TCP Performance over Multilink Long Delay Wireless Networks

by

Said Ismail Zaghloul

Bachelor of Science, Electrical Engineering
University of Jordan, Amman, Jordan, 2002

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Thesis Committee

Chairperson: Dr. Victor S. Frost

Dr. Gary Minden

Dr. David Petr

Date Accepted: _____

To my beloved parents and family for their continuous support

To my distinguished supervisor for his invaluable guidance

Acknowledgements

It is impossible to acknowledge all people who had a major influence on the conception and the fruition of this work. To the best of my ability, I shall attempt to do so. I would like to extend my sincere gratitude to Dr. Victor Frost for his remarkable supervision and for his invaluable comments that made this effort a reality. I would like to thank Dr. Gary Minden and Dr. David Petr who agreed to be on my committee. Special thanks for Dr. Shanmugan who provided me with solid understanding of digital communications.

I would also like to thank Abdul Jabbar Mohammad who was a marvelous partner who spared no effort in helping me solve intriguing problems. Furthermore, I would like to thank the ITTC system administrators for their prompt help and support, especially, Brett Becker and Mike Hulet. In addition, I would like to thank my friends Adam Bittlingmayer and Timo Kip for being wonderful roommates.

Moreover, I would like to thank the Fulbright foundation and the University of Kansas for providing me with the wonderful experience for continuing my Master of Science degree.

In closing, I am extremely grateful to my family for their continuous love and moral support.

Abstract

TCP is the standard transport protocol for many applications. In some cases, in order to satisfy application requirements, it becomes necessary to inverse-multiplex low bandwidth wireless links (using multilink point-to-point protocol MLPPP) to achieve higher bandwidth. The use of such technologies is required to provide adequate Internet access to support field research in remote regions (e.g. Greenland and Antarctica) that are only covered by low bandwidth systems (e.g. Iridium). Utilizing MLPPP is also a possibility for establishing connectivity over multiple cellular channels. The use of these technologies posed new challenges (e.g. call drops) that have not been fully analyzed yet. The TCP latency models developed in the last decade address a variety of factors that affect TCP throughput such as the effect of timeouts and the effect of packet drops due to wireless errors. This thesis focuses on TCP over MLPPP, specifically, the evaluation of the effect of call drops on the TCP performance. In order to provide insight into the nature of the call drops process, an estimate of the probability density function (pdf) of the time difference between call drops is determined experimentally. Using this model, the development in [15] is extended to account for call drops. Then, the proposed model is experimentally validated by field measurements using the Iridium network. Next, the effects of the loss probability, the timeout interval, and the ARQ operation on the TCP performance are evaluated. Finally, the design of the developed Iridium link management software is discussed to provide a complete image of the Iridium system's operation.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1. Introduction

There has been an enormous growth in the Internet with a tremendous increase in the demand for connectivity and high bandwidth. This necessitated the development of efficient transport protocols. TCP [1], transport control protocol, is by far the most common transport protocol in the Internet. Consequently, researchers concentrated their efforts in optimizing the current TCP versions in order to meet the users' expectations. As a result, the topic of predicting the TCP transfer time has received considerable attention. Models for predicting TCP performance are pivotal to the development of enhanced versions of TCP as they provide better understanding of the effect of various network parameters, such as the channel error probability and round trip time variations.

Researchers [15] started by modeling long transfers, such as FTP, over wired networks in terms of packet loss probability (due to congestion), round trip time (RTT), and the maximum window size (W_{max}). Such models were based on many simplifications related to the specific case of long file transfers. Then, researchers [23] investigated short transfer times such as web-browsing. This case is more complicated than the long transfer case and leads to more intricate models. Researchers [18] managed to develop complex models that capture several aspects of TCP such as slow start and the acknowledgement timer expiration effect. The models developed rely on numerical solutions to obtain performance results.

As Internet connectivity became a fundamental aspect of modern societies especially with the advent of wireless networking, researchers [27] studied the performance of TCP in wireless channels. Since wireless channels have higher error rates, designers include reliability mechanisms in the data link layer to mitigate packet losses which severely impair the TCP performance. The major challenge in this case is that losses might be due to wireless errors or due to congestion. Due to the complexity of this problem, researchers have only developed long transfer latency models for wireless links.

The connectivity in remote regions such as in Greenland and Antarctica is a challenging problem. Any solution has to satisfy the primary user requirements: reliable connectivity and adequate bandwidth. Satellite connections are the most attractive solution for such a problem. Polar Regions, however, are among the most difficult spots to cover by satellites because of the high latitude in these regions. The Iridium satellite network is the only satellite network that provides full geographic and temporal coverage for Polar Regions. Although the Iridium satellites provide coverage, the offered (per channel) data rate is very low (2.4 kbps) [7]. It was shown in [4] and [14] that adequate bandwidth (19 kbps) may be achieved by inverse multiplexing multiple (eight) Iridium links, using the Multilink Point-to-Point Protocol (MLPPP), into a single virtual link. Thus, the bandwidth of the virtual MLPPP link equals the total bandwidth of all the physical links.

1.1 Research Goals

The main objective of this research is to develop and experimentally validate a TCP long transfer model that takes call drops into account. A call drop event means that one of the MLPPP channels loses its connection suddenly. There are various reasons that might cause call drops to occur such as low signal-to-noise ratio (SNR), handover failure, network signaling failures, etc. It has been observed in [4] that a call drop results in a TCP timeout. The reason why a call drop leads to a timeout is not obvious. It is certain, however, that a call drop leads to losing sufficient number of packets (or their acknowledgements) resulting in a timeout. Thus, a probabilistic model for call drops was developed based on experimental measurements from the Iridium satellite network. The call drop model is then used to develop a mathematical model that predicts TCP performance at variable call drop rates.

In addition, this research aims at providing qualitative description of the multi-channel Iridium system performance by studying the effects of the packet error rate and the TCP timeout values. Moreover, the relationship between the communication channel fading conditions (slow or fast) and the packet error rate is discussed briefly. Finally, efficient link management software was developed in order to attain optimal operation and collect system operation measurements. The link management software enabled the development of the proposed TCP performance model.

1.2 Accomplishments

This work resulted in the following accomplishments:

1. A TCP performance prediction model based on call drops and link loss statistics has been developed and experimentally validated.
2. A satellite system emulator has been developed in order to test the proposed model under call drop rates higher than the normal Iridium call drop rates.
3. An estimate of the call drops probability density function (pdf) has been developed.
4. Efficient link management software has been developed with a user-friendly graphical user interface (GUI) that requires minimal configuration.
5. TCP performance over other satellite systems (i.e. Inmarsat) has been studied in terms of RTT and throughput.

1.3 Thesis Organization

In this thesis, a brief introduction to the Iridium satellite network architecture is given in Chapter 2. Then, the MLPPP protocol architecture and operation is summarized. Afterwards, the main aspects of the TCP latency models for long and short transfers are surveyed with focus on long transfer models. Next, a thorough discussion of the call drop process is given showing complete details of modeling call drops over a MLPPP bundle. Then, the TCP performance prediction model which considers the call-drop effect for TCP connections running MLPPP is derived and experimentally

validated using the Iridium system. Afterwards, a selection of system design parameters (round-trip-time and timeout value) is discussed. Finally, the development of the link's management software is described in detail.

Chapter 2. Introduction to Satellite Data Networks

2.1 Introduction

The communications industry witnessed enormous growth in the last decade. With the development of the Internet many applications became practically feasible. Connectivity and mobility are almost becoming the two main features of modern communication systems. There are many different methods that attempt to achieve those goals at a minimum cost and at the best quality, such as third generation cellular systems, fiber optic based networks, and satellite systems. Perhaps the most unique feature of a satellite system is its ability to cover wide areas of the Earth's surface [1]. This unique feature makes it possible for satellite systems to cover large areas of the globe at a relatively low cost. This makes satellite systems an attractive solution for establishing connectivity to remote points and distant locations where it is practically impossible to have other means of communications. Examples are desserts, oceans, and Polar Regions.

Satellite systems fall into three main categories depending on the selection of their orbits and their orbital dynamics: Geostationary Earth Orbit (GEO), Medium Earth Orbit (MEO) and Low Earth Orbit (LEO) as shown in Table 2.1.

Table 2.1: Comparison of Different Satellite Systems [1]

	LEO	MEO	GEO
Satellite cost	Maximum	Minimum	Medium
Satellite life (years)	7-9	10-15	10-15
Hand-held terminal	Possible	Possible	Very Difficult
Propagation delay	Short	Medium	Large
Propagation loss	Low	Medium	High
Network complexity	Complex	Medium	Simple
Hand-off	Very	Medium	No
Development period	Long	Short	Long
Visibility of satellite	Short	Medium	Always

2.2 Current Satellite Systems

This section briefly surveys the current data communications satellite systems. There are several GEO satellite systems currently available. For example, the Inmarsat network consists of four satellites located at an altitude of 35800 km along the equator covering most parts of the globe. Inmarsat is planning to launch a fourth generation of satellites in 2005 [3] that will support a new Broadband Global Area Network (BGAN) which promises to deliver Internet and intranet content and solutions: video-on-demand, videoconferencing, and LAN access at speeds up to

432kbit/s in most parts of Africa and Asia, including many parts of Russia, China, Indonesia and Australia [3]. BGAN will also be compatible with the third-generation (3G) cellular systems. The new system is expected to provide at least 10 times as much communications capacity as today's Inmarsat network. Other similar systems, Intelsat and PanAmSat, consist of a few (2-8) large satellites. Each satellite covers up to one third of the Earth's surface and has high transponder capacity to be shared by users in its footprint. They are easier to maintain, have a long life, and are very reliable.

Although GEO systems have many advantages, they have several limitations that make them unsuitable to use for some applications. Due to the high altitude of the GEO satellites, the two-way propagation delay (uplink/downlink) ranges from 240 ms to about 270 ms depending on the elevation angle. This does not take into account any processing or queuing delays, or even retransmissions (for IP traffic). The other effect of the large distance is the large propagation loss. It is well known that the distance has a square relationship with power, which makes battery life and the size of mobile devices a non-trivial issue. Being geosynchronous with one point above the equator makes the satellite partially visible or even invisible at high latitude points, such as the Polar Regions, since high latitude locations require a very low elevation angle. For example, Inmarsat has partial coverage in Greenland while most systems are not accessible beyond 70 degrees N or S latitude [4].

