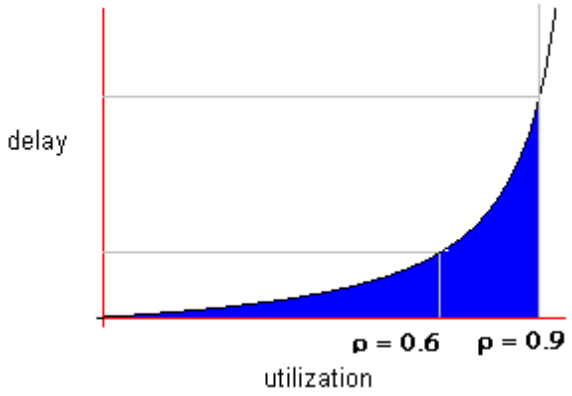

Figure 3.4: This graph demonstrates how far delay increases as utilization increase from 0.6 to 0.9.

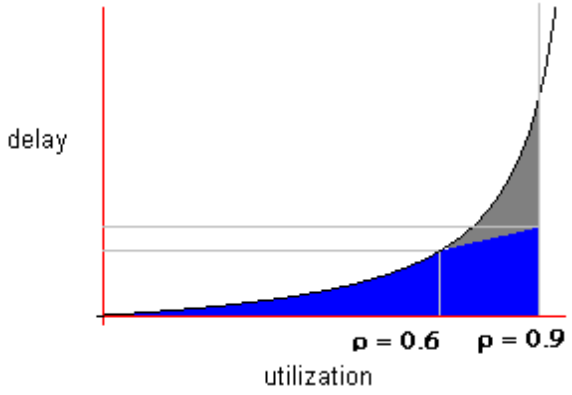

Figure 3.5: This graph demonstrates how the delay of low priority traffic increases less as utilization increase from 0.6 to 0.9 using strategic buffering.

buffering and (2) this improvement *does not interfere* with higher priority traffic. The first point is proven in this thesis. The second is left for future pursuit.

3.3 Mathematical Model

In an attempt to demonstrate the behavior of the transmission times a mathematical model of the retransmission delay is constructed. This model is based on several assumptions. These assumptions are described below and are conservative. The model is based on two distributions. One distribution models the success of an interval on the first transmission. The second models the cost of the successive fail-

ures of retransmitting the intervals that fail on the first transmission. As explained in detail below, the cost of each retransmission is based on TCP exponential back off.

3.3.1 Assumptions. To simplify the model several assumptions are made. The propagation delay is assumed to be zero. The distance covered in the lab is on the scale of feet. At the speed of light this is negligible. The transmission of the ACK for each packet is assumed to be negligible as well. The relative size of the ACK and the fact that ACKs are cumulative make their impact minimal.

The most significant assumption is the following simplification. Each successful interval is assumed to successfully transmit an entire interval of packets. In reality, the congestion control algorithm shrinks the window after a wink is noticed. Once a retransmission starts in a successful interval the congestion window grows quickly as each successive packet is acknowledged. It is assumed that within the interval the congestion window will reach or exceed its average size.

The round trip time in the test bed pinged just shy of 0.5ms. A 100 Mbit speed, in one direction there are just over 4 packets in flight. Assuming the window starts at a size of 1 and every 0.5ms an ACK shows up, the window will increase by 2 packets per ms. In a matter of 3ms, the pipe is full of packets. Assuming the entire interval is full is acceptable because the smallest interval investigated is 40ms. This assumption will underestimate the transmit time because it assumes more packets are sent in a successful interval than are actually possible. Further refinement of this assumption is left for future consideration

For a given interval, i, the number of such intervals required to transmit a file is the transmission time of the file under perfect conditions divided by the interval. For example the 20MB file takes 1.86 seconds to transmit with no winking. If $i = 40$ms, then 46.25 intervals are needed to transmit the entire file. This is rounded up to the next whole interval of 47. At $i = 1100$ms, it takes 2 intervals. Logically under good or bad conditions, the number of successful intervals required remains the same.

As a note, this interval could be reduced down to the size of a packet. It is contemplated as the interval gets smaller and smaller that the congestion windows size and other retransmission strategies will dominate the model more than the TCP exponential back-off. This is left for future research. This model reduces the interval no further than 40ms.

3.3.2 Winking. The winking environment is comprised of two mechanisms: probability and interval. The status of the link is determined periodically based on the probability, p, that the link is unavailable. The period is based in the time interval, i. Every i milliseconds with probability p the link's status is down otherwise is it up. For this test, the probability is such that $p \in \{0.00 \text{ to } 0.40, \text{ step } 0.05\}$. The probabilities stop at 0.40 because the time required to test these probabilities is prohibitively high in a testbed environment. The interval is such that $i \in \{40 \text{ to } 1100, \text{ step } 20\}$. This creates 477 data points based on the 9 probabilities and the 53 intervals. At each of these data points, the file is transmitted 30 times in order to gather a mean and a variance.

This construct models link outages that arrive in a Poison distribution and that last in a geometric distribution. For an interval i, regardless of length, the status of the link in a particular interval is independent of all of intervals. The status of the link is determined based strictly on the probability p. Every interval the "dice are rolled" determining if the status of the link is up or down. To fully understand the model, one must understand how long the wink will last.

The length, in intervals, of the wink is geometrically distributed. The geometric distribution provides a probability for how many consecutive intervals the link is down. In other words, how many intervals is it down until it is up. The expected value of the geometric distribution is $\frac{1}{p}$. In this case the probability of the link being up is $1-p$. On average this means on try number $\frac{1}{p-1}$ will get a failure after a series of successes. Because the expected value includes the interval that failed, one (1) must

be subtracted. The expected number of intervals between winks, X, is defined by this equation.

$$E[X] = \frac{1}{1-p} - 1 \tag{3.1}$$

For similar reason discussed above, the expected length of a wink, X, is defined as follows.

$$E[X] = \frac{1}{p} - 1 \tag{3.2}$$

Table 3.1: The relationship between the probability of a wink, the expected intervals between winks, and the expected length of a wink.

p	**Intervals between Winks**	**Length of Wink**
0.05	19.00	0.05
0.10	9.00	0.11
0.15	5.67	0.18
0.20	4.00	0.25
0.25	3.00	0.33
0.30	2.33	0.43
0.35	1.86	0.54
0.40	1.50	0.67
0.45	1.22	0.82
0.50	1.00	1.00
0.55	0.82	1.22
0.60	0.67	1.50
0.65	0.54	1.86
0.70	0.43	2.33
0.75	0.33	3.00
0.80	0.25	4.00
0.85	0.18	5.67
0.90	0.11	9.00
0.95	0.05	19.00

3.3.3 Initial Transmission Failure. The first part of the model portrays how many of the intervals fail on the first attempt to transmit. If a file was comprised of

10 intervals, then zero, some, or all of intervals could fail on their first transmission, e.g. the link is down. The intervals that fail incur the cost of retransmission. The number, n, of successful intervals required to send a file is a function of the time it takes to send the file under perfect conditions, $t_{perfect}$, and the interval i.

$$n = \frac{t_{perfect}}{i} \tag{3.3}$$

Employing the binomial distribution, the probability of a r number of failures is easily calculated. Equation 3.4 provides the probability of incurring r failed transmissions of n intervals.

$$\binom{n}{r} \cdot p^r \cdot (1-p)^{n-r} \tag{3.4}$$

The expected value of a binomial distribution is $n \cdot p$. For example, if there are 10 intervals and the probability of a failure is 0.2, then the expected number of failures is 2. The probability of having 10 failures is low but not impossible: 0.2^{10}.

If each failure costs c, then the expected cost is defined in Equation 3.5.

$$\sum_{r=0}^{n} \binom{n}{r} \cdot c \cdot p^r \cdot (1-p)^{n-r} \tag{3.5}$$

The probability of each possibility is multiplied by the cost of a failure. This assumes that there is a fixed cost, c, for each retransmission. TCP does not incur a fixed cost for retransmissions. The second part of the model addresses the cost of retransmission.

3.3.4 Retransmission Cost. According to Wright and Stevens [30], the first retransmission in TCP occurs after approximately 1.5 seconds. If no ACK is received for the retransmission, TCP waits exponentially longer for every subsequent transmission. The next retransmission occurs at 3 seconds; followed by 6, 12, 24, 48, 64, 64, 64, ... In most implementations, and in Linux, TCP waits up to a total of nearly a nine minutes before the an error occurs. If TCP does get an ACK for any

of these retransmissions the timer goes back to 1.5 seconds. Table 3.2 shows how the delay accumulates over successive retransmission failures.

Table 3.2: These costs represent the cost of exponential back-off in TCP's retransmission scheme. The cost of each subsequent failed retransmission is accumulated in the right most column.

Hit	**Cost (s)**	**Cumulative Cost (s)**
1	1.5	1.5
2	3	4.5
3	6	10.5
4	12	22.5
5	24	46.5
6	48	94.5
7	64	158.5
8	64	222.5
9	64	286.5
10	64	350.5
11	64	414.5
12	64	478.5
13	64	542.5

The probability of consecutive failed retransmissions is modeled with a geometric distribution. The probability of a single retransmission failure is the probability of the link being down, p. Consider a failed retransmission to be a *hit*. The probability of encountering h consecutive hits using probability p is defined as $(p^h)(1-p)$. Combining the cost Table 3.2 and the geometric probability distribution, the following formula calculates the expected cost of retransmissions.

$$\sum_{h=1}^{13} c_h \cdot p^h \cdot (1-p) \tag{3.6}$$

The value of c_h is a lookup into cumulative cost column in the cost Table 3.2.

3.3.5 Combining the Two. The first part of the model (Equation 3.5) calculates how many of the intervals need to be retransmitted. The second part (Equation 3.6) calculates the expected cost of the retransmission. Combining to the

two yields the following equation.

$$\sum_{r=0}^{n}\sum_{h=1}^{15}\binom{n}{r}\cdot p^{r}\cdot(1-p)^{n-r}\cdot r\cdot c_h\cdot p^{h}\cdot(1-p) \tag{3.7}$$

Notice that second part of the model is multiplied by r to account for the cost of *each* failed retransmission. This formula further simplifies to Equation 3.8.

$$\sum_{r=0}^{n}\sum_{h=1}^{15}\binom{n}{r}\cdot p^{r+h}\cdot(1-p)^{n-r+1}\cdot r\cdot c_h \tag{3.8}$$

To account for the time required to transmit the interval, the interval length, i, is added to the cost. For h hits, the time to transmit is $h \cdot i$. The cost is now as follows.

$$c_h + h \cdot i \tag{3.9}$$

3.3.6 Further Insights. If this model is correct and only a single interval fails and is successfully retransmitted on its first attempt, it causes a 1.5 second hit. In the case of the 20MB file that takes 1.86 seconds to transmit, the smallest time to send the file with at least one retransmission is 1.86 + 1.5 = 3.36 seconds. The preliminary data does not support this.

The smallest time greater than 1.86 is near 2.1 seconds, 200ms more. The next lowest times center around 2.25 seconds, 500ms more. TCP has two internal timers. One is 200ms and the other 500ms. The 200ms time is for fast retransmit and the other is for regular retransmit. For further information about these timers, the reader is referred to Wright and Stevens [30]. It does appear from the preliminary data that the earliest retransmission occurs at 200ms and the next at 500ms.

Once an interval fails, the first possible retransmission attempt is at 200ms. The interval fails because the link is down. If $i > 200$, the link is down for longer than 200ms. When the chance for the retransmission occurs the link is still down. Similarly, the 500ms retransmission opportunity is no longer available when $i > 500$.

To account for this behavior the cost Table 3.2 is adjusted to vary for different interval ranges and displayed in Table 3.3. For $40 < i \leq 200$ is extended to allow for 15 hits. This still allows for the maximum of 9 minutes while incorporating the 200ms and 500ms. For $200 < i \leq 500$, the table provides for 14 hits. For $i > 500$, the table remains the same. This thesis looks at only intervals up to 1100ms. It is contemplated as the the intervals grow greater that 1.5, 3, 6, etc that the tables would need to be adjusted as well.

Table 3.3: These costs represent the cost, in seconds, of exponential back-off in TCP's retransmission scheme. The cost of each subsequent failed retransmission is accumulated in the right most column.

$i \leq 200$			$200 < i \leq 500$			$i > 500$		
Hit	Cost	Accum. Cost	Hit	Cost	Accum. Cost	Hit	Cost	Accum. Cost
1	0.2	0.2	1	0.5	0.5	1	1.5	1.5
2	0.5	0.7	2	1.5	2.0	2	3	4.5
3	1.5	2.2	3	3	5.0	3	6	10.5
4	3	5.2	4	6	11.0	4	12	22.5
5	6	11.2	5	12	23.0	5	24	46.5
6	12	23.2	6	24	47.0	6	48	94.5
7	24	47.2	7	48	95.0	7	64	158.5
8	48	95.2	8	64	159.0	8	64	222.5
9	64	159.2	9	64	223.0	9	64	286.5
10	64	223.2	10	64	287.0	10	64	350.5
11	64	287.2	11	64	351.0	11	64	414.5
12	64	351.2	12	64	415.0	12	64	478.5
13	64	415.2	13	64	479.0	13	64	542.5
14	64	479.2	14	64	543.0			
15	64	543.2						

The cost is now defined in Equation 3.10. Incorporating Equation 3.10 into Equation 3.8, the final formula is defined by Equation 3.11.

$$c_{h,i} + h \cdot i \tag{3.10}$$

$$\sum_{r=0}^{n} \sum_{h=1}^{15} \binom{n}{r} \cdot p^{r+h} \cdot (1-p)^{n-r+1} \cdot r \cdot (c_{h,i} + h \cdot i) \tag{3.11}$$

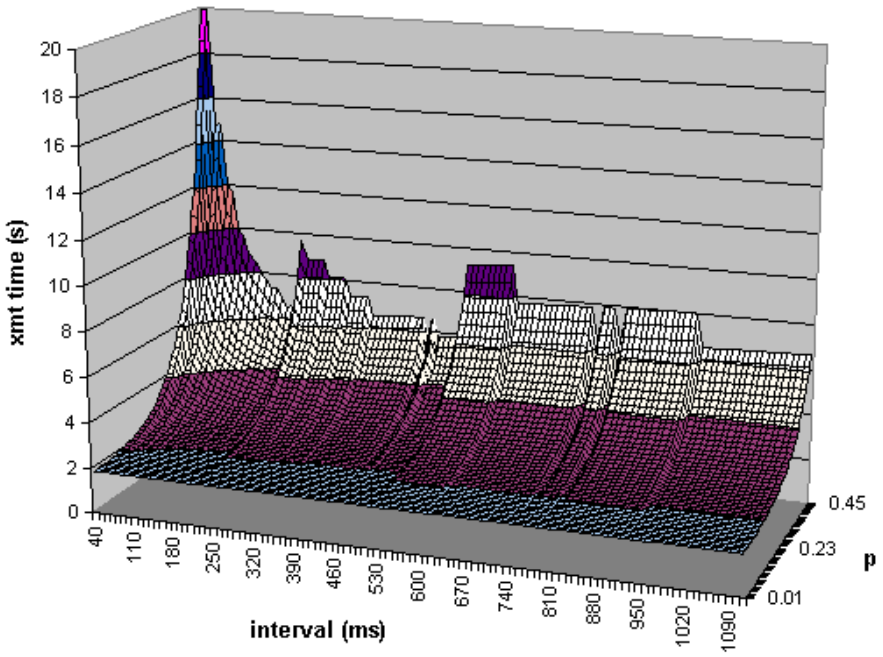

Figure 3.6: Graphs Equation 3.11.

The model produces the graph in Figure 3.6. The worst performance is where the interval is low and the probability is high. Curious though, near the interval lengths of 200ms and 500ms deep valley can be seen. These correlate to the TCP retransmission timers lengths.

3.3.7 Strategic Buffering Model. Using strategic buffering is an attempt to over come link wink. The best case scenario has a buffer in the node immediately before the winking link. Ideally this buffer always has enough packets on hand to transmit while the wink is up. In this case the buffer fills up, potentially with the entire file. The buffer retransmits as much of the buffer as the up time of the link will allow. In this scenario, the source never invokes the TCP exponential back of sue to the link winking. The source transmits smoothly to the buffering node.

The perfect model has perfect knowledge of the link's status and only transmits when the link is available. In this model, independent of file length and interval length, only p is a factor. The function for this is $\frac{1}{(1-p)}$. Figure 3.7 shows this.

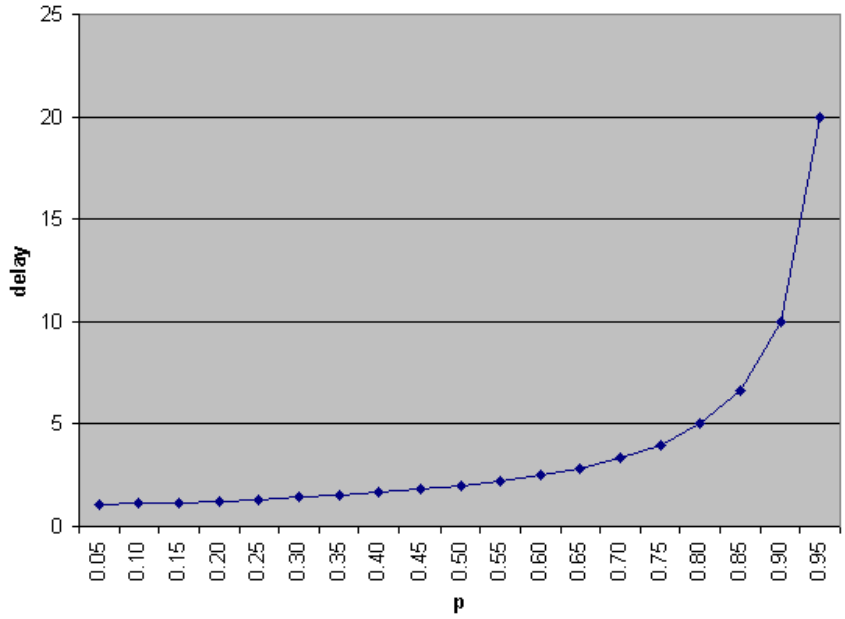

Figure 3.7: Graphs the delay incurred with a perfect retransmission scheme.

To make the earlier model reflect the new scenario requires adjusting the cost of retransmission. A few assumptions must be made to discuss this. The winking link's effective bandwidth needs to be less than the available bandwidth between the source and the buffering node. The source must continue to send more packets in the temporary absence of ACKs from the destination. The congestion control algorithm needs to provide congestion windows that instead of shrinking in the absence of ACKs actually might grow. Creating a congestion control algorithm that performs as described is left for future work.

Using the above assumptions, the analytical model is adjusted to change the cost of a retransmission. The cost of a failed transmission using strategic buffering and retransmission is optimally the cost of the lost interval. Since the buffer is full, once the link becomes available transmission occurs. The formula is changed as follows and displayed in Figure 3.8.

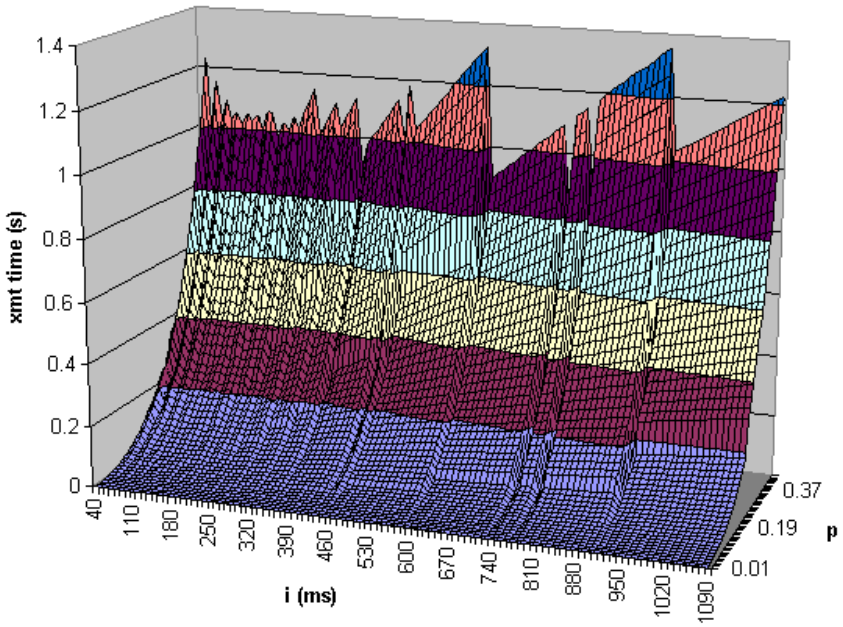

Figure 3.8: Graphs Equation 3.12.

$$\sum_{r=0}^{n}\sum_{h=1}^{15}\binom{n}{r}\cdot p^{r+h}\cdot(1-p)^{n-r+1}\cdot r\cdot(h\cdot i) \tag{3.12}$$

Figure 3.8 shows the cost of retransmission predicted by the model. It predicts the delay is higher as the probability increases. Across the intervals it appears to relatively stable. The spikes occur because the number of intervals required to send the entire file are rounded up to the next highest interval. If it takes 46.25 intervals to send the file, then 46.25 is rounded up to 47.

3.4 Testbed Implementation

This section discussed each of the components utilized in creating the testbed. The end of the section addresses security and specific issues encountered during the development process.

3.4.1 Private Networks. Private IP networks are networks with addresses that are only accessible from within the network. The privacy of these addresses is

enforced using routing mechanisms. Furthermore certain IP address spaces, such as 192.168.0.0, are defined as private and routers are designed to not forward this traffic.

The hybrid communication testbed is a dynamic meshing of many private networks utilizing many forms of communication. For example, assume a node utilizes three forms of communication: an 802.11b omni-directional network card, a directional infrared transceiver, and a satellite link. Each forms a separate physical network, perhaps with its own address space. The integration of these networks occurs at either the data-link layer or the network layer of the Open System Interconnection (OSI) model.

One of the following scenarios describes how traffic integrates across these networks. First, communication is not forwarded across the individual separate networks. In this case nodes on each network do not know of each other and do not communicate. Second, the networks are bridged at the link layer; in effect merging the two (or more) private networks into one single network. In this case, the nodes on each network have the ability to discover and communicate with each other. Third, the node is a router between the two (or more) private networks; merging the networks at layer three.

From the above circumstances, three assumptions are made for this thesis. First, Internet Protocol (IP) is used as the Network layer addressing and routing scheme. Second, the private networks are not bridged together, the nodes route traffic between private networks. Third, the use of hierarchical routing schemes (such as traditional IP addressing/routing) is not present between these meshed private networks.

Figure 3.9 shows the nodes participating in private networks. The function of the private networks is semantically similar to the link layer. In Microsoft MCL [7], meshing occurs at the link layer. The MCL however is limited to the 802.11 Ethernet. A major contributor to the success of the Internet is the fact that the ubiquitous IP protocol does not depend on the underlying physical media. Instead, IP is capable of routing over any physical media. In Figure 3.9, Node 101 communicates with node 103

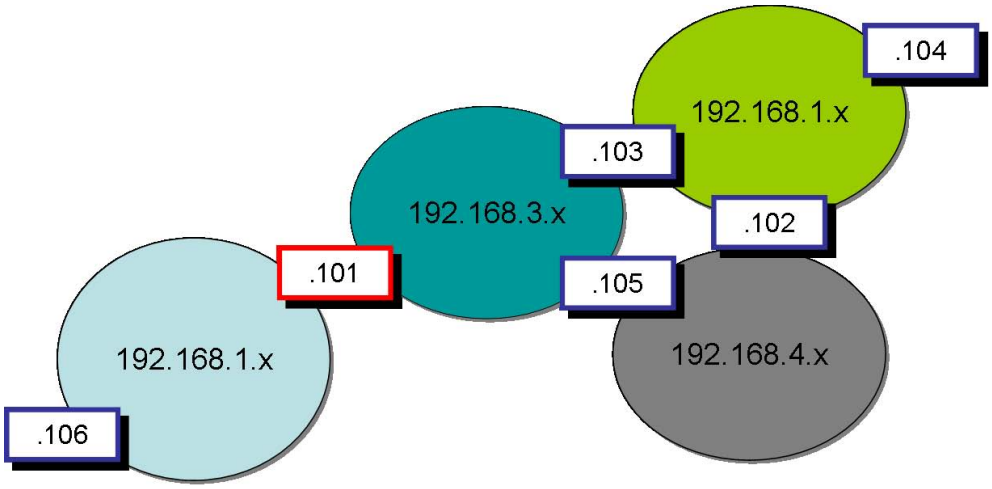

Figure 3.9: Mesh of Private IP Networks. Gateway nodes indicated with boxes.

directly on the 192.168.3.x network. Node 101 communicates with node 104 through node 103 or alternatively through 105 and 102. In the testbed for this thesis four nodes are arranged in a linear fashion. While this is a admittedly a simple setup, it is semantically identical to many complex scenarios.

The presumed dynamic environment of hybrid communication allows networks to connect to other networks without regard to any hierarchy. As a node moves through a geographical region the node dynamically connects and disconnects to various networks as needed. Each of these networks provides its own IP addressing scheme. This could be Dynamic Host Configuration Protocol (DHCP) or static or otherwise. Regardless of how it is handled, the private nature of the IP addresses prevents using the generally recognized hierarchical nature of the public IP address space. For global routing to succeed, other mechanisms are required.

3.4.2 Overlay Network. An overlay network is a network implemented on top of another network or networks. Examples of overlay networks are virtual private networks (VPN) and peer-to-peer (P2P) networks. The overlay network relies entirely on the underlying network(s).

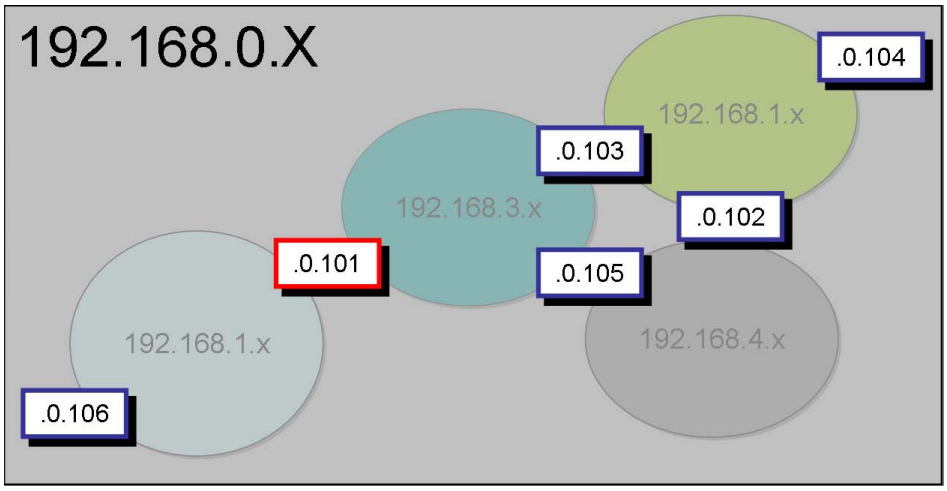

Figure 3.10: IP network overlaying the previous mesh of private IP networks.

In this testbed, an IP network is overlaid across the private networks. Each node is assigned an IP address globally unique to the overlay network. This overlay network resides between layer three and layer four of the OSI model. The services at Transport Layer act upon the addresses of the overlay network. A node that belongs to three private networks has three private IP addresses.In Figure 3.9, Node 101 has two IP addresses: 192.168.1.101 and 192.168.3.101. The isolation of these IP address spaces allows for local autonomy thereby relieving the global authority of micro-managing various networks. The address management of the global overlay network is out of the scope of this thesis. It is simply assumed that each node participating in the global communication has an address in the overlay network.

In this testbed, Figure 3.10, each node is assigned an overlay network IP address of 192.168.0.x. In order for node 192.168.0.106 to pass a message to node 192.168.0.104 (note that the addresses in the private networks are not globally unique), the overlay address is required. The packets are routed over the private networks, but use the overlay network addresses at various bridging points. All services in the Transport layer and above use the overlay IP. Routing across this overlay network is discussed below. This is not the typical hierarchical routing scheme used in typical IP networks.

Assuming the routing is updated appropriately, the global overlay IP is able to move with the node from one private network to another. For example a mobile node can leave one ad hoc 802.11 network and join another carrying with it the overlay IP address.In Figure 3.10, if node 106 joins the 192.168.4.x network, it is still referenced as 192.168.0.106; although some routing updates need to occur. As mentioned above, the structure of the IP network of the private networks is independent of that of the other private networks. Similarly the structure of the overlay network is independent of the private networks. The overlay network functions the same as regular IP networks. It can be hierarchical and interact with other outside networks.

In order to connect the mesh of private networks and the overlay network, nodes must have routing information; instructions on where to send packets. Each router is a relay, a gateway, to other private networks. Packets are sent from one gateway to next across the overlay network. This is not standard hierarchical routing. This overlay network clearly requires its own routing algorithms in order to manage the routing dynamically as nodes physically move around the mesh of private networks. This particular topic is addressed in other work on the testbed but is out of scope of this thesis.

3.4.3 Routing. The topology of the overlay network is dynamic in nature; reacting to the dynamic environment and requirements placed upon it. Dynamic topology implies that links between nodes exist sometimes and not at others subject to many considerations, both global and local. These considerations include, but are not limited to, mobility, transmission interference, security, congestion, and priority starvation.

At one extreme, a soldier in the bush concerned about avoiding capture, may choose to refrain from using an omni-directional radio frequency transmission (since it can be detected by the enemy). This local concern causes the omni-directional link to be down and without some other form of communication, this node is not transmitting. From an extreme global perspective, a node with a single directional

link positioned between two private networks must alternately point its directional laser in order for traffic to flow through the overall network.

Node mobility results in dynamic topologies, which are both a challenge and an opportunity for routing. For instance, a plane may fly through several networks along its path, breaking existing routes, but providing an opportunity to deliver communication that otherwise would be impossible. The plane may transfer information to and from one remote network and then return near the backbone network ferrying the remote network's information.

A more subtle topological concern arises from low priority traffic starvation. If a channel is consumed by high priority traffic, it is unavailable from the perspective of the lower priority traffic that gets little or no bandwidth. From the perspective of the low priority traffic, the topology of network is broken.

Routing issues are out of the scope of this thesis. However, the dynamic nature of hybrid communication presents issues to routing traffic. For this thesis the route is considered to be *a priori* knowledge. Behind this assumption, finding the proper mix of proactive and reactive algorithms is a significant challenge. In general the more dynamic the environment, the more reactive routing tactics are employed; and the more stable the environment, the more proactive routing tactics are employed. Both proactive and reactive routing rely on the connectedness of the network. Proactive routing adjusts slowly to changes in link status. Proactive routing algorithms may over react to link wink believing the link to be down. Reactive algorithms may discover sub-optimal routes or no route at all during the route discovery process. An open issue is at what point is a routing change used to over come a troublesome link versus assuming the performance hit. Future consideration is needed to integrate the strategic buffering into the routing algorithms, i.e. a route will be available if the traffic hangs out in the router momentarily. See Kim et. al [16] for a discussion of this.

3.4.4 Nodes. A testbed for investigating these issues has been developed. The testbed consists of Linux (Fedora Core 4) workstations. These workstations are connected to a single, wired network as a back channel to collect data and configure the nodes. In order to test a variety of communication media, each node can be configured with Bluetooth, 802.11, infrared, laser, etc.

On the back channel network (see Figure 3.11) is a server to be used to gather information from the nodes. From this information the server makes decisions about topology and routing. These decisions are communicated to the nodes through the back channel network. These instructions turn on or off various network cards in the nodes. These instructions also include the routing tables for each node.