LEOs have been designed to tackle the problems of GEO satellites. Examples of LEOs include ORBCOMM, Iridium, Globalstar, ICO, and Teledesic. The orbital period of these satellites is much lower than that of the Earth (between 90 and 120 minutes). They are visible for a very short time from any given point on earth. Since, the footprint size is proportional to the height of the satellite, the number of satellites needed in LEO systems is much higher than that in GEO. A thorough discussion of the derivation of the approximate number of satellites can be found in [2]. These LEO satellites are located at a relatively low distance (700-1000 km) from Earth [4]. Each satellite covers a small area on the Earth and provides connectivity for approximately 10-30 minutes. Examples of LEO networks are Iridium, Globalstar and Odyssey.

It is clear that LEO systems are a constellation of satellites that constitute a complete network. The way to establish the networking function varies from one LEO network into another. For example, the networking function could be implemented by simply completing the radio link between the ground stations through the satellite network. Thus, the satellite network only passes the signals to the next ground station. The more complex approach is to have "a homogenous autonomous system" [5] that consist of a constellation of satellites that communicate switching information between them using inter-satellite links (ISL). The switching is done completely in space by a smart switching algorithm. This is the fundamental difference in satellite network design approaches between the ground-based Globalstar satellite networks and the interconnected Iridium satellites. Figure 2.1 shows the network layer

architecture of the Globalstar and the Iridium network. It is clear that complex switching systems are harder to upgrade or modify as they implement satellite onboard switching, whereas simple ground (bent-pipe) systems are much simpler to modify and upgrade. In the next section, a more detailed discussion of the Iridium network will be investigated.

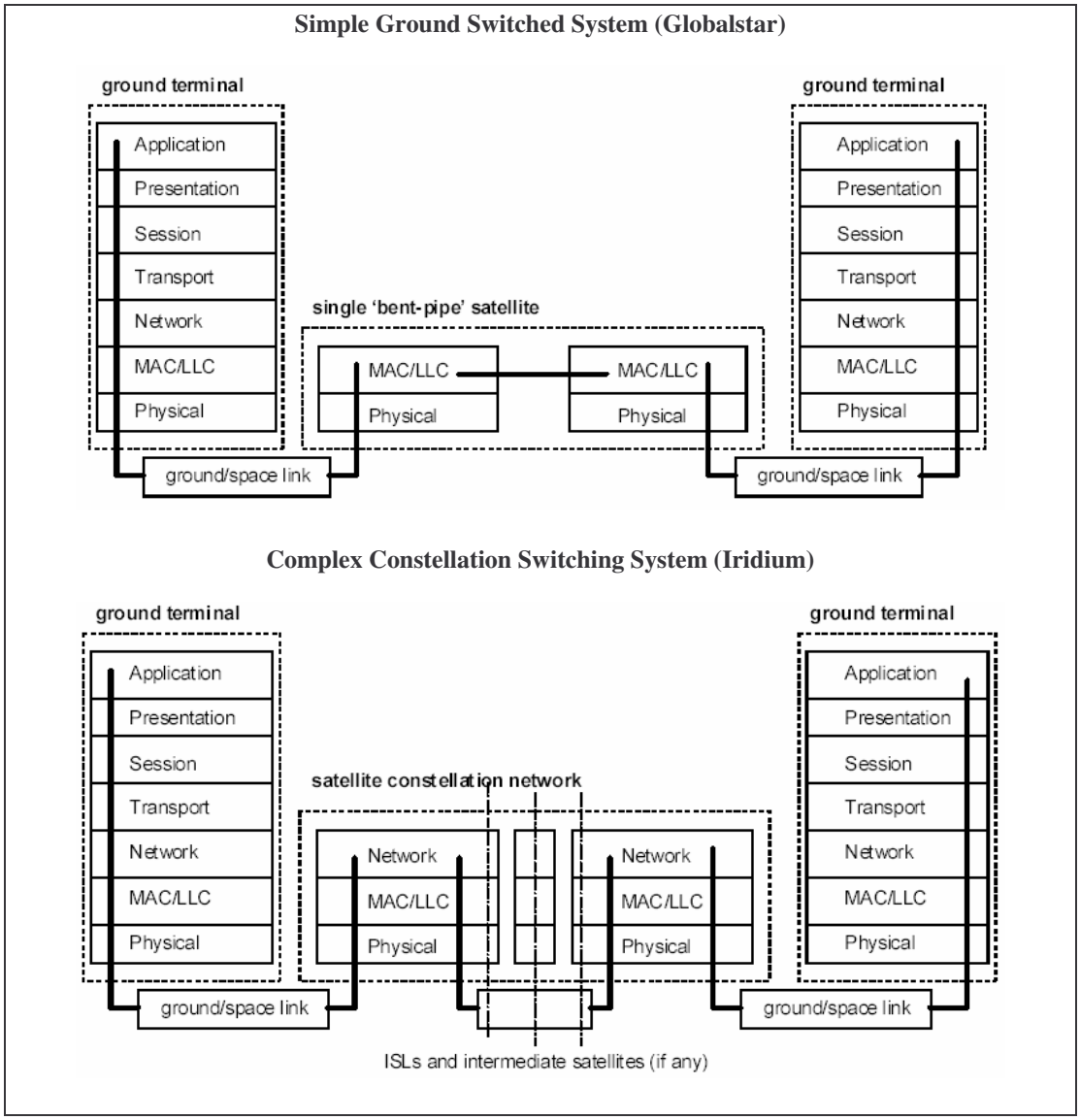

Figure 2.1: Networking Function in LEO Satellite Networks [adapted from [5]]

2.3 Overview of the Iridium Satellite Network

Motorola's original design of the Iridium system consisted of 77 satellites [6]. The name Iridium was suggested by Motorola's cellular systems engineer, Jim Williams, after the Iridium element which has 77 electrons. The design was modified later to consist of 66 satellites only. The Iridium satellite constellation was launched in May, 1998. It is currently maintained by Boeing [7]. The 66 satellites are organized in 6 planes with 11 satellites in each plane. The planes are circular around the Earth with inclinations of 86.4 degrees. The altitude of the satellites is 780 km with each satellite weighing 689 Kg. The expected lifetime of each satellite ranges from 7 to 9 years.

As discussed in Section 2.2, Iridium uses a complex system of inter-satellite links (ISLs). The ISLs operate in the frequency range of 23.18 to 23.38 GHz. The actual details of the ISL operation are not published. Nevertheless, it is known that most satellites have four links: two inter-planner (east and west) and two intra-planner links (north and south) [8]. Each satellite has an orbital period of approximately 100 minutes.

Unfortunately, the complete details of the Iridium physical layer design are proprietary. Some sources ([2], [4] and [6]), however, state that the Iridium network uses the frequencies from 1610 to 1626.5 MHz for both uplink and downlink. The 10.5 MHz bandwidth is divided into 240 channels. Each channel occupies 31.5 kHz

bandwidth and is separated from other adjacent channels by 41.67 kHz guard-bands [8]. Each satellite has three-phased array antennas with 16 beams per antenna. Thus, the satellite footprint is divided into 48 spots or cells. The satellite's footprint's radius of 2209 km results in a cell area on the order of 319,000 km^2/cell [4]. Thus, one Iridium satellite covers a cluster of 48 cells. The frequency reuse factor is 12 which results in 240 divided by 12 which leads to 20 frequency channels per cell. Each frequency channel is time divided into four uplink channels and four downlink channels. Thus, the multiple access technique is FDMA/TDMA. Each TDMA frame is 89.96 ms with four uplink channels and another 4 downlink channel slots as shown in Figure 2.2. Each slot occupies 8.28 ms burst time. The modulation scheme used is QPSK for both uplink and downlink. The system is assumed to have a raised cosine filter with a rolloff factor of 0.26. This leads in 50 kbps (31.5/1.26*2) data rate per frequency channel. But since each slot is 8.28 ms, taking the 3/4 coding into account, this leads to a data rate of 3.45 kbps per channel [4]. The Iridium voice coder rate is designed at 2.4 kbps. The used of the rest of the bandwidth is not published. Table 2.2 summarizes the Iridium network's design characteristics.

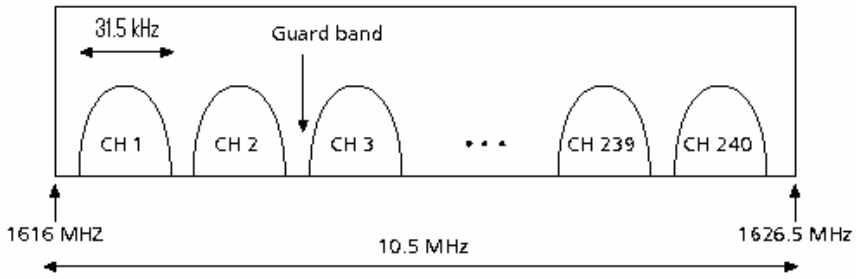

Figure 2.2 FDMA/TDMA Frame Structure
(Adapted and Modified from [8])

Table 2.2 Summary of the Iridium network characteristics

Number of Satellites	66 plus 6 in orbit backup Satellites
Orbit Height	780 km or 485 mi
Inclination of Orbital Planes	86.4
Orbital Period	100 mins
Weight	689 kg
Satellite Lifetime	7 – 9 years
Modulation Technique	QPSK
Frame Structure	FDMA/TDMA
Frequency	L-band from 1610 to 1626.5 MHz
Inter Satellite Links Frequency	Ka band from 23.18 to 23.38 GHz
Ground Segment Links	Ka band: Uplink: 29.1 – 29.3 GHz Downlink: 19.4 – 19.6 GHz
Ground-Based Digital Switches	Siemens GSM-D900
Multiple Access Technique	FDMA / TDMA
Digital Voice and Data Rate	2.4 kbps
Error Protection	FEC 3/4
TDMA Slot Length	89.96 ms
Inter-slot Guard Time	0.1 ms
Slot Time	8.28 ms
Inter-Satellite Handover	9-10 min
Minimum Elevation Angle	8.2 degrees

Iridium operates in a similar fashion to GSM. The Iridium system utilizes multiple regional gateways in order to connect to the PSTN network. Although in theory, the Iridium only requires one regional gateway, the designers of the network decided to use multiple gateways to support redundant paths and to simplify the satellite

switching. The Iridium network should always keep track of the user's location as he roams. Once, the user turns his mobile unit on, it sends a "ready-to-receive signal" to the nearest gateway through a series of ISLs [8]. Once the gateway completes authenticating the mobile unit, the mobile's location is saved in the home location register (HLR) and/or the visited location register (VLR). After finishing "the camping on the cell" procedure, the mobile is ready to make phone calls. Once a user requests a call, the signaling has to go to the HLR/VLR in order to get his location and the receiver's location (if it is Iridium mobile) and in order to complete authenticating the caller. If the receiver is not an Iridium mobile device the connection is setup is established through the gateway and then data starts to go through the gateway. However, if the connection is between two mobile users, the gateway provides the signaling that allows the data/voice packets to flow completely in the satellite network without the gateway's intervention. Figure 2.3 summarizes the call setup procedure.