On each node a modular software router, developed at MIT and called Click [17], serves to route packets. Click elements are written in C++. Elements are connected to allow packets to flow between them, mimicking the behavior of hardware routers. Each element processes the packets, potentially manipulating, forwarding, or destroying each one. Examples of elements include queues, network interfaces, routing tables, network address translators, etc. Click runs as a user process in Linux. In previous versions of Linux it could run in kernel mode, exhibiting a commensurate speedup. For our experiments, only relative speedup was of concern, hence running in user mode was sufficient.

3.4.5 Intermediate Acknowledgments. Once custody is assumed, an intermediate acknowledgment (IACK) is transmitted from the assuming router to the source. The IACK contains an acknowledgment number of the highest continuous byte number seen, similar to a normal ACK. The IACK also contains a window size similar to a normal ACK. This IACK passes through all of the buffering routers and to the source. Each of these intermediate routers now knows to flush their buffers and to not take custody themselves. The source knows not to retransmit the packets that are ACK'd by this IACK. The source however does not remove these packets from its window as if a normal ACK was received, because the IACK is not notification that

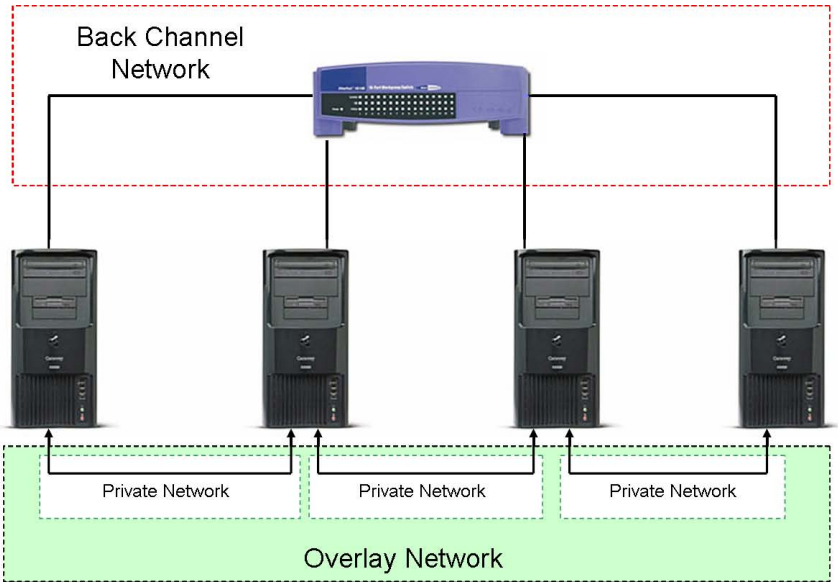

Figure 3.11: Testbed network physical connections showing backchannel, private, and overlay networks.

the packets reached the destination. An IACK is sent every retransmission round to include any recently received packets from the source.

This IACK is similar to the Local Acknowledgment (LACK) described in Split TCP [18] discussed in Chapter Two. The IACK flows to the source, not to the next intermediate router. The IACK is not sent in response to every packet received by the router. The IACK is only sent when custody is in effect for a flow.

As mentioned previously, a regular ACK is not sent precisely because of the semantics of an ACK. A regular ACK indicates to the source that the packet has successfully reached its final destination, which, in this case, it has not. Furthermore the receipt of an ACK is used to calculate round trip time. Using a regular ACK would skew this calculation. Duplicate ACKs are used to invoke fast retransmission. Possibilities exist for the source to receive duplicate IACKs. In summary, a regular ACK has semantic implications to be avoided.

The IACK also contains the suggested window size. This controls the size of the send window in the TCP source. This field in a regular ACK is used to control the amount of data streaming from the source. Depending on the nature of the link

wink more or less data may be needed from the source. Manipulating this field in the IACK to optimize overall flow control is left to future research. This thesis ignores the adjustment of the congestion control algorithms other than the effects of the IACK suppressing the retransmission of packets.

It is a zero length packet consisting of an IP header and a TCP header, shown in Figure 3.12. The IACK is based on a pure ACK with a few differences. The Identification field is set to 0xFFFF. The time to live field is set to 255. To identify this as an IACK, the ACK and URG flags are set in the TCP header flags field. This allows our modified Linux TCP to recognize the IACK. The ACK field is set to the highest sequence number in the buffer up to the first missing packet or the end of the buffer. After the IACK is sent, the entire buffer is traversed and a copy of each packet is transmitted.

A few notes about some of the fields in the IACK. The Identification field is a monotonically increasing packet identifier used by IP. Since this packet is being inserted into the conversation mid stream, there was concern about what the value of this field should be. It is set to 0xFFFF. The value 0x0000 appeared to cause trouble in prototyping. In the Linux implementation of IP, the fact that 0xFFFF is not in sequence does not appear to cause a problem. Similarly the TCP Sequence Number field raised similar concerns. The sequence number is set to 0x00000000. The code for handling the IACK in TCP was inserted before the sequence number was validated. The window size field could be useful for future congestion control algorithms to increase or decrease the flow of packets into the buffer. For now, it is set to the last window size seen from the destination.

In Linux's TCP code, a handful of changes are required to properly handle the reception and processing of an IACK. A flag (**iacked**) was added to the control block (**struct tcp_skb_cb**) for each packet. If true, this flag indicates the packet has been IACK'd. When an IACK shows up, the flag is set for every transmitted packet whose sequence number is less than the ACK in the IACK packet. In the

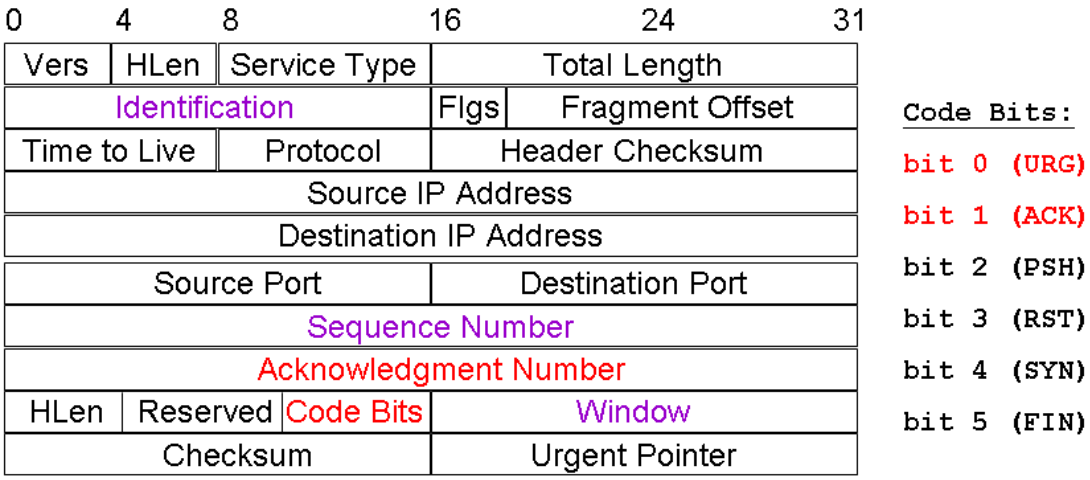

Figure 3.12: IACK packet's relevant fields in the TCP and IP Header.

retransmission function (**tcp_retransmit_skb**) checks this flag. If true, the packet is not retransmitted. Three files were changed: tcp.h, tcp_input.h, and tcp_output.h.

It is believed that the careful employment of a SACK packet can replace the IACK. A segment that is SACK'd it not retransmitted and it is not flushed from the send buffer either. This is semantically similar to the IACK but would not require modification in the the source's TCP. This is left for future consideration.

3.4.6 TCPStore. A Click element handles the buffering and retransmission of flows. This element, TCPStore, keeps a *flow control block* to keep all of the information about a flow. These control blocks are stored in nested linked lists. Each flow control block has its buffer of packets and a pointer to the flow control block for the other half of the TCP conversation. As packets arrive, the original packet is forwarded on if custody has not been taken for this flow, and a copy of the packet is stored in the buffer.

Each packet has a *packet control block* created for it to track the state of the packet and to hold data extracted from the packet. For example the packet's sequence

number is used frequently so instead of extracting the data and flipping the bits every time it is needed, the sequence number is extracted and manipulated once. The packet control blocks for each flow are stored in a sorted linked list, sorted on sequence number. Once a packet is stored the state of the flow is changed as needed. If a duplicate packet arrives, the packet control block has a pointer to the original packet. This packet is removed and the latest copy is kept in its place. The motivation for this is that a duplicate packet may have an acknowledgment number that is higher than the acknowledgment in the original packet.

Once the packet is stored in the buffer, if the packet is an acknowledgment the sibling flow control block handles the removal of packets from its buffer. A walk through the sorted list pops off any packets that are acknowledged by this packet. Before walking through the list a check is made to see if this packet is a pure ACK. If three consecutive duplicate pure ACKs are observed, the needed packet can be retransmitted instantly from the buffer.

A separate thread runs to check for custody conditions, perform garbage collection, and retransmit data. This task executes and then sleeps for 10 milliseconds. Upon waking, this task loops through all of the flows. For each flow the following process takes place. First, if there is only one packet in the buffer it is inspected to see if this is a loitering packet. If it is then it is removed. The flow is checked to see if it can be collected for garbage (if the flow and its sibling are closed). Next if there is no custody for the flow, a check is made to see if custody should be taken. If custody of the flow has been taken, then retransmission occurs and an IACK is Sent.

3.4.7 Notable Implementation Issues. Inserting a retransmission scheme in the middle of a TCP causes a few exceptional cases. Three factors combine to create irregular behavior: intense retransmission, duplicate acknowledgments, and a lossy channel.

With the same packet arriving multiple times at a buffering node, duplicate packets must be replaced in the buffer instead of storing multiple copies. This is not

as simple as looking in the buffer for a packet with the same sequence number. Some packets, like ACKs, have no data so the next packet will have the same sequence number. In our particular case a duplicate packet is defined as having the same sequence number, TCP header flags, and length. If such a match is found, the new packet replaces the old. The new packet can have a higher ACK. The newer packet can also have newer time stamp, if RFC 1323 is being used.

Another issue with RFC 1323 arises. The RFC describes that the algorithm yields "certain unlikely circumstances". It is unlikely to occur and less likely to cause issues in a regular TCP environment. Because strategic buffering has been injected into the stream this circumstance occurs with some regularity. Essentially packets are discarded if their time stamp is older than the time stamp of other packets near them in the sequence number space. The reader is deferred to the RFC for precise details. The destination was dropping some packets and requesting they be sent again. The buffering node would send them again, only to have the destination continue to drop them because of their bad time stamp. The fix involves setting the time stamp of all packets sent in response to duplicate ACKs to the most recent value.

Due to the intense and repeated retransmission of the buffer, the destination receives the same packet multiple times. The destination can also receive a significant number of packets after missing the first packet of the buffer. This situation causes a large number of duplicate acknowledgments to be sent. Letting these duplicate acknowledgments reach the source triggers unnecessary retransmissions from the source. The buffering node allows one of these duplicate ACKs through. The ones that follow are discarded.

TCP receivers discard any duplicate packets without issuing an acknowledgment. This is not true for duplicate pure ACKs. A pure ACK is defined as a zero length packet, i.e. only a TCP header with the ACK flag set. When TCP receives duplicate pure ACK's, it assumes a packet was lost, and needs to be retransmitted. The custodial retransmission of pure ACKs needs special attention. Lost ACKs are

acceptable in TCP. As the buffer is retransmitted repeatedly, the ACK is sent only twice and then discarded. The repeated sending of an acknowledgment has semantic implications on congestion control algorithms. Furthermore, acknowledgments are cumulative so losing a few is insignificant.

Another oddity discovered in prototyping is the loitering packet. The last packet sent before a lull in conversation and the final ACK of the conversation are curious cases. These packets are not ACK'd and therefore are not removed from the buffers. If these are pure ACKs, they are removed after two retransmissions. These loitering packets tend to be pure ACKs and are eliminated after two retransmissions. If custody is not taken for the flow these packets can hang around for some time and may cause custody to be taken. Periodically the buffer is investigated for this loitering packet. If it qualifies as a loitering packet it is removed.

3.4.8 Security Considerations. Information security is a three fold construct consisting of confidentiality, integrity, and availability. Encryption protects confidentiality. Asymmetric encryption techniques provide a mechanism for protecting integrity. For example, signing an email address with an individual's certificate. The message is essentially guaranteed to be from the holder of the certificate and essentially guaranteed to be free of tampering. Availability is addressed using redundant systems, backups, and mechanisms designed to avoid resource depletion. The focus of this thesis is not security; however below each of these aspects are briefly discussed.

The first line of defense in securing a network is using encryption. This encryption occurs at the data link layer, the network layer, and at the application layer. The 802.11 encryption such as WEP and WPA encrypts everything in the packet except the Ethernet header. This header contains the MAC addresses that are essential to the successful transmission of the packets. IPSec runs at the network layer. The IP payload is encrypted leaving the IP header open for routers to read. SSL is an example

of application layer encryption. In this case, the TCP payload is encrypted; leaving the TCP header, such as the ACK, SEQ, flags, and ports available for inspection.

The routers proposed in this thesis require the ability to inspect the TCP header. An encryption method such as IPSec breaks this unless the routers are doing point to point IPSec between each other. This allows the encrypted packets to be unencrypted, inspected, stored, re-encrypted, and finally forwarded. Data Link Layer encryption, by its nature, is point to point. Therefore this method will have no effect on the prescribed packages as the packets are decrypted before moving up the network stack. Application layer encryption strictly affects the payload of the packets; therefore not affecting the function of the routers.

Fabricating and injecting IACKs into the communication stream presents security concerns; primarily concerns of integrity. The IACK is spoofed with the receiver's IP address and appears in all respects to be from the receiver even though, logically, the receiver never generates an IACK. As currently designed, there is no indication where the IACK originated. These packets could be falsely generated by a malicious router along the path of communication. The simple attack is to IACK the sender while never forwarding the packets.

Other security concerns manifest from the temporary storage of the packets. If these packets are not encrypted with end-to-end techniques, the packets are stored free for the taking. Trouble occurs if the node is physically compromised by falling into the adversary's hands. Alternatively if a malicious node is trusted, the traffic is compromised.

All significant security decisions come down to trade-offs between competing interests. The proposed model in this thesis provides increased availability. The increased throughput across troubled link makes the link more available than it is otherwise. Initially, without countermeasures, this comes at the cost of the issues outlined above. Further consideration and investigation should relieve the impact of these effects.

IV. Analysis Link Wink and Strategic Buffering in TCP

4.1 Overview

This chapter details and analyzes the results of the experiments. This chapter first discusses the specifics of the testbed experiments. The first experiment transmits a file using different TCP congestion control algorithms over a troubled link. The second experiment transmits the same file using TCP and strategic buffering with retransmission over the same troubled link. The variables and the factors of the experiments are outlined including data collection and organization. The graphs of the results without buffering and retransmission are presented, analyzed, and contextualized. The model presented in Chapter Three is used to predict the behavior of the strategic buffering and retransmission. This prediction is compared with the results of using strategic buffering. During this discussion some specific points of interest in the data are visited and discussed. Finally the performance gains using strategic buffering are presented.

4.2 Testing

The purpose of the testing is three-fold: to understand the performance implications of link wink on the different congestion control algorithms and to demonstrate the performance benefits of strategic buffering in the presence of link wink and to compare the testbed results to the mathematical model.

The testing takes place on four PCs with the following specifications, shown in Figure 3.11. Each node is connected to the next mode using a 100 Mbit Ethernet switch creating a chain similar to a predetermined route. Each node has Click installed and configured with a Click configuration file.

The private IP networks created between each node are described in the diagram. An IP overlay network is created across all four nodes providing a mechanism for each node to communicate to each node.

Additionally a back channel network is established using a 100 Mbit Ethernet switch to which all the PCs are connected. This back channel is used to execute the

test scripts and to gather data from the experiment. The soft router Click is unaware of the back channel network. The back channel network traffic sends insignificant amounts of commands and data. All of which is sent between experimental trials.

The test simply transfers a 20MB file from Node 1 to Node 4. Node 1 runs a small command line program that reads a directory of files into memory. This program listens on a TCP port for a request of a file, similar to a web server. The file is sent in response.