Since the Iridium system is only capable of supporting a very low data rate of 2.4 kbps, many user applications may not be able to execute properly. In order to satisfy the applications requirements, multiple Iridium links may be combined into a virtual link, using the multilink point-to-point (MLPPP) protocol, with bandwidth equal to the sum of the individual links' bandwidths, as shown in Figure 2.4. The virtual link introduces new challenges to the TCP operation as the individual links

might experience call drops. This effect has to be taken into account when predicting file transfer times over such MLPPP links.

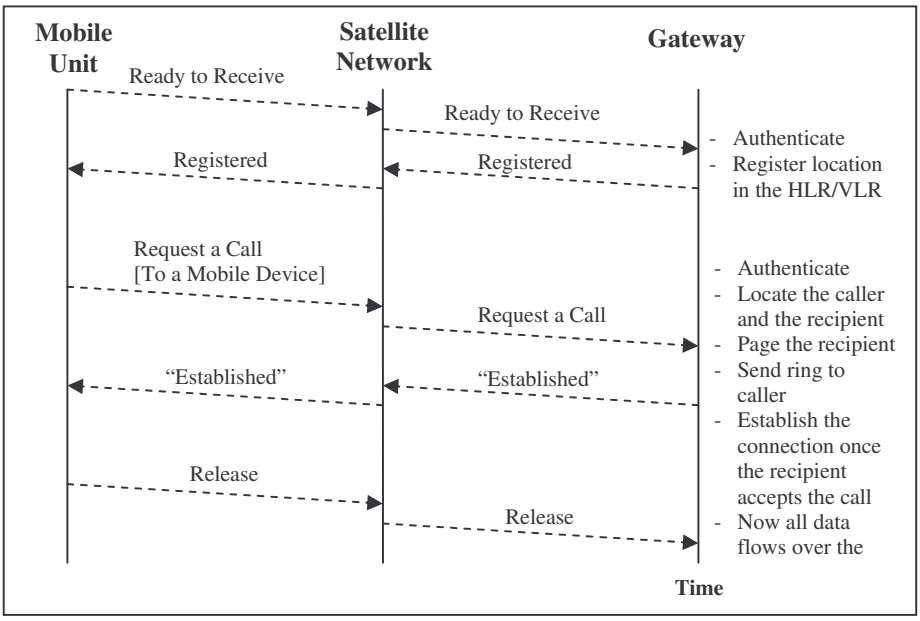

Figure 2.3: Interaction between the Mobile Unit and the Satellite Gateway

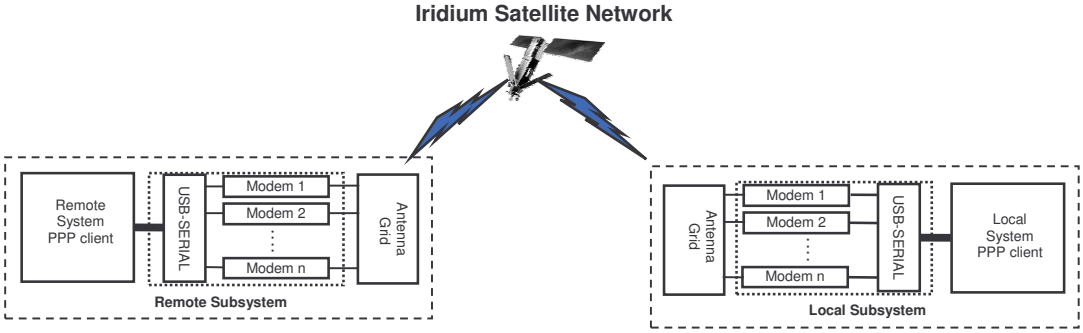

Figure 2.4: The Multilink-Iridium Communication System (Adapted and modified from [14])

Chapter 3. Overview of Multilink PPP over Iridium

Multilink Point-to-Point Protocol (MLPPP) is commonly used when the available technologies (for example at a remote location) do not meet the applications bandwidth needs. This process is referred to as inverse multiplexing [4]. MLPPP provides the end user applications with higher bandwidth by aggregating multiple low rate connections to one virtual higher speed connection. Thus, the transport layer (and hence the application layer) is completely unaware of the inverse multiplexing process as the whole "bundle" appears as a single high speed connection.

This chapter goes through the main aspects of the point-to-point (PPP) protocol, discusses the MLPPP operation briefly, and goes through MLPPP implementation details.

3.1 Introduction to the Point-to-point (PPP) Protocol

The Point-to-Point Protocol (PPP) was designed to encapsulate layer 3 protocols' data in a standardized frame format and transport it over point-to-point links such as T1, ISDN, or dial up. This is accomplished by the seamless interaction between the three main components that comprise PPP: an encapsulation mechanism, a link control protocol (LCP), and a family of network control protocols (NCPs) [9].

PPP encapsulation, shown in Figure 3.1, is designed to be carried out efficiently on most of the available hardware. This is achieved by having the hardware simply examine simple fields that fall into 32-bit field boundaries.

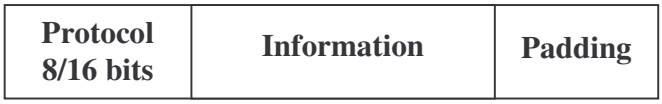

Figure 3.1: PPP encapsulation

The link control protocol (LCP) is primarily used to negotiate the encapsulation format options and packet sizes, establish the end-to-end link, terminate the link, detect failures upon their occurrence, and optionally carry out authentication. The network control protocol (NCP) is used to configure the network layer protocols when a connection has been established. NCP could be viewed as the access point between the network layer and PPP. The PPP phase diagram [9] is shown in Figure 3.2.

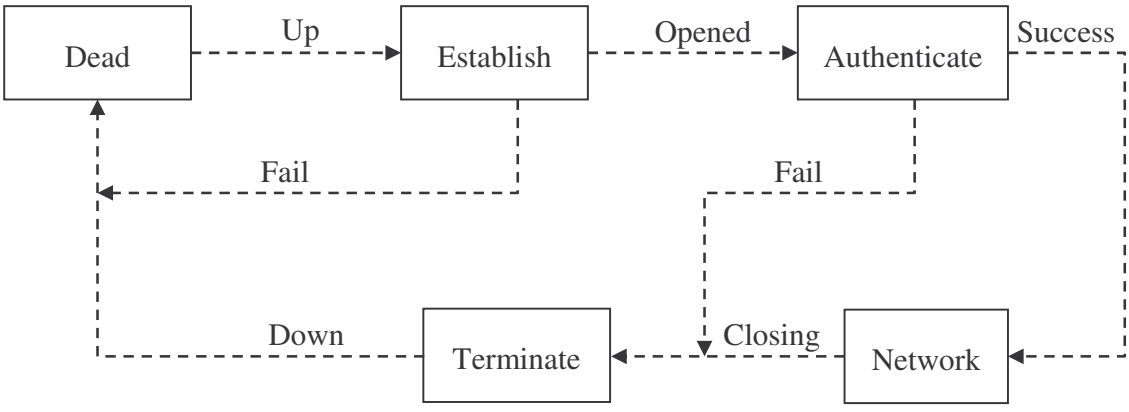

Figure 3.2: PPP Phase Diagram (adapted from [4] and [9])

It is clear from Figure 3.2 that PPP starts and ends at the “dead phase”. When a connection is ready to start, LCP proceeds to the link establishment phase where link configuration parameters are negotiated. Once the configuration parameters have been exchanged successfully, authentication (using the Password Authentication Protocol (PAP) or the Challenge Authentication Protocol (CHAP)) might take place. Then, PPP moves to the network phase where NCP configures the network layer. At this point, data packets start to flow over the link until a request to terminate or a link failure occurs. Failures are detected by means of sending periodic echo messages. In the termination phase, PPP informs the network layer protocols in order to handle the event properly. In addition, PPP performs all clean up actions needed in this stage before finally returning to the “dead phase”.

3.2 Introduction to Multilink PPP (MLPPP) Operation

“Multilink PPP (MLPPP) is a method of splitting, recombining, and sequencing datagrams across multiple logical data links [10]”. MLPPP is an example of inverse multiplexing multiple lower rate links into a higher bandwidth virtual link. The goal of the multilink operation is to coordinate multiple independent PPP links between fixed pair of systems providing a virtual PPP link with greater bandwidth than any of the constituent links.

MLPPP is designed to operate over various types of links, frames might arrive out-of-order. For example, if a MLPPP bundle is composed of a T1 line and an ISDN-B line,

then based on the intricacies of the telephone network, the delays of the two lines might vary significantly. This might result in out-of-order reception of the MLPPP protocol data units (PDUs). This problem is solved in MLPPP by including a non-decreasing 4-octet sequence number in the MLPPP packet header. MLPPP operation is illustrated in Figure 3.3.

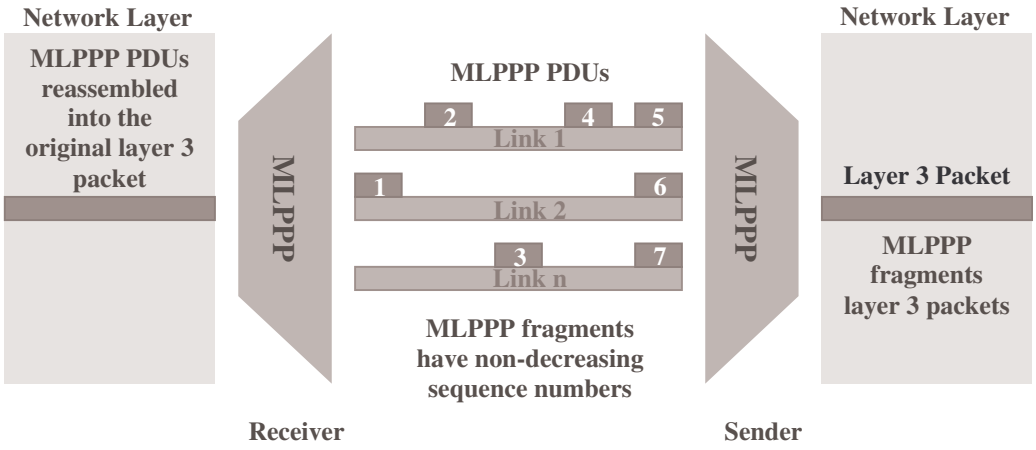

Figure 3.3: Multilink Operation

Since MLPPP is an extension to PPP, the multilink functionality is provided by negotiating a LCP option which indicates whether the other end is able to combine links into a "bundle". Multilink is negotiated by sending the multilink option in the initial LCP option negotiations phase [11]. Once the multilink option is negotiated successfully, the maximum received reconstructed unit (MMRU) is negotiated. Depending on the network layer packet size, the member links' properties, and the current load, MLPPP might fragment network layer packets into multiple PDUs to be reassembled at the receiving end.

3.3 MLPPP Implementation Overview

This section provides a brief discussion about the Linux implementations of PPP/MLPPP. Protocols that wish to support PPP/MLPPP are required to implement a generic class referred to as the "PPP channel". A channel class provides a general mechanism to transport PPP/MLPPP frames over arbitrary links [12]. Thus, the complexity of the channel structure varies depending on the underlying technology. However, the channel class only exposes simple standardized methods that allow sending and receiving PPP/MLPPP frames, and handling input/output control requests. The channel implementation is usually included in the Linux kernel. The kernel provides a generic PPP layer that uses the channel structure in order to provide a general standardized layer for programmers implementing PPP/MLPPP functionality. Thus, MLPPP implementations are completely relieved from the task of handling physical/data link layer issues. For example, currently PPP is supported over synchronous and asynchronous serial ports, and over Ethernet. Programmers dealing with PPP/MLPPP implementations use the generic layer to transmit and receive PPP/MLPPP frames regardless of the underlying hardware interface. The software implementation that uses the generic layer functionality to provide PPP/MLPPP operation to the end user is called the PPP daemon. Figure 3.4 summarizes the interactions between MLPPP generic layer and the PPP daemon (PPPD).

In short, MLPPP implementations are split into two parts: the generic PPP layer provided by the kernel and a PPP/MLPPP implementation installed optionally by the user. For example, in this thesis, Paul's PPP package (PPP-2.4.3) [13] is used in the Iridium system design. Thus, the user is mainly concerned with configuring the installed PPPD package in order to establish the MLPPP connection. Some configuration parameters are listed in Table 3.1.

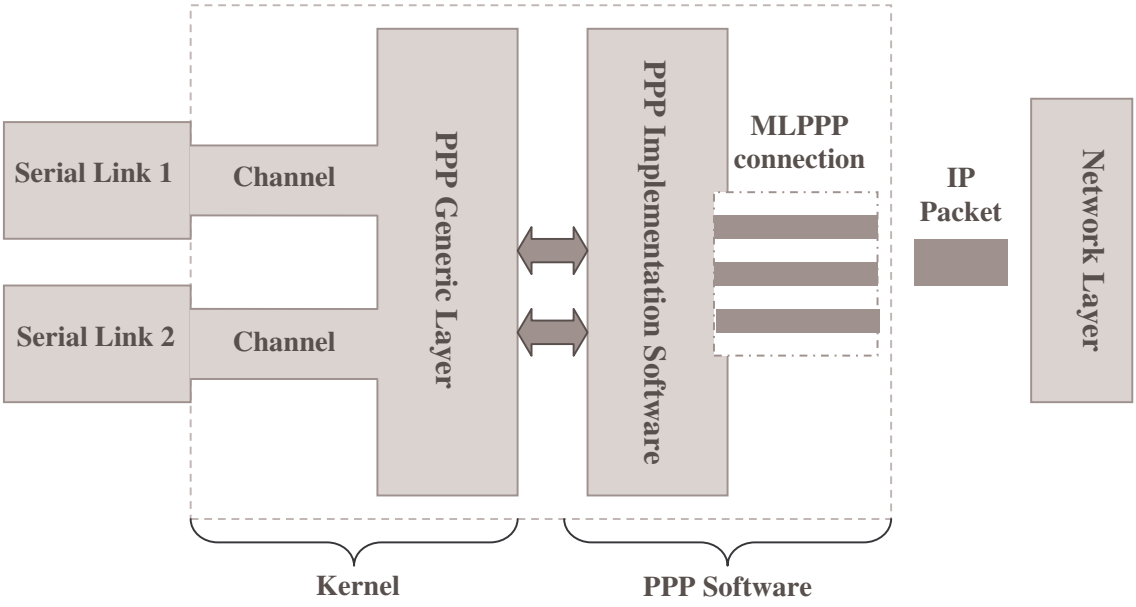

Figure 3.4: A Simplified view of PPP impelemntation operation

Table 3.1: Some PPPD configuration parameters

Parameter	Description
PAP-restart	Timeout interval before issuing another authentication request
LCP-restart	Sets the retransmission timeout interval for LCP configuration packets during the link establishment phase
LCP-max-configure	Maximum number of retrials for sending configuration packets
LCP-echo-interval	Time between LCP echo packets. Echo packets are primarily used by LCP to make sure that the link is still alive
LCP-echo-failure	Maximum number of echo packets without receiving replies. Once this parameter is exceeded a link failure is declared.
Connect-delay	Time needed to establish the physical connection, for example to dial and establish a serial connection before LCP starts sending negotiation packets.

Finally, Table 3.2 illustrates the recommended values for running MLPPP over the Iridium network [4].

Table 3.2: Recommended PPP parameters for Iridium

Parameter	Recommended values for Iridium
PAP-restart	10 – 15 sec
LCP-restart	10 - 15 sec
LCP-max-configure	10
LCP-echo-interval	30 sec
LCP-echo-failure	2
Connect-delay	5000 ms

Chapter 4. Introduction to Modeling TCP Performance

This chapter introduces three TCP versions (Tahoe, Reno and SACK) briefly. Then, some aspects of modeling TCP performance are explained. Finally, several analytical TCP models are surveyed with focus on the models for long-transfer latency which constitute the necessary background information for the proposed TCP model presented in this thesis.

4.1 Overview TCP Development

Transmission Control Protocol (TCP) is the most widely used transport protocol for Internet browsing and file transfers. The current TCP implementations are the fruit of the enormous efforts of the research accomplished by several entities: universities, research laboratories, and the Internet Engineering Task Force (IETF).

TCP is a reliable data transport protocol. The TCP sender assigns unique sequence numbers to its data units, called segments, in order to keep track of the segments' order and to ensure the delivery of all segments to the receiver. The receiver informs the transmitter about the successful reception of data by sending an acknowledgement to the transmitter. The time at which an acknowledgement is sent back to the transmitter is a protocol parameter. In fact, TCP standards (RFC1122) recommend

that the TCP source send one acknowledgement for every two segments or whenever the acknowledgement timer expires, whatever comes first.

TCP avoids overwhelming the receiver by implementing the sliding window algorithm for flow control. The receiver periodically informs the transmitter about its capacity by advertising the so-called maximum window size W_{max} back to the transmitter. The W_{max} estimate is a function of the bandwidth-delay product and the receiver's maximum buffer size.

TCP evolved through different releases. The next section goes over three versions of TCP (Tahoe, Reno and SACK) briefly. Afterwards, some main TCP procedures (delayed acknowledgements and round trip time (RTT) estimation) are discussed.

4.1.1 The Evolution of TCP Tahoe

The first deployed version of TCP known as TCP Tahoe was developed after the Internet congestion collapse in October, 1986. Van Jacobson made a major contribution to explaining the reasons behind the congestion collapse and to solving the problem by developing TCP Tahoe.

Jacobson proposed a mechanism, referred to as the slow start algorithm, which increases the congestion window size (*cwnd*) by one TCP segment, referred to as

Maximum Segment Size (MSS), for every received acknowledgement (ACK). This effect is referred to by the "multiplicative increase". The window size continues to grow up to a certain point, called the *slow start threshold (ssthresh)*, where the window size reaches a limiting value. Afterwards, i.e., when *cwnd* > *ssthresh*, the window size is linearly incremented by the minimum of $\frac{MSS^2}{cwnd}$ for every received acknowledgement or one MSS for every RTT [17]. If delayed acknowledgements are taken into consideration this results in an increase by one MSS every two RTTs. The process of incrementing the congestion window size is repeated until the maximum window size $W_{\max}$ is reached. Afterwards, the flow keeps a constant window size of $W_{\max}$.

TCP as described by Jacobson [16] is a self-clocking protocol. This means that the received acknowledgements trigger sending new packets. Thus, no new packet is sent until it is assured that one packet exited the network. Since the acknowledgements need data and data need acknowledgements to be sent, slow start initiates the first pulse for the clock and keeps increasing the data size until reaching the threshold.

According to Jacobson [16], 99% or more of the packet losses are due to congestion. Since TCP has a robust RTT estimation that prevents TCP timers from firing unreasonable timeout, then a timeout may only be due to congestion. Thus, following a timeout, it is required that a conservative mechanism (i.e., slow start) that recovers

from losses and avoids driving the network into congestion be implemented. Thus, after a timeout, slow start is initiated with (*ssthresh* = *cwnd/2*).

Finally, Tahoe adds the fast retransmit algorithm to the original TCP implementation. The fast retransmit algorithm is executed upon the reception of enough (three) duplicate acknowledgements indicating a missing packet. Thus, upon the arrival of the third duplicate acknowledgement, Tahoe immediately sends the lost packet immediately and enters the slow start phase avoiding the need for a timeout.