Node 4 runs a simple program similar in function to a web browser. This program is a command line program specifying an IP, port, file name to request, and a number of iterations. Node 4 for runs a bash script to coordinate and automate the tests. This script, as described in the scenarios below, sets the TCP send and receive windows on Node 1 and Node 4 to 60MB. This script loops through the various winking probabilities, winking intervals, and turns on custody as needed in Node 2. Inside the loops the script requests the file to be transferred from Node 1 to Node 4.

4.3 Factors

This experiment has several possible factors, outlined in the following list. These are items that have potential to affect the outcome of the experiment. Each is discussed briefly in no particular order. The factors selected for examination in this thesis are congestion control algorithm, probability of wink, and length of wink.

Number of nodes - The number of nodes in the route effects the time to send the file. Additional transmission and queuing delays are incurred. Four nodes are used in this experiment.

File size - The length of the file directly effects the time required to transmit. A single file of approximately 20MB is used throughout this thesis.

Transmission rate of NICs - The speed of the network cards used directly effects the time required to transmit. In this thesis, each PC uses 100Mbit Ethernet NICs.

Round trip time - The round trip time affects the number of packets in flight. This number could effect how the congestion control algorithms behave.

Choice of network media - Some media are shared. Some are subject to more outside interference. Choice of media effects the time to transmit. Each PC is connected to the next with an Ethernet cable dedicated only to the two connected PCs.

Send window size - The size of the TCP transmission window effects performance. Linux provides a means of setting the minimum, default, and maximum window size. For this thesis the minimum and default are left unchanged, but the maximum is set to 60MB.

Congestion control algorithm - Performance of congestion control algorithms varies under different conditions. Each is designed to perform best under specified conditions. This is a factor in this thesis.

Socket configuration - In opening a socket, various options exist for tuning the socket for optimal performance as needed. In addition to options, the sending and receiving of data can be handled in a variety of fashions. In this thesis the only option set is the maximum window size. The read() and write() C functions are implemented in a generally simple method.

Other traffic - The load on the network effects the transmission time. Congestion and contention increases delay. In this thesis there is no other traffic on the network.

Size of retransmission blast - The unsophisticated method of retransmission used in this thesis sends everything in the buffer up to a constant size in order to avoid overloading the NIC. Varying this amount effects the transmission time. If only one packet is sent at a time performance will suffer. If too many are sent the time is wasted sending packets that are lost in the NIC.

Retransmission frequency - In this thesis the buffer is retransmitted every 10 milliseconds. This was determined by trial and error in the present testbed

environment. It primarily depends on the bandwidth provided by the NIC. More and less than 10ms yielded longer times in prototyping the test.

Size of buffers - This thesis assumes an unlimited amount of space for buffer storage. The thesis suggests using large buffers, but infinite is excessive. Fairness and performance have to be considered in a multi-flow environment. This thesis has only a single flow with all the buffer space required.

Get more data from source - Throughput over the troubled link is maximized only if the buffer is never empty. Congestion control algorithms, properly written, could keep the buffer full. In the extreme case, if the link is down long enough to transmit the entire file into the buffer, it should.

Implications of IACKS - Once a packet arrives in the buffer, the source need not send it again. This functionality is implemented using Intermediate Acknowledgments. This uses cumulative IACKs. Not using them or optimally implementing them affects performance.

Probability of winking - This is a factor in this thesis. The likelihood of the link being available directly effects the time to transmit. Period of winking The "arrival" of the outages effects the time of transmission. This thesis assumes a Poison arrival process.

Interval of winking - This is a factor in this thesis. The length of time the link is down directly effects the time of transmission.

Direction of winking - The outage of the link occurs in one or two directions. One direction allows for data packets to be dropped while the acknowledgments are allowed to passed, or vice versa. A bi-directional link drops both data packets and acknowledgments during an outage. This thesis uses a bi-directional wink.

Location of winking - The location of the wink has potential to effect the time of transmission. It is either closer to the source or to the destination. This thesis assumes the wink is between Node 2 and Node 3 in a four node route.

Custody - The condition under which the custody of retransmission occurs affects the time of transmission. This thesis assumes full time custody regardless of the environment. It is not efficient when the link is generally available.

4.4 Congestion Control Algorithms

The congestion control algorithms used include all of the algorithms that ship with Linux 2.6.14.7. These are Reno, Vegas, Hybla, Highspeed, H-TCP, Scalable, Westwood, and BIC. They are described in Chapter Two. Each manipulates the TCP congestion control in differently. In addition to these eight algorithms, a ninth one is introduced called Unfair that attempts to maximize the congestion window. A more detailed discussed is covered Chapter Three. The intent of investigating each of these is to understand if there exists a difference in their performance under stress.

4.5 Trials

Each trial is a transmission of the file. Under perfect conditions, the 20MB file takes 1.86 seconds to transmit. This time is defined to be the total time from the moment the first byte is received until the moment just after the last byte is received. Perfect conditions is defined has no winking; $p = 0$ and $i = 0$. For each data point (p,i) the file is transmitted 30 times for each congestion control algorithm. With 9 algorithms, 477 data points, 30 trials, and estimated average 10 seconds per trial, the total estimated runtime $(9 \cdot 477 \cdot 30 \cdot 10)$ is approximately 357 hours. In order to reduce this estimate, if any trial exceed 600 seconds the rest of the 30 trials are canceled. This exception should be rare and only affect the data points in the most extreme cases. This is also the motivation for only investigating p ¡= 0.40.

4.6 Results with no Custodial Buffers

The initial qualitative analysis on the congestion control algorithms appears that they generally behave the same with the exception of Highspeed. Looking at Figure 4.1, the mean transmit time for each congestion control algorithms across

the probabilities, excluding Highspeed, each increases at a similar rate. At 0.05, the transmit time is 3 seconds increasing up to 10 seconds at 0.40. The following graph visually displays the congestion control algorithm's behavior across increasing probabilities. The variances across the probabilities increase at similar rates. It is worth noting that the variances all increase to a level that having only 30 samples is not sufficient to have a tight confidence interval.

Further observations come from the performance of the congestion control algorithm across the intervals. Again with the exception of Highspeed, all of the algorithms' means increase as the intervals get to shortest of lengths. Figure 4.2 visually demonstrates the similarity each algorithm has to each other. It is contemplated that the extremely short round trip time may prevent the algorithms from differentiating themselves. Highspeed stands out from the other algorithms. The transmit times suffer significantly, consistently 3 times higher from 0.05 to 0.40. This can be seen in figures 4.1 and 4.2. Further investigation into this algorithm is needed to understand why this occurs.

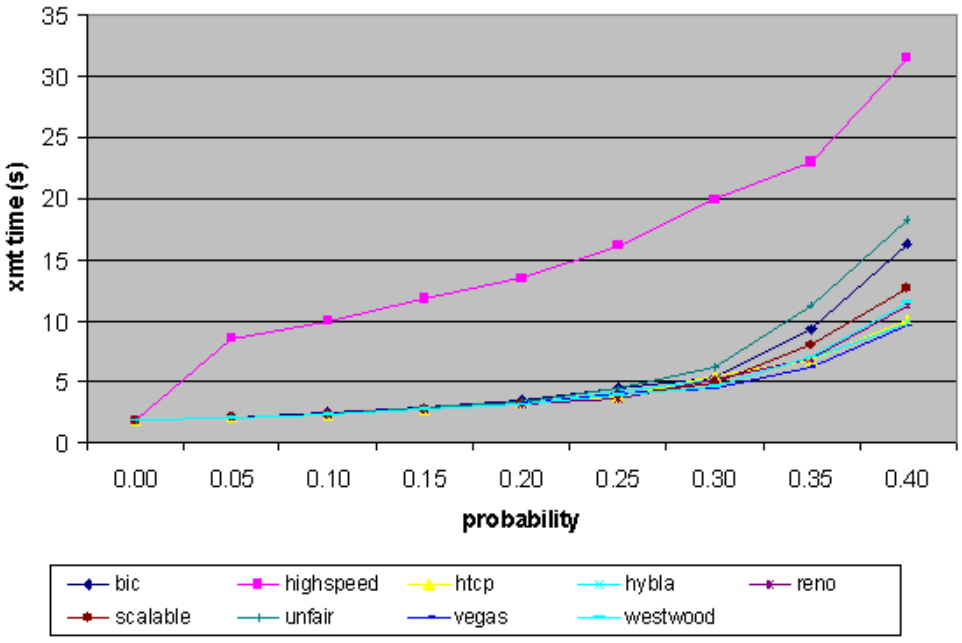

Figure 4.1: Average observed time of file transmission plotted for probabilities aggregated across all interval lengths.

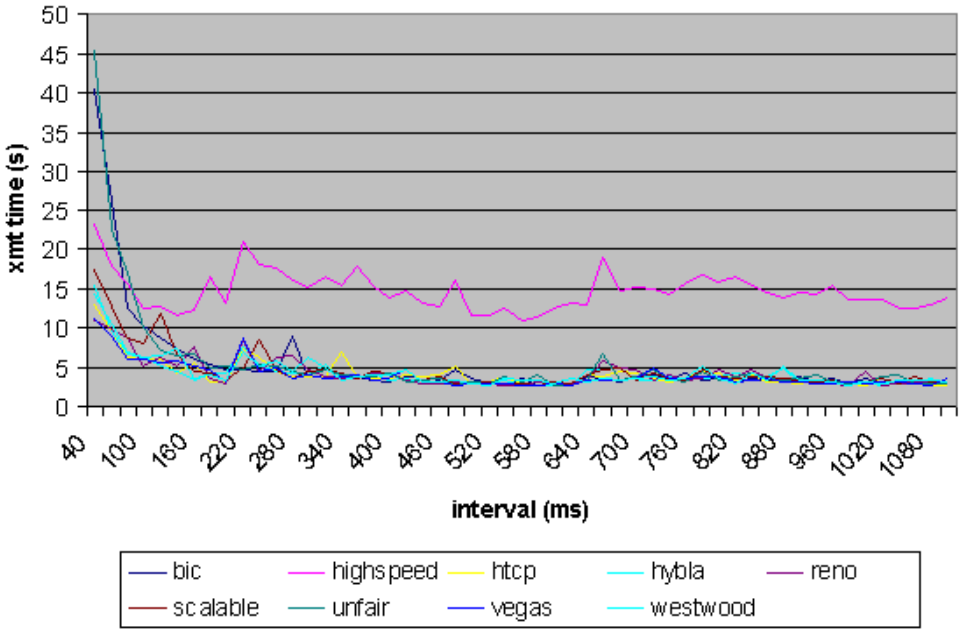

Figure 4.2: Average observed time of file transmission plotted for interval lengths aggregated across all probabilities.

Looking at the variance across intervals, nearly half are over 100 and several are over 1000. These higher variances do not necessarily increase as the interval gets shorter. The sporadic nature of the data suggests more samples are needed in order to have a tighter confidence on the means. This exercise is left to future simulation investigation.

Removing Highspeed from the calculation and not displaying the smallest intervals to get good scaling, Figure 4.3 is the mean of all congestion control algorithms (sans Highspeed) and all probabilities broken down by intervals. Two curious low spots exist around 200ms and 500-600ms. These two valleys in the graph match the valleys witnessed in Chapter Three's mathematical model. These valleys have the lowest averages and the tightest confidence intervals of the dataset.

TCP has retransmit timers triggering at 200ms and 500ms. These valleys are likely to be caused by these timers as demonstrated by the model and the data. The length of the link wink interval and the TCP timers each create a frequency. At

these two low spots, a sort of harmony exists between the two while the other interval lengths create a dissonance with TCP timer frequency. Figure 4.4 demonstrates that the valley exist even at the worst probability of 0.40.

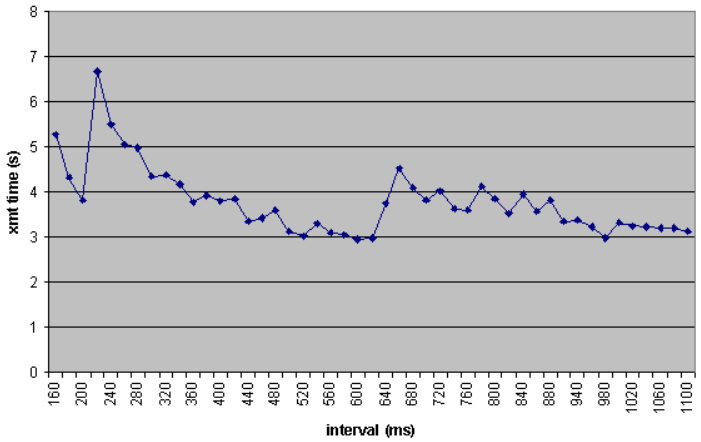

Figure 4.3: Average time of file transmission for intervals aggregated across all probabilities and all algorithms.

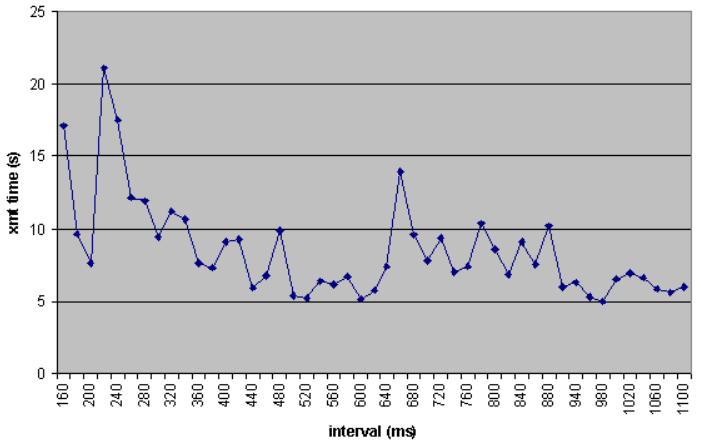

Figure 4.4: Average time of file transmission for intervals at probability 0.4 aggregated across all algorithms.

Each of the following graphs (Figures 4.5– 4.13) is the set of trials for each of the congestion control algorithms. The z-axis is average send time for the file over 30 runs for each (p,i) pairs. The long x-axis is the interval. The short y-axis is the probability. The z-axis is cut off at a maximum of 60 to visually scale the graph. This brings out the terrain of the graph better. The highest spikes occur at the highest p (0.40) and the lowest i (40). The following table shows these highest values that are eliminated from the graphs. One last footnote, the worst area of Highspeed was taking too long to reasonably test. This area is from $0.30 < p \leq 0.40$ and $40 \leq i \leq 160$.

Table 4.1: Data too high to plot in graphs.

Algorithm	**Xmt time (s)**
BIC	217.6
Scalable	105.9
Unfair	275.0
Highspeed	-

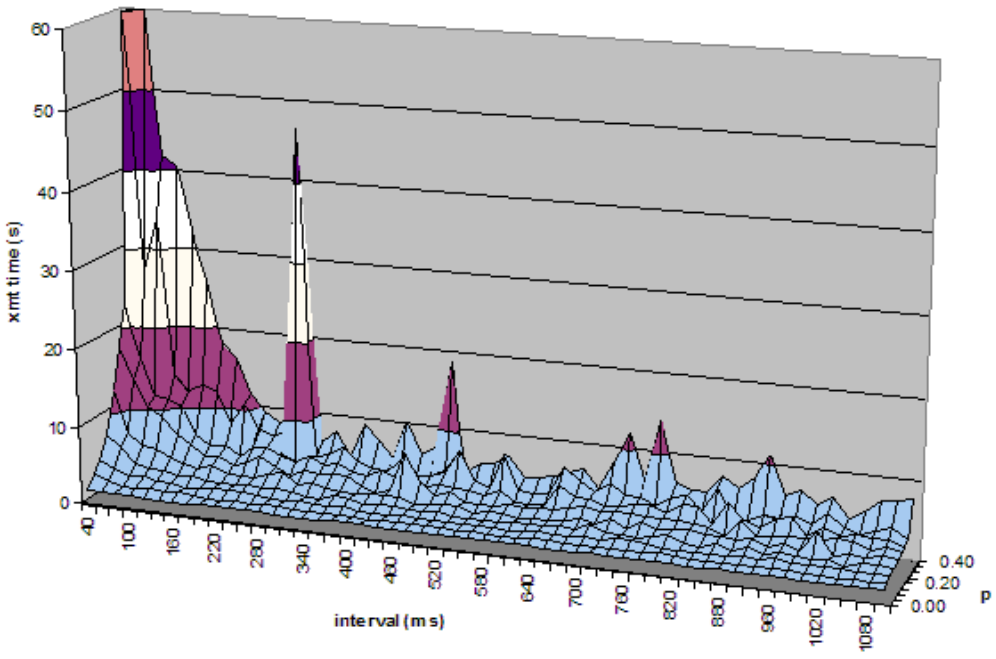

Figure 4.5: Average observed time of file transmission plotted for link wink probability and interval length using BIC congestion control algorithm.