4.1.2 The Introduction of TCP Reno

In April 1990, Van Jacobson sent an email [19] describing a new congestion control mechanism. In his email, Jacobson illustrated that the sender has two window controlling mechanisms: slow start and congestion avoidance. If the duplicate ACK threshold is small (3 acknowledgements in practice [20]) compared to the bandwidth-delay product, the loss will be detected with the pipe almost full. To illustrate, for a congestion window of 18 packets, the loss is detected with the pipe (1 – 3/18 = 83%) full. One might be tempted to say that a 10% loss might lead to 10% degradation in throughput. However, according to [19], this results in a severe degradation of throughput (50%-75%) due to the frequent slow starts. Thus, it would be very beneficial, especially for long fat network pipes (LFNs) (or large delay-bandwidth lines), to have the average throughput degrade as a function of the loss probability rather than the product of the loss probability and the bandwidth of the pipe. Hence,

the target of Reno is to keep the pipe nearly full and to always have accurate estimates about the packets traveling in the pipe.

In short, Jacobson states that TCP "Reno" should enter a new mode, called the recovery mode, upon the reception of three duplicate acknowledgments instead of going to slow start with *cwnd = 1* as in TCP Tahoe. Thus, upon the reception of 3 acknowledgements, the lost packet is retransmitted, the congestion window is halved, and the slow start threshold (*ssthresh*) is set to the same value of the new congestion window size. Afterwards, the flow enters the recovery mode during which *cwnd* is increased by one upon the reception of any acknowledgement. This process is repeated until the acknowledgement of the lost packet is received or the timeout value has been exceeded. At that point, the recovery mode ends and the congestion avoidance phase starts with *cwnd* reset to *ssthresh*. Table 4.1 illustrates an example of the operation of TCP Reno.

Table 4.1: Reno Operation
[Duplicate acknowledgements are assumed, *ssthresh* = 4]

Round	*cwnd*	Packets	# ACKs	Mode
1	1	1		Slow Start
2	2	2 3	1	Slow Start
3	3	4 5 6	1	Slow Start
4	5	7 8 9 10 11	2	Cong. Avoid
5	6	12 13 [14] 15 16 17: 14 is lost	3	Cong. Avoid
6	6	14 15 16 17 18 19	3	F. Recovery
7	4	20 21 22 23 24	2	Cong. Avoid

On round 5, packet 14 is lost. On round 6, 3 duplicate acknowledgements on packet 14 are received (for packets {12,13}, {15,16}, {17}). This results in entering the fast

recovery mode in round 6. Thus, the new congestion window is given by $cwnd = newssthresh = \max\left\{2, \left\lceil \frac{cwnd}{2} \right\rceil\right\}$. But *cwnd* is modified immediately to account for all those packets leaving the network (since 3 duplicate acknowledgements of packet 14 due to packets (15, 16, 17). This results in an increment of 3 to the current value of *cwnd*. This allows sending two more packets (18 and 19). In round 7, the acknowledgement of packet 14 is received along with the acknowledgement of packets 18 and 19 (in practice the receiver sends a cumulative acknowledgement for packet 19). Thus, *cwnd* is updated to be 8. But since the fast recovery mode ends by receiving the acknowledgement of 14, the congestion mode starts again with *cwnd* = 8/2=4. Note that when returning to the congestion avoidance mode again, there is no sudden burst of packets (i.e., $cwnd \neq 8$) and that is what Jacobson meant by saying "there is no sudden burst of packets as the `hole' is filled". It is noteworthy to reemphasize that the fast recovery is performed in one round which is much faster than recovering by slow start, which requires 4 steps to achieve the same *cwnd* of 4, as in Tahoe.

4.1.3 Introduction to TCP SACK (Selective Acknowledgements)

TCP SACK extends TCP Reno by including new fields (up to three) in the TCP segment header for selective acknowledgements (SACK) which allows the receiver to inform the sender about multiple received blocks. In other words, the gaps in the acknowledgements will indicate losses. For example, if the receiver informs the

sender that it received packets from 1-4, 6-8, this means that packet 5 is lost. SACK performs the fast recovery phase exactly as in Reno. The main difference is that SACK is able to send multiple lost packets in the same round. More details on multiple loss scenarios in TCP SACK can be found in [21].

4.1.4 Delayed Acknowledgements

Normally, TCP does not send an ACK the instant it receives data. Instead, it delays the sent ACK, hoping to have data going in the same direction as the ACK, so that the ACK can be sent along with the data (ACK piggybacked with the data) [22]. The main reason why delayed acknowledgements are used is to avoid overwhelming the link with small ACK packets unnecessarily. When the first packet in the flow arrives, the receiver waits eagerly for another packet but in vain. Thus, the acknowledgement for this packet is sent after the acknowledgement timer expires. For example, Windows NT4 and Win95 use a delay timer that has a uniform random distribution between (100-200 ms) while UNIX uses a uniformly distributed timer between (0-200 ms) [23].

4.1.5 Round Trip Time and Timeout Estimation

TCP sets the timeout value based on its estimated value of RTT between the two ends of the connection [24]. Since RTT values vary depending on: the connection's type, bandwidth, and the congestion conditions, the choice of a proper timeout value is not

a straightforward problem. The original approach of estimating the round trip time before the Internet congestion collapse in 1986 was based on the following algorithm:

$$Estimated\_RTT = \alpha \times Estimated\_RTT + (1-\alpha) \times Sample\_RTT$$
$$Timeout = 2 \times Estimated\_RTT$$

where α is between 0.8 and 0.9. The choice of a large value of α leads to a stable behavior; however, it might not be quick to adapt to changes in the network. On the other hand, small value of α might result in intolerable instabilities and oscillations. Note that a value of twice the estimated RTT was used in order to account for the variance in the RTT value.

An enhancement was introduced to the original algorithm by Karn and Partridge to have a back-off similar to Ethernet. Thus, whenever the flow times out, the timeout value is doubled for the next time. Moreover, they solved the problem of the ambiguity associated with the received acknowledgements after retransmitting packets. The difficulty was in the fact that it is impossible to know with which packet this acknowledgement is associated (the original or the retransmitted). The solution was to ignore those acknowledgements when estimating RTT.

In 1988, Jacobson and Karels (in two different papers) proposed a new method of estimating the RTT by taking the variance of the samples into consideration. The new approach is as follows:

$$Difference = Sample\_RTT - Estimated\_RTT$$
$$Estimated\_RTT = Estimated\_RTT + (\delta \times Difference)$$
$$Deviation = Deviation + \delta(|Difference| - Deviation)$$
$$Timeout = \mu \times Estimated\_RTT + \Phi \times Deviation$$

where δ is a value between 0 and 1, $\mu = 1$ and $\Phi = 4$. It is clear that for a large variation of RTT, the *Deviation* term will dominate. On the other hand, for low variation of RTT, the *Timeout* value is close to the *Estimated_RTT* [24].

It is noteworthy to compare Jacobson/Karels method with the original scheme. As the load on the network increases, the delay and its variance increase. The factor of 2 was chosen to take care of RTT variations in the original algorithm is not appropriate as it works only for loads up to 30% [19]. In fact, this method was proven to be completely inefficient for congested networks as it worsens the situation. An underestimation of the timeout value makes the sender retransmit the packet too early, adding more congestion to the network. On the other hand, in Jacobson's algorithm, the RTT variance is estimated and the timeout value takes this estimate into consideration. Figure 1 illustrates the retransmit timer estimated values based on the real network RTT samples.

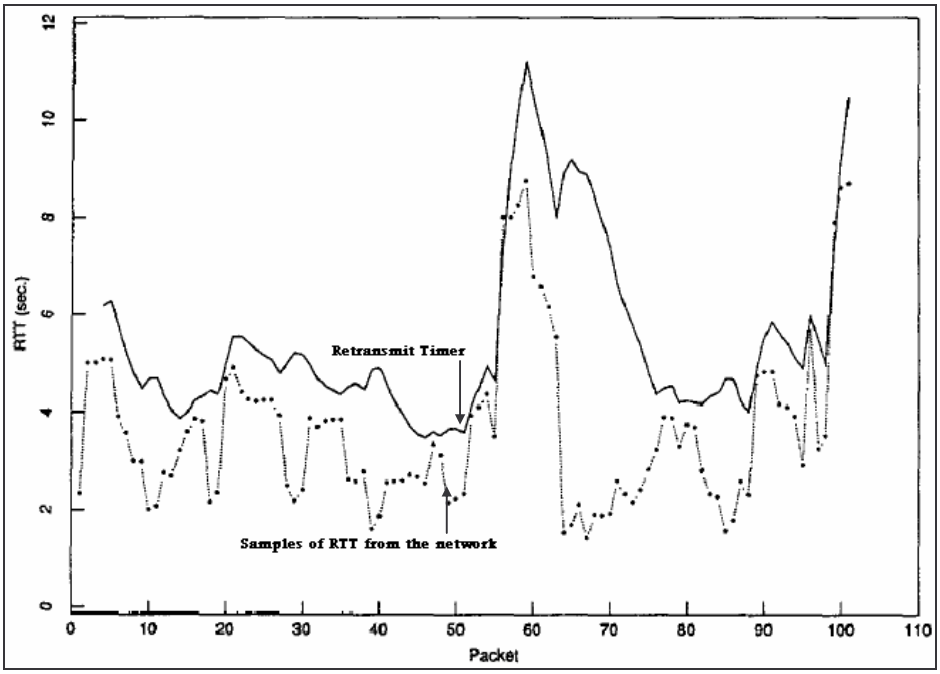

Figure 4.1 Performance of a Mean+Variance retransmit timer (Adapted and modified from [19])

4.1.6 A Brief Discussion of Round Trip Time over Wireless Links

TCP has been optimized to run over wired links with low bit error rates (BER). A packet drop typically indicates congestion over wired networks. However, in wireless networks, this assumption does not hold [25]. A packet might be dropped due to a link error as the BER of the wireless link is usually much higher than the one for wired networks. Thus, running TCP over a wireless link may result in a large number of unnecessary timeouts which severely impair the observed TCP throughput.

In order to alleviate this problem, a reliable transmission mechanism is added to the medium access layer. This is achieved by an automatic repeat request (ARQ) mechanism that is completely transparent to TCP. As a result, one can think of TCP as a mechanism that only protects against end-to-end packet losses primarily from congestion while ARQ protects against packet errors between the wireless hops. Figure 4.2 shows the architecture of TCP over wireless links with ARQ in the data link layer.

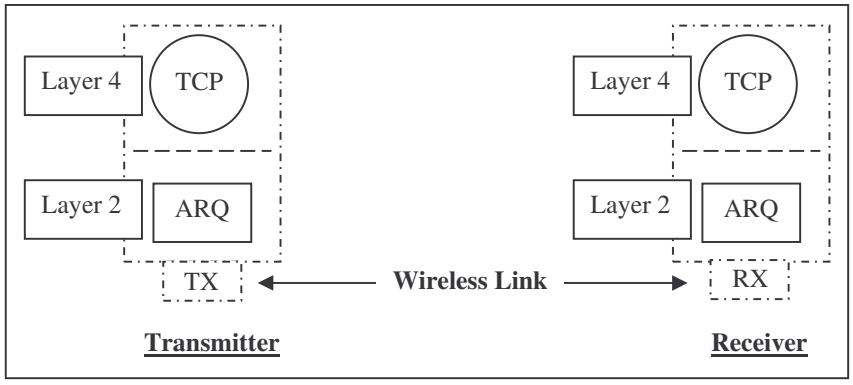

Figure 4.2: TCP over Wireless Link

There are many types of ARQ algorithms in the literature, one example is the Go-back-N ARQ scheme described in [26]. The ARQ transmitter divides the TCP segment into *n* ARQ frames as shown in Figure 4.3. The *n* frames are sent over the wireless link using a transmission window of N frames. The choice of the ARQ's frame size depends on the communication system design parameters in terms of the expected fading scenario, the Doppler bandwidth of the channel, the BER of the channel, etc [27].

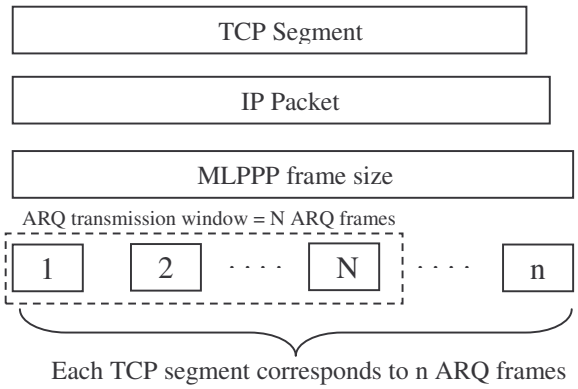

Figure 4.3: ARQ Operation
[Assuming that an IP packet is mapped to one Layer 2 (MLPPP) frame]

In the Go-back-N ARQ protocol, once an ARQ acknowledgement (ACK_{ARQ}) is received, the ARQ window is moved forward by one frame. If the receiver receives a single frame in error (note that the corrupt frame marks the beginning of a cycle (window) of N frames as the previous frames are already acknowledged), the receiver drops the following N-1 frames and sends back an explicit message instructing the transmitter to go back to the first frame in the cycle (which is the corrupt frame). It follows that since the receiver ignores the next N-1 frames, then having multiple errors within the same transmission window does not cause multiple retransmissions. In other words, in a window of 5 frames, if the first and the third frames are corrupt, the transmitter needs to retransmit the 5 frame window only once to fix the errors.

Here, TCP acknowledgements (ACK_{TCP}) are contingent on the arrival of the ACK_{ARQ} for the whole *n* ARQ frames. Since, TCP works on top of ARQ, the TCP's round trip time will suffer from an increased mean and variance due to the ARQ retransmissions. For example, the standard deviation of RTT_{TCP} over Iridium using

[NAL Model A3LA-D] modems is in the vicinity of 8 seconds with an approximate average of 20 seconds. Although the design details of the ARQ mechanism for the modems are not published, the following example may explain the high value (20 sec) of the observed RTT. Recall from Chapter 2 that under high loads MLPPP does not fragment packets in order to use the available bandwidth as efficient as possible. This means that a packet is always transmitted on a single 2.4 kbps link. For a single Iridium connection, the theoretical RTT is 6 sec taking the effect of the ARQ into account (the details of estimating the single link RTT with ARQ is given in [25]). The average of the observed RTT, however, is 20 seconds irrespective of the number of modems. This is due to the fact that Linux 2.6.x kernel implements a transmission queue size of 3 packets (including the packet being sent). Figure 4.4 shows a simplified conceptual diagram for the operation of MLPPP over Iridium. Under full load (steady state conditions), all the modem queues will be full as well as the MLPPP queue.

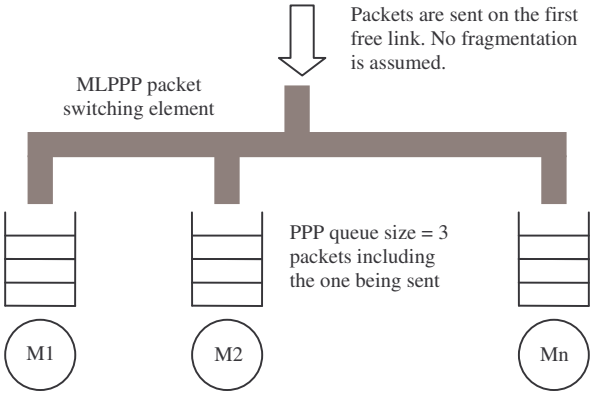

Figure 4.4: MLPPP operation over N Iridium Modems

As a result, a new packet that arrives needs to wait for the two packets ahead of it to be processed before it gets served. If T_{MP} is the average modem transmission (clocking) time plus time spent in ARQ retransmissions (see eq. (47)) and Q_P is the MLPPP queue size, then the RTT is given by,

$$RTT = Q_P T_{MP}$$

For the Iridium system, since the propagation delay = 0.5 sec, then $T_{MP} = 6.5$ sec (see [25]), and $Q_P = 3$ packets, then the average RTT = 19.5 sec which is approximately equal to the average observed RTT. Here, note that the transmission time of a single packet is given by $\frac{1500*8}{2400} = 5 \text{ (sec)}$. In other words, if the channel was completely error free, i.e., no ARQ retransmissions, then (T_{MP} = transmission time = 5). As a result, the average observed RTT would be 15 seconds. Finally, it is noteworthy to say that such RTT is different from the one measured by a 64 byte ICMP packets using the PING tool as such a low packet size does not overload the link in a similar way as the TCP transfer does. Moreover, a 1500 byte PING can not be used to imitate TCP because the PING reply (1500 bytes) is much larger than the small TCP ACKs which are of the order of few bytes. Figure 4.5 shows a 100 byte PING RTT distribution and Figure 4.6 shows the RTT distribution of a TCP RTT based on RTT measurements from a file transfer using FTP.

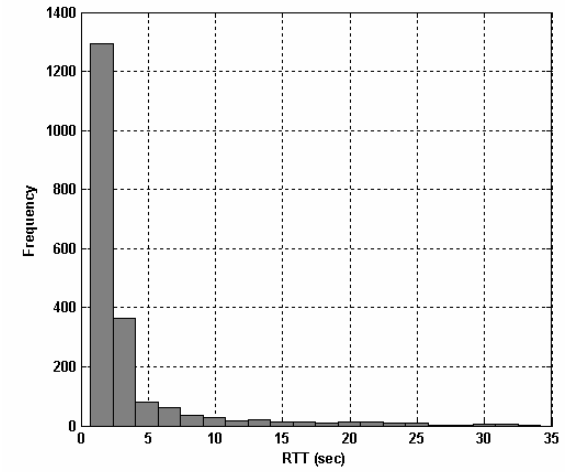

Figure 4.5: PING RTT Distribution
[PING packet size 100 bytes, PING every 2 sec, 2000 Measurements, Greenland - Kansas , 7/16/2005, Mean =3.3 sec, Standard Deviation = 4.8 sec]

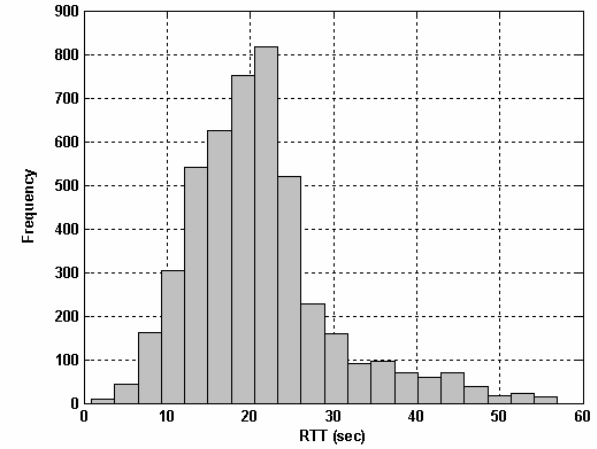

Figure 4.6: TCP RTT Distribution
[TCP segment size 1448 bytes, 4645 Measurements, 6 Modems, ITTC Laboratory, 3/19/2005, Mean = 21 sec, Standard Deviation = 8.7 sec]

Since TCP and ARQ implement similar retransmission mechanisms (although ARQ is usually a much simpler protocol), they may enter into conflict and hence cause throughput degradation [25]. The conflict occurs when the ARQ retransmission process spends more time than the TCP timeout value. Thus, the TCP timeout fires and causes the stream to go into slow start. Finally, it is noteworthy to state that if the

ARQ algorithm used has a predefined maximum number of retransmissions, then the ARQ recovery failure may result in a TCP packet loss.

4.2 General Modeling of TCP's Transfer Time

Over the past ten years, many contributions were made to model the latency and throughput of TCP in terms of round trip times and loss probabilities. Initially, researchers started with modeling long file transfers, for example FTP transfers, as this leads to closed form solutions [28]. Nevertheless, few other researchers tried to model finite (short term) TCP flows such as HTTP transfers. Developing analytical models provides better understanding of the sensitivity of TCP to several network parameters (such as the packet loss rate and the round trip time), and helps the TCP designers develop enhanced protocols and evaluate their efficiency.