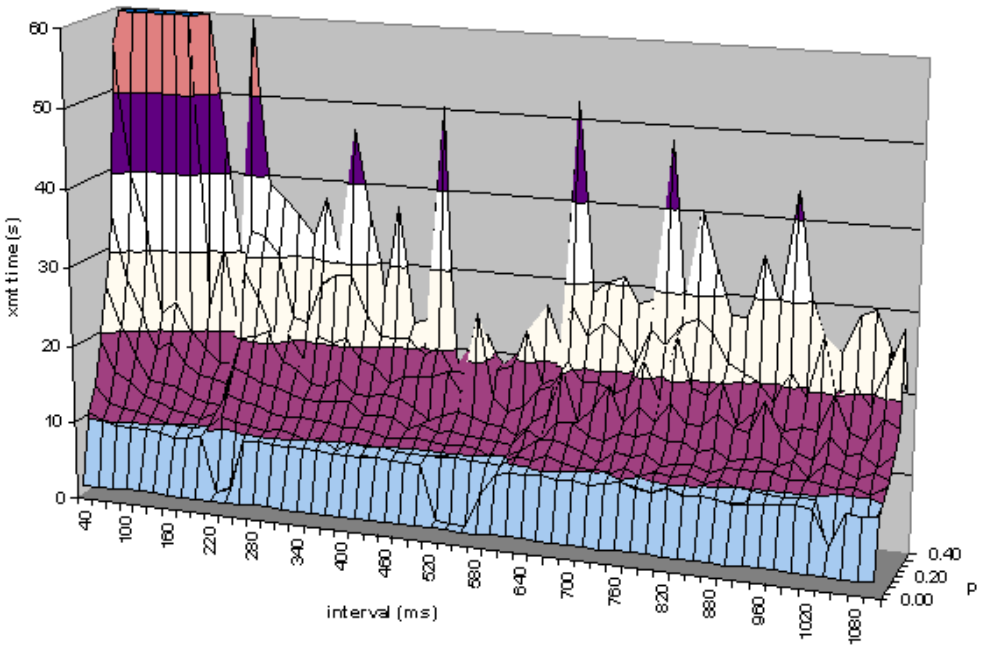

Figure 4.6: Average observed time of file transmission plotted for link wink probability and interval length using Highspeed congestion control algorithm.

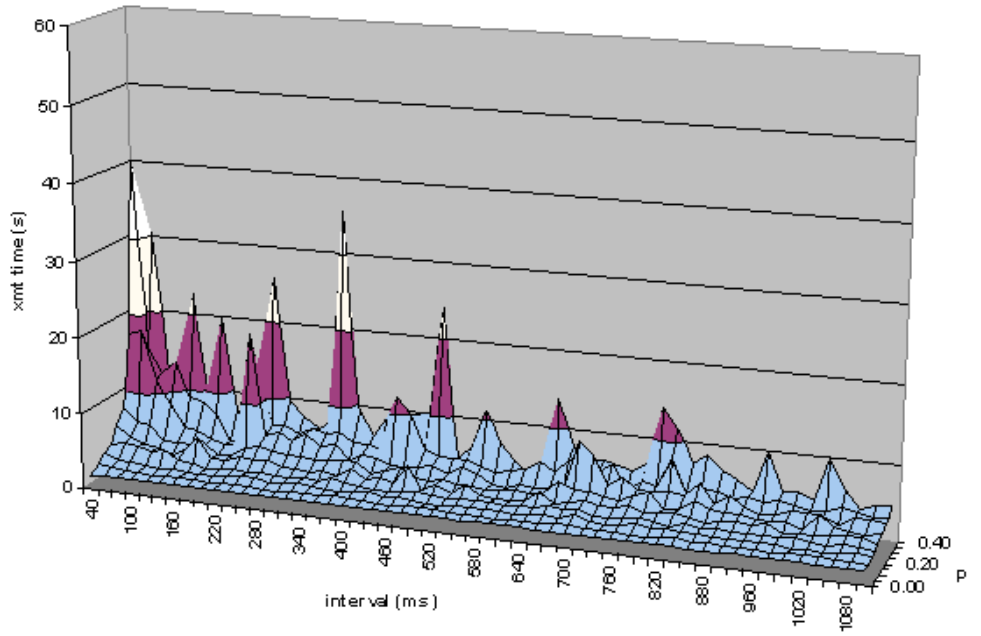

Figure 4.7: Average observed time of file transmission plotted for link wink probability and interval length using H-TCP congestion control algorithm.

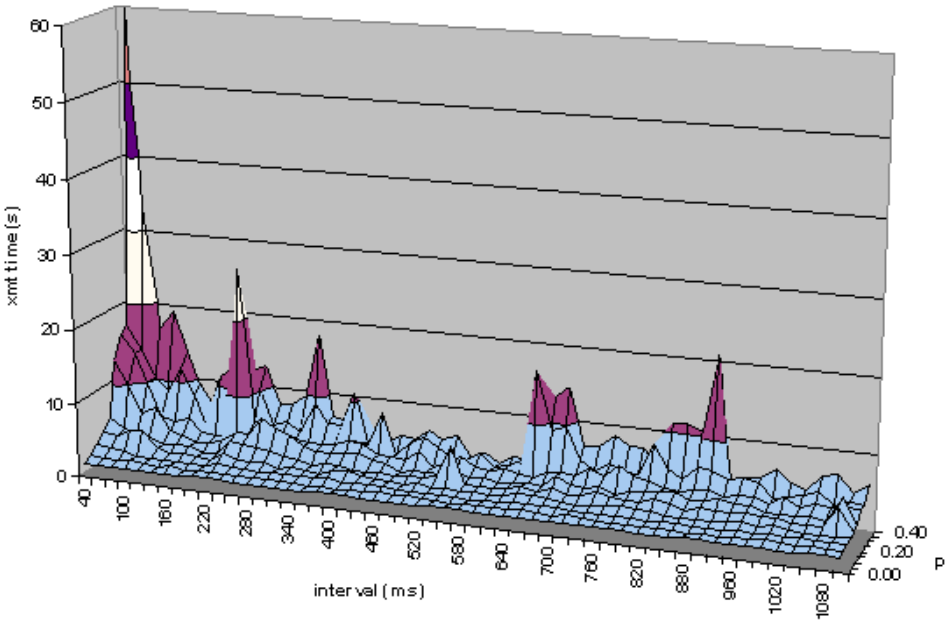

Figure 4.8: Average observed time of file transmission plotted for link wink probability and interval length using Hybla congestion control algorithm.

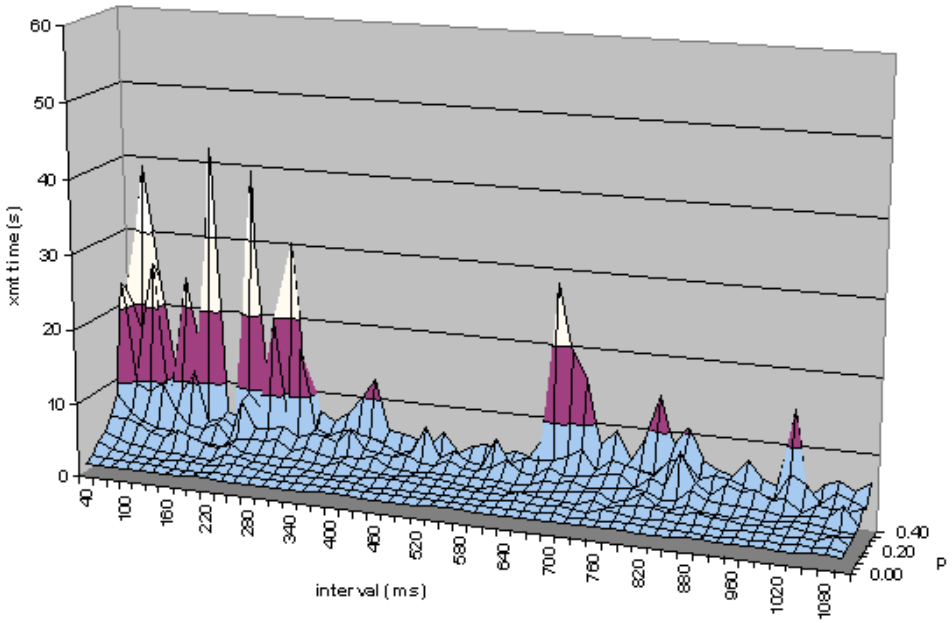

Figure 4.9: Average observed time of file transmission plotted for link wink probability and interval length using Reno congestion control algorithm.

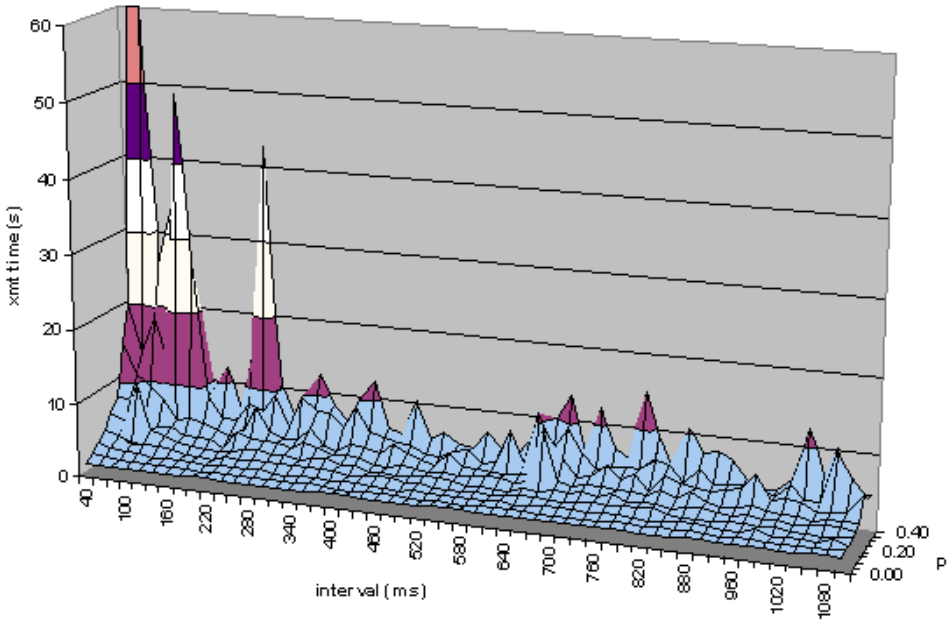

Figure 4.10: Average observed time of file transmission plotted for link wink probability and interval length using Scalable congestion control algorithm.

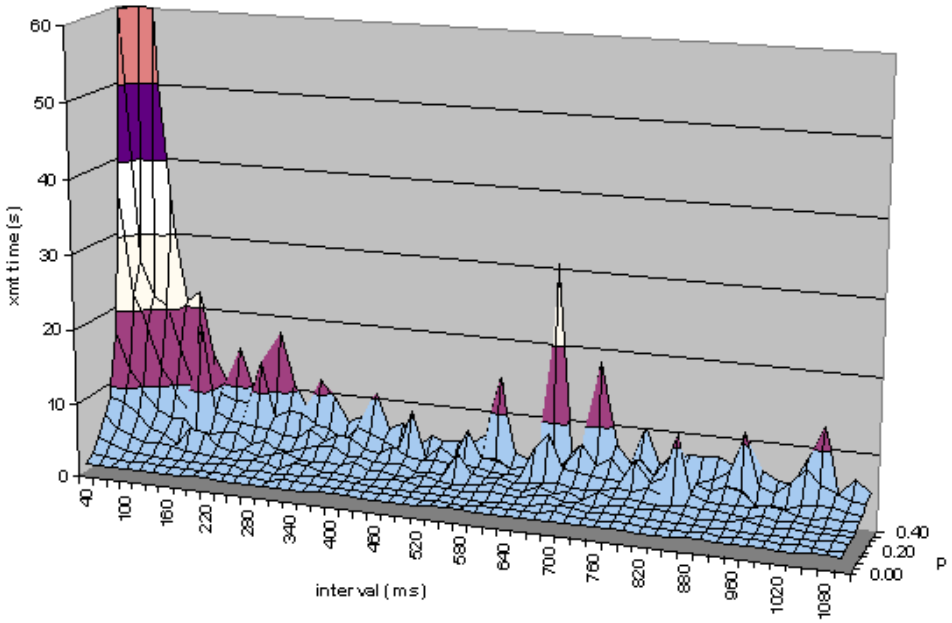

Figure 4.11: Average observed time of file transmission plotted for link wink probability and interval length using Unfair congestion control algorithm.

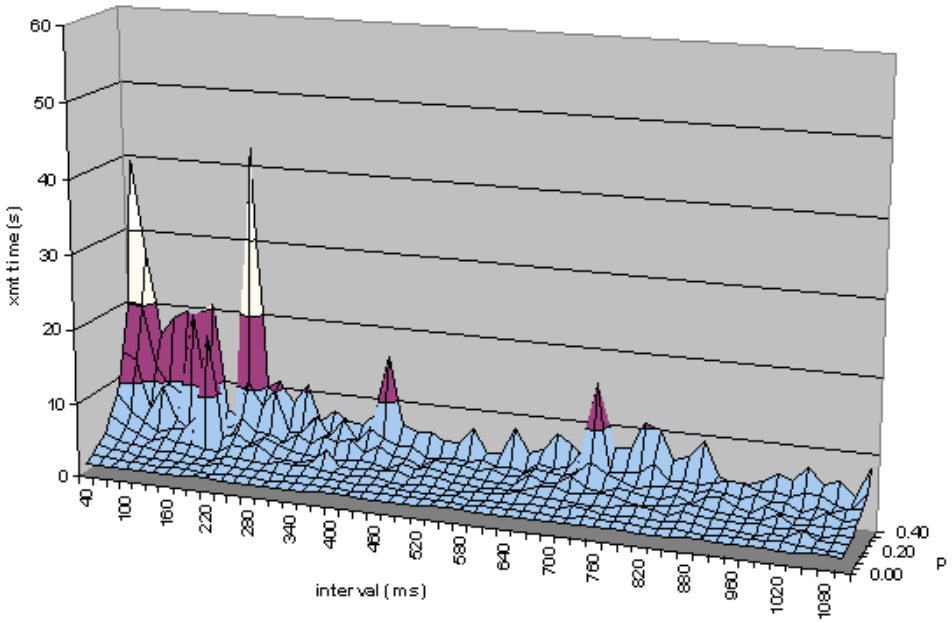

Figure 4.12: Average observed time of file transmission plotted for link wink probability and interval length using Vegas congestion control algorithm.