In this section, the two primary TCP transfer time models (long term transfers and short term transfers) are discussed. More focus will be on the bulk (long term) transfer models as it is the most suitable to estimate the long term FTP transfer time over an Iridium connection.

In order to develop TCP latency models, it is necessary to have mathematical representations for several operational aspects of TCP which play a role in defining the TCP throughput. Such operational aspects include: the connection setup time, the

delayed acknowledgement timer effect, the mean timeout value, and the packet-loss patterns incurred.

4.2.1 Models for Some General TCP Sub-Procedures

Here, a brief introduction on modeling connection establishment, timeout interval, and the effect of delayed acknowledgements on the period of the slow start phase is given. Modeling connection establishment and the effect of delayed acknowledgements is very important for short transfers as they might have a considerable impact on the transfer time. However, models that try to characterize long TCP transfers ignore these effects as they are insignificant to the total transfer time.

Connection Setup Time

Let $P_h(i,j)$ be the probability of having a successful 3 way handshake [29], with a probability of i failures for transmitting SYNC packets followed by j failures for SYNC ACK packets is given by:

$$P_h(i,j) = P_r^i(1-P_r)P_f^j(1-P_f)$$

where P_r is the probability of failure in transmitting the SYNC packet and P_f is the probability of failure in transmitting the SYNC ACK packet.

Let Ts be the timeout period for each SYNC message to be declared lost. Since the timeout value doubles after having a timeout, the latency can be found as follows:

$$Latency(i,j) = L_h(i,j) = RTT + \sum_{k=0}^{i-1} 2^k.T_s + \sum_{k=0}^{j-1} 2^k.T_s$$

$$L_h(i,j) = RTT + (2^i + 2^j - 2).Ts$$

Thus, the expected value of the latency can be approximated as follows [29],

$$E\{L_h\} = RTT + TS\left\{\frac{1-\Pr}{1-2\Pr} + \frac{1-P_f}{1-2P_f} - 2\right\}$$

if $Pr = P_f = p$, then as stated in [30]:

$$Tsetup = E\{L_h\} = RTT + 2TS\left\{\frac{1-p}{1-2p} - 1\right\}$$

The Effect of Delayed Acknowledgements on the Slow Start Phase

In the current TCP implementations, the receiver sends one ACK if it receives two packets, or if the acknowledgement timer expires, whatever occurs first. The delayed acknowledgements timer in UNIX implementations is defined as a random variable uniformly distributed between 0 to 200 ms (expected delay of 100 ms), while in MS-Windows based systems timer values are uniformly distributed between 100 to 200 ms (expected delay of 150 ms) [29].

In the case of delayed acknowledgements [24], the congestion window grows by one MSS upon the reception of two acknowledgements:

$$CWND = CWND + MSS\left\lceil \frac{\#ACK}{2} \right\rceil$$

It is essential for transfer delay estimation in the case of short transfers, which spend most if not all of the time in the slow start phase, to be able to model the congestion window size as accurate as possible. Alman and Paxon [31] proposed the following method to estimate the congestion window size during the slow start phase.

$$cwnd_{i+1} = cwnd_i + \frac{cwnd_{i+1}}{b} = (1+1/b)cwnd_i = \gamma cwnd_i$$

where b is the number of acknowledgements sent by the receiver for every k segments received and i is the round number. Since $b=2$ in most implementations of TCP, then it follows that $\gamma = 1.5$ because the receiver sends one acknowledgement for every two packets.

Sikdar's model [30], however, considers the effect of the acknowledgement timer expiration on the development of the congestion window. Sikdar's proposed window growth function averages the two cases with delayed ACK timer expiration and without ACK timer expiration, to find *cwnd(n)* as follows:

$$cwnd(n) = \left\lfloor 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} \right\rfloor \qquad \text{where } n \text{ is the number of rounds}$$

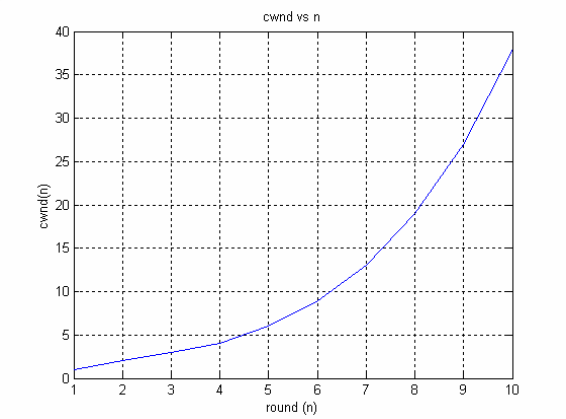

Figure 4.7: Congestion Window (*cwnd*) vs the Round Number (*n*)

Expected Value of Timeouts

The average timeout period is estimated taking the exponential back-off of the re-transmit timer into account [28]. The first step is to consider the probability distribution of the number of timeouts (k) in a timeout sequence given that there is a timeout, i.e, $P(R = k / k > 0)$. If the probability of loss is denoted by p then there are $(k-1)$ timeouts and one successful transmission.

$$P(R = k / k > 0) = p^{k-1}(1 - p)$$

$$E\{R\} = \sum_{k=1}^{\infty} k.P(R = k / k > 0) = \frac{1}{1 - p}$$

Now, the target is to estimate the average duration of the timeout sequence. The standards state that exponential back-off takes place up to 6 times afterwhich a constant value of $64T_0$ is used instead [28].

$$L_k = \begin{cases} (2^k - 1)T_0 & k \le 6 \\ 63 + 64(k - 6)T_0 & k \ge 7 \end{cases}$$

Thus, the expected value of the timeout period can be calculated as follows:

$$E\{To\} = \sum_{k=1}^{\infty} L_k P[R = k / k > 0] = T_0 \frac{1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6}{1 - p}$$

4.2.2 Packet Loss Models

Most TCP models in the literature assume that only packets may be lost but not the acknowledgements. The packet loss probability is assumed to be fixed in most models. The assumption made about the packet loss mechanism is a key issue in the

discussion of TCP performance. According to [32], there are two common packet loss mechanisms:

- Bernoulli (independent packet loss pattern): The loss of a packet does not affect any of the following packets. All packets in the flow suffer from the same loss probability, *p*. This model is most suitable for networks that implement RED [30] as packets are lost according to a uniformly distributed random variable. This model may also be reasonable for wireless connections running ARQ, especially in the cases with fast fading, were losses may be assumed to be independent (as discussed in [25]).
- Drop-Tail (correlated losses pattern): In this mechanism, if a packet gets lost in a particular round all the following packets in the same round are lost as shown in Figure 4.8. In other words, before the first packet loss, any packet is lost with a probability of *p*. However, after this loss, all packets in the same round are lost. This model is suitable for Internet as most routers use simple drop-tail (FIFO) queues which drop all received packets when their queues get full.

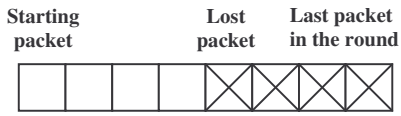

Figure 4.8: Correlated packet losses in a particular congestion window

It is clear that the loss assumptions play an important role in estimating TCP throughput. For example, a simulation based performance comparison between

Tahoe, Reno and SACK was carried in [33] where losses are assumed to be independent had more optimistic delay results than in the model in [30], which assumes correlated losses.

In this thesis, the physical layer errors are assumed to follow a two-state Markov model. In fast fading scenarios, the losses become independent (see [25] and [27] for more details). TCP is losses due to call drops are assumed to be correlated (see Chapter 5 and Chapter 6 for complete details). The reasons why those models are chosen will be clear after reading Chapter 5 and Chapter 6.

4.2.3 Modeling Long Lived Transfers

Modeling the TCP transfer latency for long transfers, such as FTP transfers, received more attention as it is possible to reach tractable analytical results without sacrificing accuracy. Padhye's model, proposed in [28], is one of the most popular models used for predicting the latency for long transfers for TCP Reno. The discussion in this section is largely based on Padhye's model.

From the previous discussion about Reno, it is clear that a loss indication is triggered by the reception of triple acknowledgements for the same packet. Thus, if the slow start phase is ignored (which is a fair assumption for long transfers as it takes up only few cycles relative to the over all number of cycles), one can view the TCP flow as a random periodic process with period limits defined by the triple duplicate

acknowledgements (TDP)s. During a TDP, the flow is running in the congestion avoidance mode. A TDP starts with an initial congestion window (W_i) equal to half the congestion window size at the end of the previous period as illustrated in Figure 4.9 (For the moment, timeouts are ignored).

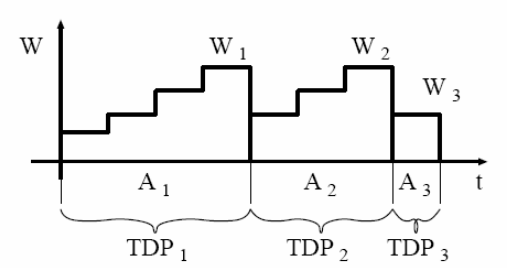

Figure 4.9: Evolution of window size (W) over time in terms of TDPs (Adapted from [28])

Assume a TCP flow (without timeouts) consisting of N TDP_i periods starting at half the previous transmission window (W_{i-1}) and ending by a loss indication signaled by receiving triple acknowledgements for the same packet. If A_i is the duration of a TDP_i and Y_i is the number of packets transmitted during A_i, then it can be shown that the average throughput (B_{NT}) can be written as:

$$B_{NT} = \frac{E\{Y\}}{E\{A\}} \qquad (1)$$

Let X be the penultimate (last) round index in the i^{th} TDP and b be the number of packets per acknowledgement, then the congestion window undergoes a linear increase with a slope of $1/b$ as shown in Figure 4.10.