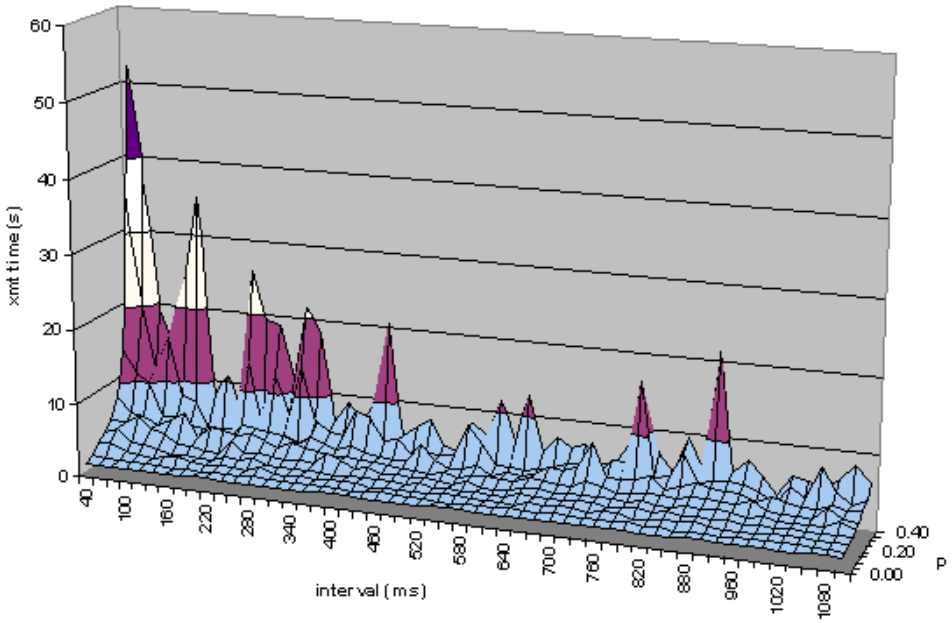

Figure 4.13: Average observed time of file transmission plotted for link wink probability and interval length using Westwood congestion control algorithm.

Qualitatively investigating the frequency of send times shows smallest time greater than 1.86 is near 2.1 seconds, 200ms more. The next lowest times center around 2.25 seconds, 500ms more. This is discussed in Chapter 3 and is incorporated into the model. The following graphs are the complete data demonstrating this part of the model. The model adjustments were based on some preliminary data collected during prototyping.

The shortest transmit times seen for the lower intervals for $i > 40$ and $i \leq 200$ is 2.1 seconds (see Figure 4.14. This demonstrates that the minimal transmission time with at least on retransmission is 200ms. For $200 < i \leq 500$, the smallest transmission times is at 2.5 seconds; Figure 4.15. For $i > 500$, the smallest significant time is 3.3 seconds; Figure 4.16.

The model consistently underestimates the values where p is low, like 0.05. It also underestimates the regions where p is high, like 0.40, and i is low, such as 40. Further refining the model is left for future consideration. It appears all of the costs are not included or are underestimated by some of the assumptions made in Chapter Three.

In particular, as intervals fail and are successfully retransmitted, the moment the retransmission starts is not likely to be exactly at the beginning of an interval. On average it starts half way through the interval. Since only part of the interval is available for retransmission additional interval are required for every successful retransmission.

The graph of the model does appear to model the graphs generated by the testbed data for the various congestion control algorithms. The graph also depicts a couple low spots near intervals 200ms and 500ms. These low spots are represented in the testbed graphs as well however they are not exactly at 200ms and 500ms. The peaks are generally represented also. Certain as the intervals get smaller the value increases.

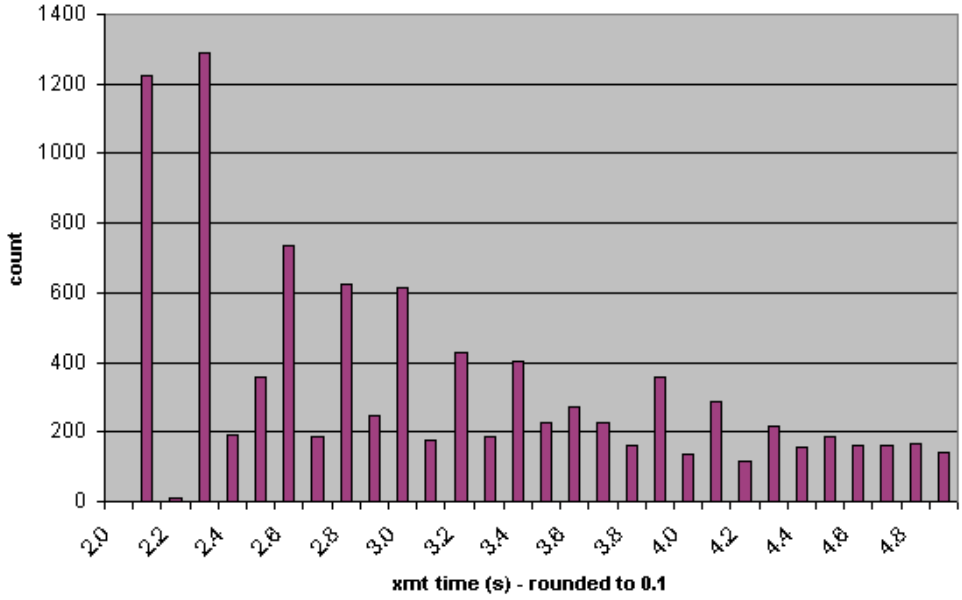

Figure 4.14: Counts of observed transmission times with interval lengths less than or equal to 200ms.

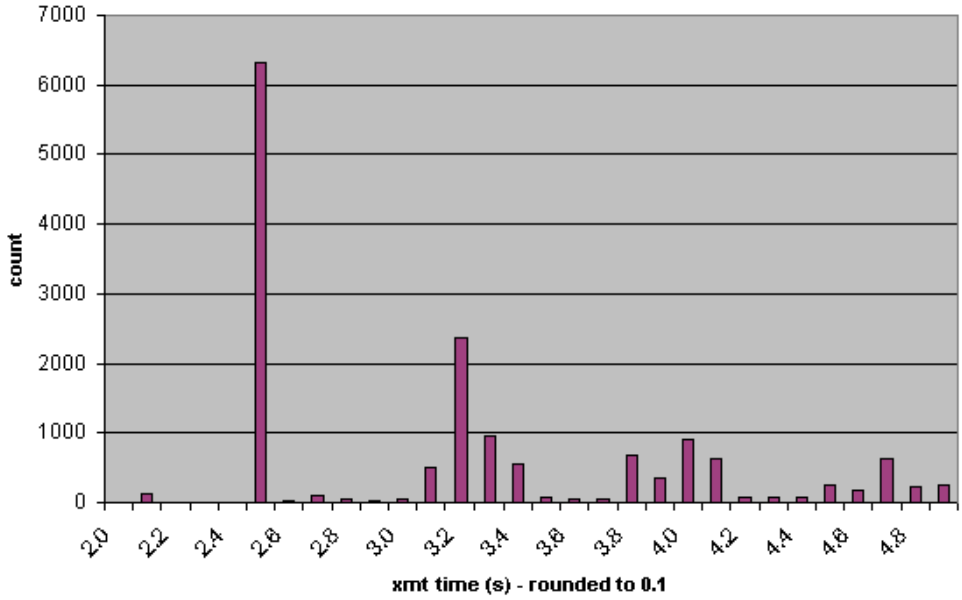

Figure 4.15: Counts of observed transmission times with interval lengths greater than 200ms and less than or equal to 500ms.

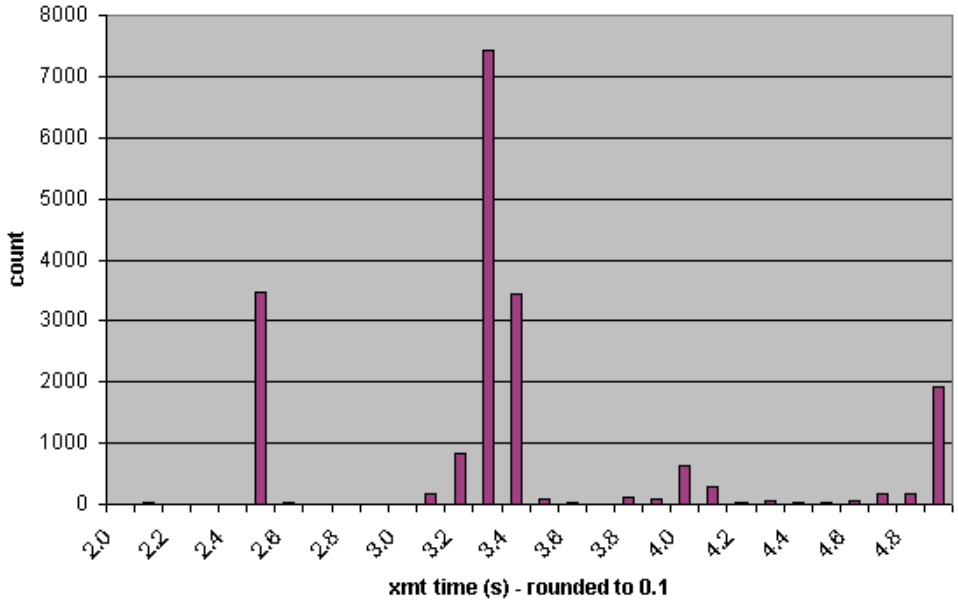

Figure 4.16: Counts of observed transmission times with interval lengths greater than 1100ms.

While the shapes of the graphs are generally the same, the values of the model are lower than the actual data. The values in the low i, and high p region are an order of magnitude short. The high i region appears close. The assumption that the number of intervals is strictly tied to the interval length appears to be weak. The number of intervals required to send the file needs to be tied to the probability too. For example,

$$n = \frac{t_{perfect}}{i} \cdot \frac{1}{p} \tag{4.1}$$

or

$$n = \frac{t_{perfect}}{i} \cdot \frac{1}{p^2} \tag{4.2}$$

would increase the number significantly. This was attempted but an overflow occurred. Instead, using

$$n = \frac{t_{perfect}}{i} \cdot (1 + p) \tag{4.3}$$

yielded the following graph. Further model refinement is left for future investigation.

The variances witnessed in the data grew substantially as the probability increased. Accurately modeling the variance would further demonstrate the accuracy of the model. This type of model is a *mixture distribution*. According to *Statistical Inference* [6], the expected value and the expected variance are shown here where X is the binomial distribution and Y is the geometric distribution.

$$E[X] = E[E[X|Y]] \tag{4.4}$$

$$VarX = E[Var(X|Y)] + Var(E[X|Y]) \tag{4.5}$$

The complex nature of these calculations are out of the scope of this thesis. It is noted for future investigation.

4.7 Results with Custodial Buffering

A second set of experiments was run to understand the implications of strategic buffering and retransmission in a challenged TCP stream. Due to the time constraints of this thesis only 3 of the 9 congestion control algorithms are considered. The probability and intervals tested remain the same. At each of these points 30 trials are run.

The three algorithms investigated are Reno, BIC, and Unfair. Reno was selected because it is the closest to the original TCP. BIC was selected because it is the default congestion control algorithm used in Linux at the time of this thesis. Unfair was selected because it was crafted with the intention of keeping the buffer as full as possible by never adjusting the congestion window.

As described in Chapter Three, the retransmission scheme employed in the buffering node is not efficient. With little to no link wink it was expected that the insertion of full time custodial retransmission would be worse than the regular

TCP. However as the link wink gets worse, it is anticipated the new mechanism will outperform TCP.

Using the described scheme, the file takes around 4 seconds to send without any wink: BIC 4.7s, Reno 4.6s, and Unfair 3.6s. This is expected to be greater than 1.86s due to the inefficient retransmission scheme in the buffering node. Two points are to be made here. One, as better retransmission scheme reduces this number. Two, when there is no wink, or little wink, employing custody is not optimal. Knowing when to turn it on and off is critical to optimization. Both of these points are left to further investigation.

In figures 4.17, 4.18, and 4.19, several points are of interest. Unfair appears to standout from the other two. Unfair appears to look as to match the model presented in Chapter Three, see Figure 4.20. All three increase as the probability increases. BIC and Reno peak at 200ms. All three appear to do the same at the lowest interval and the highest probability.

Unfair continues to keep the buffer full enough that as it retransmits, it does not run out of packets. In other words, the buffer appears to have fresh packets to send when it needs them. The model assumes the buffers are occupied and the empirical data for Unfair appears to reflect this. Unfair outperformed the other two algorithms at all points. In the shorter intervals, Unfair with custody also beat Unfair without custody (Figure 4.11). From about 300ms and up, it outperformed the non-custodial case by approximately 50%. At 620ms, the non-custodial was better than the custodial case 3.7 to 5.8. Although the points where the non-custodial times beat the custodial times are curiously interesting, optimizing the retransmission scheme may change this. More testing is required here. There is significance in the lowest interval (40ms) and the highest probability (0.40) point where the improvement went from 280ms to 7ms. This was accomplished using a sub-optimal retransmission scheme.

At the shortest interval and highest probability, both Reno and BIC show significant improvement. It took less than 10ms to send compared to 217ms for BIC and

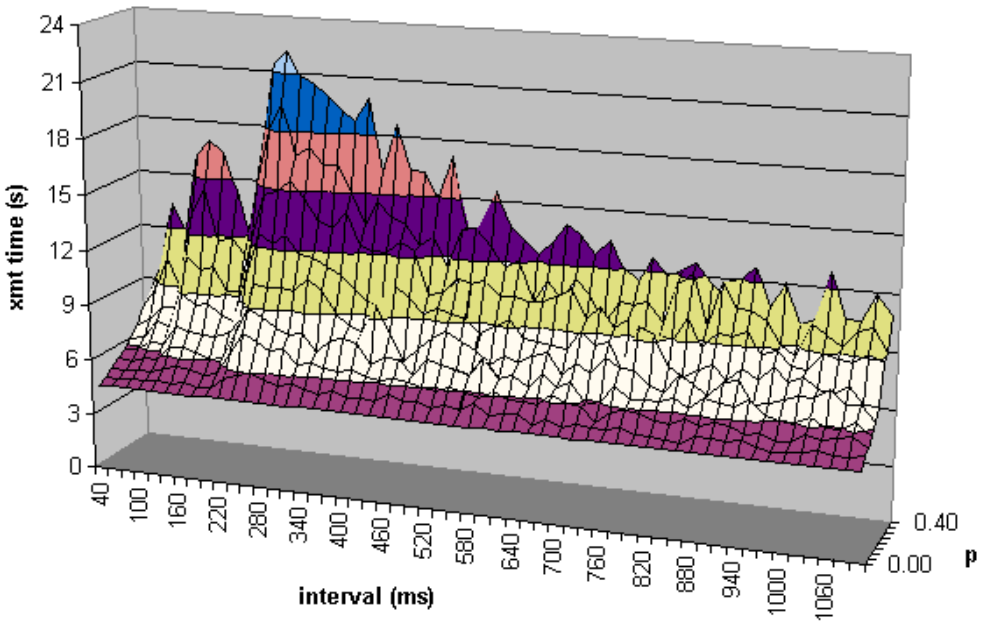

Figure 4.17: BIC Mean Transmission Time with custody.

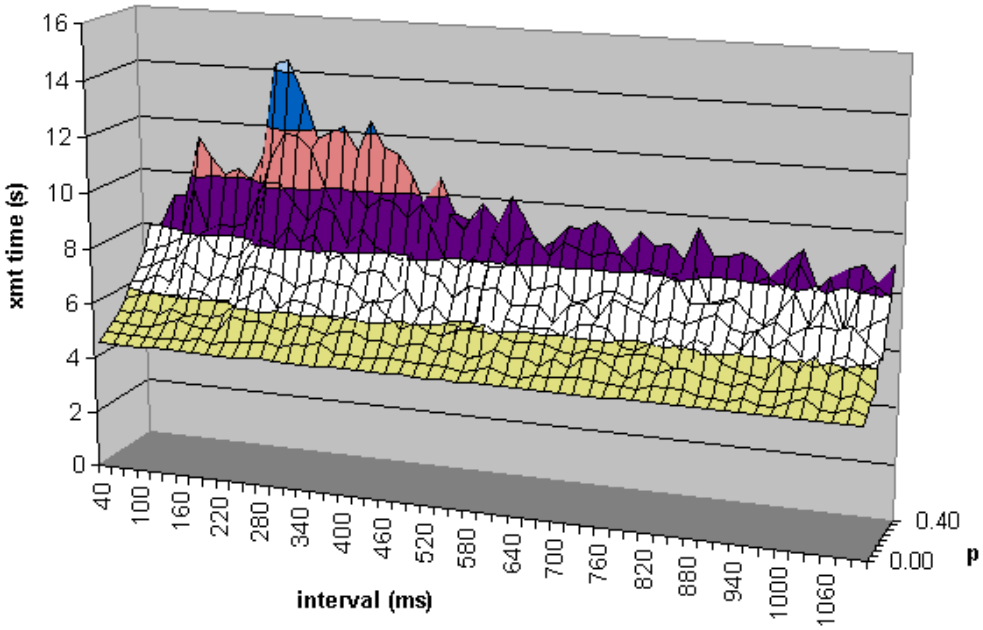

Figure 4.18: Reno Mean Transmission Time with custody.

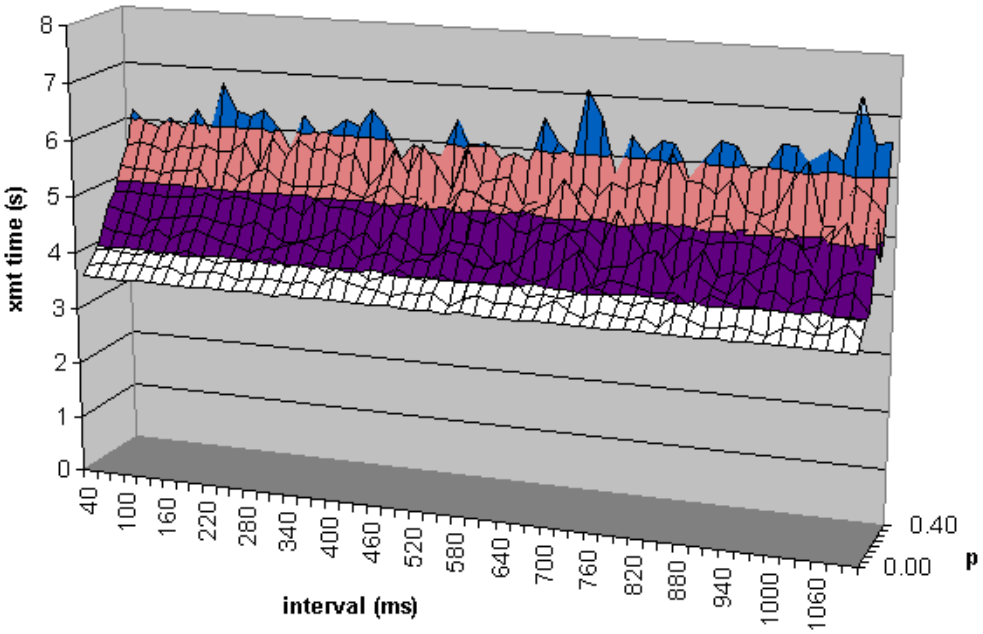

Figure 4.19: Unfair Mean Transmission Time with custody.

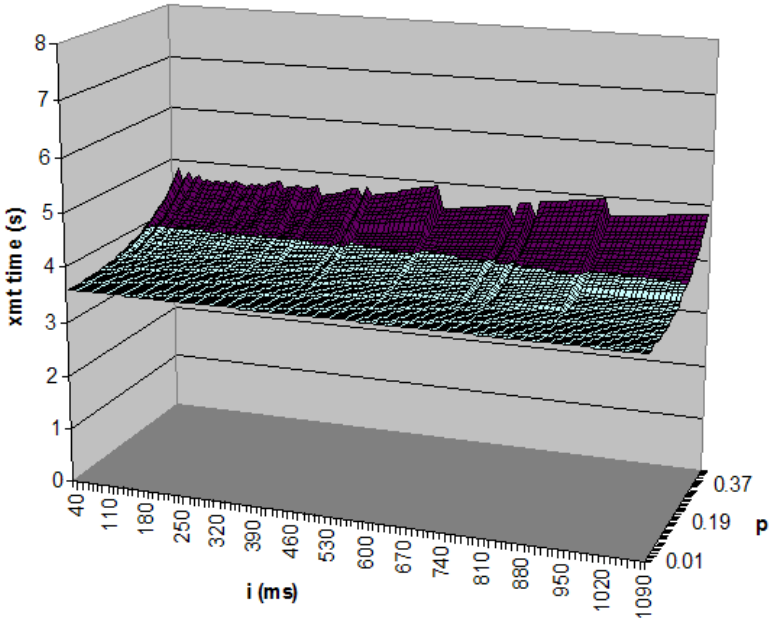

Figure 4.20: Predicted Unfair Transmission Time with custody. Mathematical model delay plus baseline send of 3.6 seconds.

27ms for Reno. BIC and Reno in the 40ms interval follow Unfair and the above analytical model. Compared to BIC, the buffer must have a steady flow of fresh packets. Both perform the worst with custody at 260ms. Two possibilities are contemplated to cause this. One, the congestion window is too small to keep the buffer loaded with packets. Or two, the 200ms timer in TCP is somehow being retransmitting the same packets over and over. Recording and inspecting the specific traffic flows needs to be investigated to answer this question. At the high probability and from 260ms and up, BIC and Reno both perform worse, nearly double, than they do without custody, see Figure 4.5 and 4.9 respectively.

4.8 Performance Improvements

In the scenario without custody, the transmission times spike in the area of high probability and short intervals for all three congestion control algorithms. Consider the worst case scenarios of p=0.40. In Figure 4.21, the dashed line represents BIC without custodial buffering. Notice at the shortest interval (i=40) spikes up and out of the graph up to 193 seconds. The graph is truncated for scaling purposes. In Figure 4.22, the dashed line represents Reno without custodial buffering. While it does not spike as high as BIC, it does reach near 40 seconds to send the 20MB file at some of the shorter intervals. It is noted even though 30 trials per data point were run, the variances grew rapidly as the probability p increased up to 0.40.

In each graph, the thin, solid line represents how each respective algorithm performed employing full-time custodial buffering just before the winking link. Notice the significant decrease in transmission times where the interval length of the wink is small ($i < 240$). Also noticed at ($i > 240$), the custodial buffering performed worse than without it. Some of this poor performance is blamed on the simple retransmission scheme. It also demonstrates certain conditions call for custodial buffering and retransmission and others do not. However, the most significant contributing factor to the poor performance is that the buffering node's buffer did not have fresh packets to retransmit.

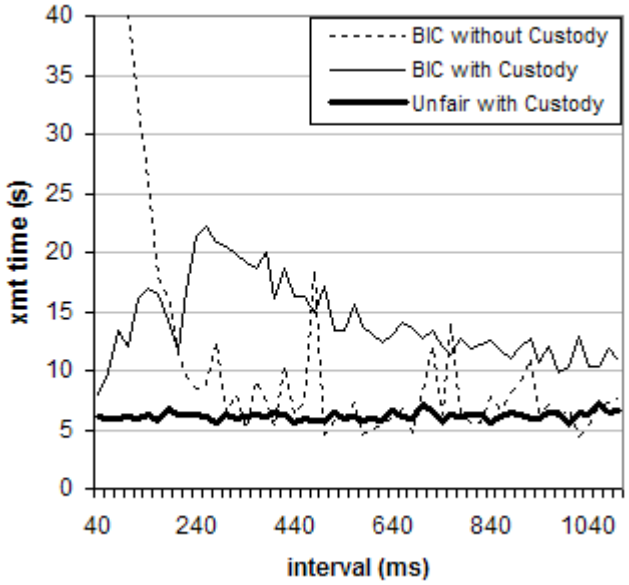

Figure 4.21: BIC with and without custody only for *p=0.4*. Note at *i=40*, the value is 193.

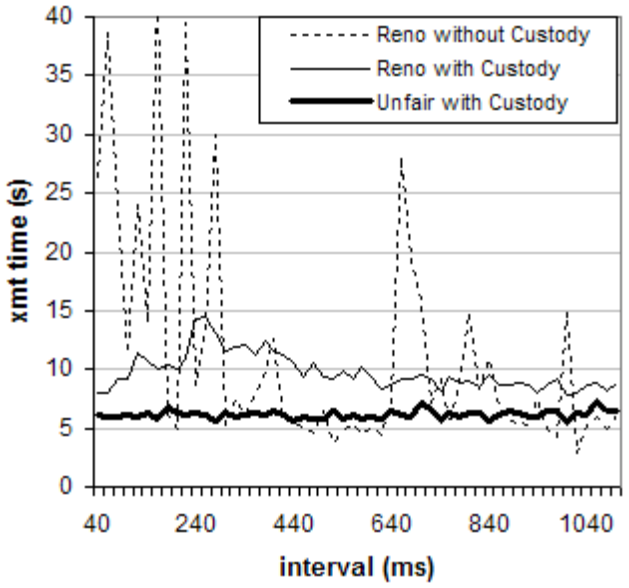

Figure 4.22: Reno with and without custody only for *p=0.4*.

In each graph, the bold, solid line represents the performance of the Unfair algorithm employing full-time custodial buffering. In nearly all cases, this combination outperforms the others. This combination appears consistently transmit around 6 seconds across all intervals. Because Unfair does not shrink its congestion window in response to the lack of acknowledgements, it does not retreat to sending a small amount of data. This keeps the buffering node's buffer full enough of fresh packets. For optimization, a node taking custody must communicate to the source node to increase its transmission window instead of shrinking it.

4.9 Summary

This chapter has covered the data collected both with and without strategic custodial buffering. The differences and similarities of the congestion control algorithms' behavior in the presence of link wink. The mathematical model qualitatively demonstrates the behavior of TCP's retransmission scheme in the presence of link wink. Using this model, the behavior of strategic custodial buffering is predicted. The results using the Unfair congestion control algorithm closely match the predictions. The next chapter uses these results to highlight the conclusions and future research.

V. Conclusions and Future Research

5.1 Overview

This chapter concludes and discusses the implications of this thesis. The conclusions focus on the benefits of custodial buffering and retransmission and the usefulness of smart, resourceful routers. Following the conclusions, future research items are discussed.

5.2 Conclusions

5.2.1 Performance. This thesis demonstrates that where TCP performs poorly under certain challenged conditions which we anticipate to be common in scenarios of interest, the use of custodial buffering and retransmission improves TCP significantly while retaining desirable properties of reliable delivery and flow control. The results from Chapter Four show that when the route of a TCP flow has a high probability (0.4 or greater) of breaking for short intervals (40ms), TCP takes 20-100 times longer to complete transmissions. Most of this time is spent idle as TCP's congestion control timer increases exponentially between failed retransmissions. This thesis demonstrates that these complications can be overcome by strategically inserting a proactive agent in the route. By copying the flow's traffic into a buffer, the agent is capable of intense retransmission the moment a forward link experiences trouble. This strategy only required 3-4 times longer to transmit the file at the worst of conditions: short, intense winking. This is a significant improvement from 20-100 times longer. In theory this method is capable of pushing transmissions through troubled routes with failures much greater than 50%.

5.2.2 Delay Tolerant Traffic. Not only does retransmission provide improvements, but the simple buffering of traffic allows for expanded possibilities. Delay tolerant traffic is traffic that does not have to arrive immediately. This traffic can be buffered in the middle of the route and held for some period of time before it expires. Examples of traffic that need a guarantee of arrival but does not have to arrive quickly

include email, the sharing of large datasets, and system backups. The type of delay is defined by the use of the data. Data gathered either from remote sensors or network nodes is useful data but does not require instantaneous delivery.

Currently, data with a tolerance for delay is routinely dropped from intermediate routers whenever the traffic demand exceeds transmission capability. The assumption is that if it can withstand delay then let the source retransmit the traffic. This thesis challenges this assumption by suggesting this traffic should be buffered in intermediate nodes, if possible, instead of being dropped. The network may have already incurred great cost in bringing the data this far, and will have to do so again if it is dropped. If there is storage available to the node, then storing the data may prove far more efficient. Acknowledgments can be returned to the source, preventing unnecessary retransmissions and informing the source of the progress of the data and perhaps anticipation of future progress

An important issue to examine is the condition that requires this delay tolerant traffic be dropped. The only reason for dropping any traffic is congestion, i.e., insufficient transport capability for the current load. If a prioritization scheme is in place which places delay sensitive traffic at higher priority than delay tolerant traffic, then delay tolerant (lower priority) traffic is dropped before delay sensitive (high priority) traffic. The logical, detrimental conclusion to this is that when the high priority traffic is decongested or gone, there is little to no low priority traffic available to send. From the perspective of the delay tolerant traffic, the hope is that the source is lucky enough to retransmit during periods of little high priority traffic. Because of the delay associated with re-transmitting the data back to the point at which it was lost, much potential bandwidth is wasted, i.e., if the node at which the data was lost had an unused period, this bandwidth could be used.

The custodial buffering alleviates this problem. This thesis asserts that (1) data that can't be sent and (2) data that fails to be sent, are semantically equivalent. The data is not successfully transmitted. The model of link wink presented in Chapter

Three and tested in Chapter Four is semantically equivalent to situations (1) and (2). Whether a packet is dropped due to congestion, consumed by a lossy channel, or simply held indefinitely due to its priority level, it is semantically identical from the perspective of the sender and receiver. Thus, the availability of the channel to a flow of traffic is semantically general. The transmission time of a flow does not care whether packets are evaporating into the ether, rotting in the bit bucket, or stuck sideways in the network card. Considering the link wink to be the presence of high priority traffic, this thesis demonstrates that having low priority traffic available to for transmission immediately during periods of little high priority traffic significantly improves the transmission time of the low priority traffic.

5.2.3 Valleys. Further implications for the experiments in the thesis are found in some "sweet spots" in the TCP timers. In all of the congestion control algorithms investigated, certain winking intervals demonstrated that the transmissions did not suffer as much as expected, even at high probabilities of winking. Specifically intervals in the range of 200-250ms and 500-600ms demonstrated at a high probability (0.4) that the average time to send was only 2-3 times as long. The other intervals were consistently higher. The frequency of these interesting winking intervals coincides with frequency of TCP's retransmission timers. This harmonic alignment represents the moments when TCP got "lucky" and transmitted when the wink ended.

This has a couple of useful implications. If the environment is such that the winks of the link are exist and are controllable, the timing and duration can be adjusted such that their frequency coincides with these "sweet spots" in TCP's timers. With custodial buffering, the flows' transmission times might be optimized in a challenged environment, keeping just barely enough low priority traffic available in intermediate nodes to capitalize on periods of low high priority traffic, or equivalently periods of high available bandwidth. It is contemplated that with proper calculations this could be used in the scheduling algorithms used in transmitting flows from vari-

ous priority queues. If a router is using a round robin algorithm to pull from various queues, the length of time and how often each queue is visited can be optimized to minimize all of the flows transmission times and/or the size of buffers at intermediate nodes. As a counter intuitive example, consider that a flow may need to be transmitted less often in order to get better throughput. In certain conditions, the data suggests that sending it every 200ms is more productive than sending it every 100ms and orders of magnitude better than sending it every 40ms. While these results are highly sensitive to the models and assumptions used in these experiments, the presence of such "sweet spots" illustrates the possibility for considerable optimization.

5.2.4 Smart Routers. The most general implication of this thesis is that enabling routers with more resources and more sophisticated functionality is beneficial in the scenarios outlined. The driving force in router design today is strictly minimizing the delay experienced by traffic passing through the router. Additionally if the router is too busy the traffic is dropped. It is suggested that overall performance can be increased if some processing time and storage capability is dedicated to handling the traffic smarter instead of quicker. These additional resources such as storage and processing units must keep the common case fast. If there is no congestion, no priority starvation, or no troubled data links then minimizing delay is the number one factor.

Smart routers need to consider the common case outside of the common case. Smart routers can increase performance if provided with the means and the information to make and implement good decisions. A smart router needs a large amount of storage for buffering the data flows. The smart router needs co-processing to manage these buffers. It also needs some global and some local knowledge. The local knowledge comes from analyzing all of the traffic passing through the router. Analysis of the flows occurs at the Transport Layer. The local knowledge also comes from exchanging this higher level knowledge with its neighbors. This information includes forward route conditions such as buffer capacities and throughput performance. Globally, the network at large can provide useful information such as hints or explicit knowledge of

large scale network conditions such as downstream issues or alternate routes. Global instructions can include changes in priorities or specific commands to decrease or to increase transmission rates to coordinate several smart routers to maximize performance.

5.3 Future Research

In the course of this thesis, several assumptions were made. Several factors such as round trip time, number of nodes, and file size were held constant. Many of these factors need to be understood more. Other points of interest useful for future investigation include: retransmission strategy, when to take custody, richer congestion control algorithms, and the mathematical models. Each area is discussed below.

5.3.1 Retransmission Strategy. The retransmission strategy presented in this thesis is rigid and unsophisticated. It simply sends a fixed amount of packets periodically. The entire buffer (up to a static pre-defined maximum amount) is transmitted every retransmission cycle. Further investigation into how often to send packets to overcome a winking link should increase the performance. This strategy involves many aspects including but not limited to bandwidth, volume of other flows, frequency and duration of the wink and downstream congestion. Understanding the conflict between fairness (between flows or between nodes in a shared medium) and the need to overcome the winking of a particular is very important.

The size of the retransmission is related to the frequency of retransmission. Many parameters are available to influence this calculation: size of the buffer, defined capacity of the link, perceived capacity of the link, rate of data arrival, other flow usage of the link, other buffer capacities, flow priority, receiver's suggested window size, link behavior, round trip time, and explicit external notifications. Each of these parameters must be monitored and computed somehow, per flow, in order to assist in optimizing the amount and frequency of retransmission over a given link.

5.3.2 Custody. Under good conditions, the overhead of buffering and retransmission degrades performance. Therefore, we believe that under good conditions custody of the flow is best avoided. As conditions worsen a crossover point exists at which taking custody of the flow increases performance. Determining the conditions under which to take custody requires further research. TCP has limited information about the status of the links along the route. The lower and closer data link layer is in a much better position to provide information about the status of the link. The inter-layer solution requires further research. Additionally global network monitors may be able to compile information useful in determining when to take custody. A "network weatherman" could be constructed which would create and maintain knowledge of the current status and the forecast such that this information can be provided to the router. Custody could then be directed ahead of the "storm". Similarly, determining when to halt custody is an open concern. Because custody hurts performance under good conditions, knowing when to stop is just as critical to optimization. It may also be possible to reduce the buffering overhead to point at which custody is always preferred, negating the need for these decisions.

A route may have multiple nodes taking custody over multiple links. Understanding the performance and semantic implications of using more than one strategic buffer in a route is important. Furthermore determining how many hops away from a wink a buffering node can be would provide information on determining how many buffering nodes are needed in a network to provide "complete" coverage.

In the short term, establishing nodes on both sides of the wink, working together, to overcome the wink in both directions is important. Each node can absorb the repeated transmissions (and acknowledgments from the other) to overcome the bad link and then forward the traffic away from the link more fairly. This situation is better than what is presented in this thesis. If both nodes are attempting to overcome the wink, they must work together so as to not blast packets at each other.

A bad link also presents difficulty in exchanging not only data, but handshaking packets and negotiating packets, such as RIP. Perhaps a back channel is available for communication which can accommodate control traffic. This back channel could be a separate form of communication or could be a separate route through the network. If the alternative channel exists, consideration must be given as to whether the channel could be used effectively. For example a slower more reliable channel may be more efficient than a fast unreliable one.

Of serious concern is the potential for loss of a buffering node. If a node takes custody and notifies the source and proceeds to accumulate a generous, if not entire, portion of the flow in its buffer and then is lost or destroyed, the source will not retransmit the message. The intermediate node has committed itself to getting the data to the destination. The source is not to send the data again because an agent has assumed responsibility. Some mechanism at the source or the stranded node must be developed to detect the loss of an intermediate buffer and determine what actions to take.

This is also a serious security concern. As described in Chapter Three, security needs to be addressed. The cornerstones of information security: confidentiality, integrity, and availability are all ripe for research in strategic buffering.

One final custodial area open for pursuit is a leader election protocol. Along a route with more than one strategic buffer node, more than one node may independently take custody. To scale properly each node should keep state for nodes further than one hop away. The various nodes need to communicate in some fashion to negotiate which is to take control. During the prototyping of the testbed, some discussion looked into a solution using the TCP data packets themselves. The leader algorithm should pick the buffering node furthest from source and closest to the bad link. The TCP packet's TTL field provides a monotonically decreasing value that might be useful in determining where along the path a particular node is. More research is need here.

5.3.3 Congestion Control. As presented in this thesis, differences exist between the congestion control algorithms in the face of winking links; whether custody is taken or not. The algorithm Highspeed suffered significantly compared to the others where there was no custody. The algorithm created for this thesis, Unfair, outperforms the others with custody; a natural result given that Highspeed was optimized for a very different situation and Unfair was somewhat optimized for this situation. Research into creating a congestion control algorithm that performs well under both conditions is needed.

If custodial retransmission is in effect, the source should continue to feed packets to the node so that the buffer always a fresh set of packets to send when the opportunity presents itself. In the IACK packet, described in Chapter Three, some information can be provided to the source. Specifically the window size field in the packet could be used to request more packets or fewer packets depending on the conditions in the buffering node.

During the development of this testbed, it was contemplated that SACKs could replace the use of IACKs. Originally defined in RFCs 1072, 2018, and 2883, Selective Acknowledgments (SACKs) are altered acknowledgment packets intended to increase the performance of TCP. They are implemented as TCP options stored in the TCP header. The data included in the header defines ranges of sequence numbers that the destination has received. This allows for gaps in sequence numbers to be communicated to the source. The source is then able to send only the missing packets and not the entire window.

More importantly, the sequence numbers indicated as having been received by the destination are not truly acknowledged as if a regular ACK had been sent. These packets are marked as SACKed in the source and not retransmitted. Furthermore, they are discounted from the congestion window, i.e. the window slides. This is semantically the same as the IACK described in Chapter Three, but may allow today's implementations of TCP (that support SACK) to be used with out modification. This

presents an area for future consideration. One reason SACKs were not initially used is that under certain conditions the packet marked as SACK'd is reset to not SACK'd. Further trouble occurred in considering a chain of nodes having custody of a flow. Certain combinations of SACK ranges create a loss of packets between the custodial nodes. Due to the time constraint of this thesis the straightforward IACK approach was implemented. Again, using SACKs offers a means of implementing strategic buffering without changing TCP in the source node. This helps to honor the end-to-end approach. Our strategic buffering and retransmitting could be implemented without actively changing the end points and may work with existing congestion control algorithms.

5.3.4 Mathematical Model. Although topographically similar, the mathematical model presented in Chapter Four consistently underestimates the values seen in the testbed. The probability functions accurately reflect much of the TCP retransmission mechanism. The cost and number of retransmissions appear to be underestimated. The real question seems to be: how many packets are really sent in a successful interval? This thesis assumed the entire interval was full of successful packets. Some of the intervals will have less than a full interval of packets. Additionally from these under populated intervals, additional intervals of packets are needed to make up the shortfall.

The minimum number of intervals to send a file was based on the perfect send time of the file divided by the length of the interval. It is asserted that the real number of successful intervals needed to send the file under imperfect conditions is more than under perfect conditions. What this formula should be is an area of investigation. It is likely tied to the probability of the link winking.

This thesis stopped with intervals as small as 40ms. This amount can approach 0 up to the point that the interval is as short as a packet. How small can the interval be and the model still work? At some point it is contemplated that the link winking

interval will be so small that the current TCP retransmission scheme will perform better. It is designed to handle the intermittent loss of a few packets.

5.4 Summary

In final conclusion, this thesis demonstrates ways to overcome some of TCP's over reaction to lost ACKs in challenged environments. This was accomplished by inserting retransmission into the path between the source and the destination. This research has created several other angles for future research ranging from mathematical analysis to protocol implementation to network performance.

Mobile networks an intricate part of Network Centric Operations. We are confident that strategic buffering will play an important part in optimizing network utilization of future mobile networks. In addition to utilization, TCP transmissions that would otherwise be impossible to send, can be transmitted in harsh environments such as the battlefield and disaster areas. The demands of Network Centric Operations places on its infrastructure will grow. As the work in the thesis is refined more of those demands will be met.

Bibliography

1. Bakshi, Bikram S., P. Krishna, N. H. Vaidya, and D. K. Pradhan. "Improving Performance of TCP over Wireless Networks". *17th IEEE International Conference on Distributed Computing Systems (ICDCS '97)*, 00:365, 1997. ISSN 1063-6927.

2. Balakrishnan, H., S. Seshan, and R. H. Katz. "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks". *ACM Wireless Networks*, 1(4), December 1995.

3. Beck, M., T. Moore, and J. Plank. "An End-to-End Approach to Globally Scalable Network Storage". *In Proceedings of the ACM SIGCOMM 2002 Conference*. ACM, Pittsburgh, PA, August 2002.

4. Brakmo, Lawrence S. and Larry L. Peterson. "Tcp vegas: End to end congestion avoidance on a global internet". *IEEE Journal on Selected Areas in Communications*, 13(8):14651480, October 1995. Vegas.

5. Caini, Carlo and Rosario Firrincieli. "International Journal of Satellite Communications and Networking". *IEEE Journal on Selected Areas in Communications*, 22(5):547566, August 2004. Hybla.

6. Casella, George and Roger Berger. *Statistical Inference*. Duxbury Press, Pacific Grove, CA, second edition, 2002.

7. Draves, R., J. Padhye, and B. Zill. "Routing in multi-radio, multi-hop wireless mesh networks". *In Proceedings of the 10th Annual international Conference on Mobile Computing and Networking*, 114–128. ACM, ACM Press, New York, NY, Philadelphia, PA, October 2004.

8. Fall, Kevin. *A Delay-Tolerant Network Architecture for Challenged Internets*. Technical Report IRB-TR-03-003, Intel Research, Berkeley, February 2003.

9. Floyd, Sally. *Highspeed tcp for large congestion windows*. Technical report, RFC 3649, 2002. Highspeed.

10. Hanbali, Ahmad Al., Eitan Altman, and Philippe Nain. *A Survey of TCP over Mobile Ad Hoc Networks*. Technical Report 5182, INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE, May 2004.

11. Jacobson, Van. "Congestion avoidance and control". *ACM SIGCOMM 88*, 314329. ACM, Stanford, CA, August 1988. Reno.

12. Jain, S., Kevin Fall, and Rabin Patra. "Routing in a Delay Tolerant Network". *Proceedings of the ACM SIGCOMM 2004 Conference*. ACM, Portland, OR, August 2004.

13. Karrer, R., A. Sabharwal, and E. Knightly. "Enabling large-scale wireless broadband: the case for TAPs". *SIGCOMM Comput. Commun*, 34(2):27–32, January 2004.

14. Karygiannis, A. and E. Antonakakis. "mLab: A Mobile Ad Hoc Network Test Bed". *1st Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing in conjunction with the IEEE International Conference in Pervasive Services 2005*. Santorini Island, Greece, July 2005.

15. Kelly, Tom. "Scalable TCP: improving performance in highspeed wide area networks". *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, 2003. ISSN 0146-4833. Scalable.

16. Kim, D., Chai-Keong Toh, and Yanghee Choi. "TCP-Bus: Improving TCP Performance in Wireless Ad-Hoc Networks". *ICC (3)*, 1707–1713. 2000.

17. Kohler, E., Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. "The click modular router". *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000. ISSN 0734-2071. URL `http://www.pdos.lcs.mit.edu/click/`.

18. Kopparty, S., S. Krishnamurthy, M. Faloutsos, and S. Tripathi. "Split-TCP for Mobile Ad Hoc Networks". *Proceedings of IEEE GLOBECOM*. Taipei, Taiwan, November 2002.

19. Kramer, W.T.C., A. Shoshani, D.A. Agarwal, B.R. Draney, G. Jin, G.F. Butler, and J.A. Hules. "Deep scientific computing requires deep data". *IBM Journal of Research and Development*, 48(2):209–232, March 2004.

20. Lee, Y., Kwong-Sak Leung, and Mahadev Satyanarayanan. "Operation-based Update Propagation in a Mobile File System". *Proceedings of the USENIX Annual Technical Conference*. Monterey, CA, June 1999.

21. Leith, D.J., R.N. Shorten, and Y.Li. *H-tcp: Tcp for highspeed and long-distance networks*. Technical report, Hamilton Institute, National University of Ireland, Maynooth, August 2005. Htcp.

22. Mascolo, S., Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. "Tcp westwood: Bandwidth estimation for enhanced transport over wireless links". *MobiCom 01: Proceedings of the 7th annual international conference on Mobile computing and networking*, 287297. ACM, ACM Press, Rome, Italy, July 2001. Westwood.

23. McDonald, Ian and Richard Nelson. "Congestion control advancements in Linux". *linux.conf.au 2006*. Dunedin, New Zealand, January 2006.

24. Nordstrom, E., Per Gunningberg, and Henrik Lundgren. "A Testbed and Methodology for Experimental Evaluation of Wireless Mobile Ad hoc Networks". *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05)*, 100–109. Trento, Italy, February 2005.

25. Rajamani, Rajesh and Gogul Balakrishnan. *Efficient Large-scale data movement on the Grid - Augmenting the Kangaroo approach*. Project, University of Wisconsin, Madison, Wisconsin, 2001.

26. Robinson, J., K. Papagiannaki, C. Diot, X. Guo, and L. Krishnamurthy. "Experimenting with a Multi-Radio Mesh Networking Testbed". *Workshop on Wireless Network Measurements (WiNMee)*. Trentino, Italy, April 2005.

27. Satyanarayanan, M. "Mobile Information Access". *IEEE Personal Communications*, 3(1):26–33, 1996.

28. Satyanarayanan, M., James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. "Coda: A Highly Available File System for a Distributed Workstation Environment". *IEEE Transactions on Computers*, 39(4):447–459, 1990.

29. Thain, D., Jim Basney, Se-Chang Son, and Miron Livny. "The Kangaroo Approach to Data Movement on the Grid". *High Performance Distributed Computing, 2001, Proceedings of Tenth IEEE International Symposium on High Performance Distributed Computing, HPDC-10*, 0325. San Francisco, CA, August 2001.

30. Wright, Gary R. and W. Richard Stevens. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley Professional, New York, NY, first edition, December 1993.

31. Xu, L., Khaled Harfoush, and Injong Rhee. "Binary increase congestion control (bic) for fast long-distance networks". *IEEE Infocom 2004*, 735–740. Institute of Electrical and Electronics Engineers (IEEE), Hong Kong, May 2004. Bic.

32. Yang, G., Ren Wang, Mario Gerla, and M. Y. Sanadidi. "TCPW with Bulk Repeat in Next Generation Wireless Networks". *Proceedings, IEEE ICC 2003*. Anchorage, AK, May 2003.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
13–06–2006	Master's Thesis	Sept 2004 — Jun 2006

4. TITLE AND SUBTITLE

Mitigating TCP degradation over intermittent link failures using intermediate buffers

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

M. Brent Reynolds
Civilian ND-04
USNAVY
michael.reynolds@navy.mil

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

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

Air Force Institute of Technology
Graduate School of Engineering and Management
AFIT/EN Bldg 641
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GIA/ENG/06-09

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

Dr. Robert Herklotz (ph: 703-696-6565)
AFOSR/AFRL
Suite 325, Room 3112
875 Randolph Street
Arlington, VA 22203-1768

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This thesis addresses the improvement of data transmission performance in a challenged network. It is well known that the popular Transmission Control Protocol degrades in environments where one or more of the links along the route is intermittently available. To avoid this degradation, this thesis proposes placing at least one node along the path of transmission to buffer and retransmit as needed to overcome the intermittent link. In the four-node, three-link testbed under particular conditions, file transmission time was reduced 20 fold in the case of an intermittent second link when the second node strategically buffers for retransmission opportunity.

15. SUBJECT TERMS

TCP, buffering, buffers, retransmission, link wink, lossy link, congestion control, delay tolerance, testbed

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Scott Graham
U	U	U	UU	105	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636, ext 4918

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18