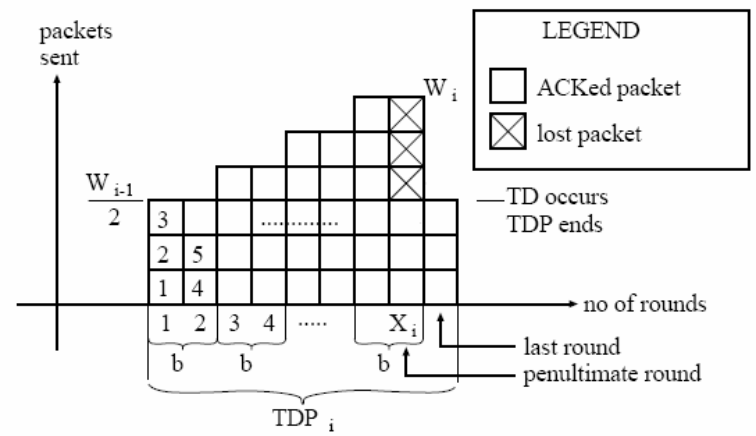

Figure 4.10: Packets sent during a TDP (adapted and modified from [28])

Since b = 2 in most implementations (due to delayed acknowledgements), then the linear development of the congestion window size during a TDP_i is given by:

$$W_i = \frac{W_{i-1}}{2} + \frac{X_i}{2} \tag{2}$$

If r_{ij} represents the round trip time in a TDP_i, then it follows that:

$$A_i = \sum_{j=1}^{X_i+1} r_{ij}$$

Assuming that the congestion window growth is independent of the round trip time, then:

$$E\{A\} = (E\{X\}+1)E\{r\} \tag{3}$$

Let p be the probability of a packet loss, then it can be shown that the average value of the number of packets sent during a TDP_i is [28]:

$$E\{Y\} = \frac{1-p}{p} + E\{W\} \tag{4}$$

As shown in [28], the average congestion window size and the average number of round trips in a TDP period is given by (5) and (6).

$$E\{W\} = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \tag{5}$$

$$E\{X\} = \frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{2+b}{6}\right)^2} \tag{6}$$

Consequently, substituting (5) and (6) into (3) and (4) respectively, one gets:

$$E\{Y\} = \frac{1-p}{p} + \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \tag{7}$$

$$E\{A\} = RTT\left(\frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{2+b}{6}\right)^2} + 1\right) \tag{8}$$

Thus, the throughput for a flow without timeouts is found by substituting (7) and (8) into (1) as shown in (9).

$$B_{NT} = \frac{\frac{1-p}{p} + \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2}}{RTT\left(\frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{2+b}{6}\right)^2} + 1\right)} \tag{9}$$

Once the throughput equation (with no timeouts) has been established, one can follow similar analysis to discuss the estimated time for a flow that undergoes timeouts. Let Z^{TO} be the duration of a sequence of consecutive timeouts, and Z^{TD} be the interval between two consecutive timeout sequences. Then, in this case the TCP period S can be defined as the sum of the time spent without timeouts (which is subdivided into sub TDP periods) and the timeout interval Z^{TO} (see Figure 4.11).

$$S = Z^{TD} + Z^{TO} \tag{10}$$

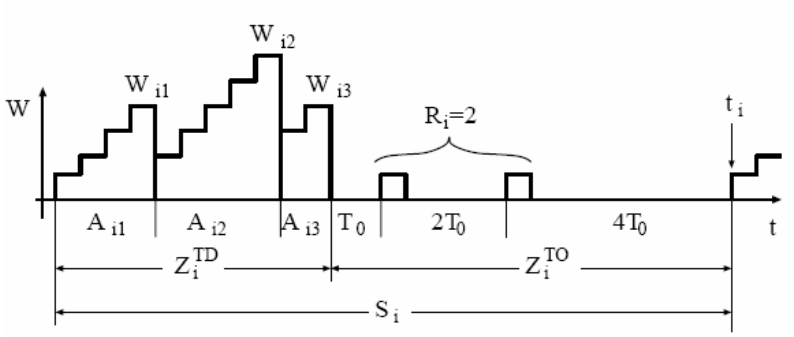

Figure 4.11: TCP flow with timeouts (Adapted from [28])

Now, one can extend the previous analysis about flows without timeouts by defining the following set of variables.

n_i : Number of TDPs spent in a Z_i^{TD} period

Y_{ij} : Number of packets sent in $TDP_{(i,j)}$

A_{ij} : Time period spent in a $TDP_{(i,j)}$

X_{ij} : Number of rounds spent in $TDP_{(i,j)}$

W_{ij} : Window size at the end of a $TDP_{(i,j)}$

R_i : Number of packets sent during the timeout period Z_i^{TO}

M_i : Number of packets sent during the S_i period

Let M be the number of packets sent during the TCP period (S), then the throughput for a TCP flow with timeouts can be written as:

$$B_{TO} = \frac{E\{M\}}{E\{S\}} \tag{11}$$

Let Y_{ij} be the number of packets sent in $TDP_{(i,j)}$ and n_i be the number of TDPs in the Z_i^{TD} period. Then, it follows that M_i , the number of packets sent in S_i period, is found by summing all Y_{ij} packets and any packets (R_i) sent during the timeout period.

$$M_i = \sum_{j=1}^{n_i} Y_{ij} + R_i \tag{12}$$

If A_{ij} denotes the time period spent in a $TDP_{(i,j)}$ then the TCP period (S) is given as:

$$S_i = \sum_{j=1}^{n_i} A_{ij} + Z_i^{TO} \tag{13}$$

In short, after taking the loss scenarios into consideration, it was shown in [28] that after estimating E{S} and E{M}, the throuput of a TCP flow with timeouts can be approximated by:

$$B_{TO} \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \tag{14}$$

4.2.4 Modeling Short Lived Transfers

For completeness, it is beneficial to provide a summery of the development of short transfer models. Caldwell [29] was among the first researchers to address the problem of modeling short lived transfers where he tried to predict both long and short transfer times. The proposed model extends Padhye's steady state model by taking the connection establishment time and slow start into account. The transfer latency

equation is given as a function of the transfer size, average round trip time (RTT), and packet loss rate.

One of the shortcomings of Caldwell's model is that it ignores the effect of the delayed acknowledgements timer on the development of the congestion window especially for short transfers. Assuming an acknowledgement every two packets, the average window size at the end of the i^{th} round is given as,

$$cwnd_{i+1} = (1+1/b)cwnd_i = \gamma\, cwnd_i \qquad (15)$$

To get more accurate latency estimates for short transfers, Sikdar [30] accounted for the effect of the delayed acknowledgements timer expiration and proposed a new average window growth function as:

$$w(n) = \left\lfloor 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} \right\rfloor \qquad (16)$$

According to [30], the number of packets transmitted in the first k rounds, denoted as *pkt(k)* (where k is the round number), can be expressed as,

$$pkt(k) = \sum_{n=1}^{k} w(n) = 2^{\frac{k+1}{2}} + 3(2)^{\frac{4k-3}{8}} - 2 - \frac{3\sqrt{2}}{2} = 2^{\frac{k+1}{2}} + 3(2)^{\frac{4k-3}{8}} - 4.12 \qquad (17)$$

Here, *pkt(k)* gives the index of the last packet in the k^{th} round. Note that the slow start ends when the slow start threshold (*ssthresh*) or if the maximum window size is reached. In order to model the transfer time for a certain flow, an average estimate is evaluated for three different cases: transfer without losses (t_{nl}), transfer with a single loss (t_{sl}), and transfer with multiple losses (t_{ml}) as:

$$T_{transfer}(N) = t_{setup} + t_{dack} + (1-p)^N t_{nl}(N) + p(1-p)^{N-1} E\{t_{sl}(N)\} + (1-(1-p)^N - p(1-p)^{N-1}) t_{ml}(N) \quad (18)$$

where p is the packet loss probability, N is the total number of packets, t_{setup} is the connection setup time and t_{dack} average delayed acknowledgement time after transmitting the first packet.

Sikdar's analysis for estimating t_{nl} is summarized here. Let N_{wm} represent the number of rounds required to reach the maximum window size, W_{max}. This could be found by solving (16) for $w(n) = W_{max}$

$$n_{wm} = \left\lfloor 2\log_2\left(\frac{2W_{\max}}{1+\sqrt{2}}\right)\right\rfloor \quad (19)$$

The expected value of the number of packets transferred when *cwnd* reaches $W_{max} = N_{exp}$ (n_{wm} does not include the last round with $W_{\max}$ packets) can be expressed as:

$$N_{\exp} = pkt(n_{wm}) + W_{\max} = \sum_{n=1}^{n_{wm}} w(n) + W_{\max}$$
$$N_{\exp} = \left\lfloor 2^{\frac{n_{wm}+1}{2}} + 3(2)^{\frac{4n_{wm}-3}{8}} - 2 - \frac{3\sqrt{2}}{2}\right\rfloor + W_{\max} \quad (20)$$

According to Sikdar, the transfer time for a lossless transmission of N packets is

$$t_{nl}(N) = \begin{cases} \left\lceil 2\log_2\left(\frac{2N+4+3\sqrt{2}}{2\sqrt{2}+3(2)^{5/8}}\right)\right\rceil RTT & N \le N_{\exp} \\ \left[n_{wm} + \left\lceil \frac{N-N_{\exp}}{W_{\max}}\right\rceil\right] RTT & N > N_{\exp} \end{cases} \quad (21)$$

The next step is to estimate the transfer time for a single loss case for a flow consisting of *N* packets. Note that the correlated loss model is assumed, i.e. packets from the i^{th} packet to the last packet in that round are lost. The effect of fast recovery is considered in the analysis by taking either one or two rounds after the loss indication with congestion windows of $cwnd_i^1$ and $cwnd_i^2$ respectively.

$$t_{sl}(N) = \begin{cases} [t_{nl}(i) + nloss + 1 + t_{lin}(a,n)]RTT & without\_timeouts \\ [t_{nl}(i) + t_{TO} + r(n) + E\{TO\} + t_{lin}(a,n)]RTT & with\_timeouts \end{cases} \quad (22)$$

where:

$t_{nl}(i)$	Transfer time required to transmit the first *i* packets without losses
$nloss$	Number of packets lost in the round that has the loss
$r(n)$	Number of rounds needed to reach the slow start threshold *n*
$t_{lin}(a,n)$	Time needed to transmit *a* packets in the congestion avoidance mode with an initial window size of *n*
t_{TO}	Number of rounds spent before a timeout (this is mainly a consequence of considering fast retransmissions)
$E\{TO\}$	Average duration of timeout periods

In order to estimate the time spent in the case of multiple losses, the flow with M losses is broken into two parts separated by the loss (at the m^{th} packet): a transfer with a single loss (first loss) at the i^{th} packet and ending with the second loss [$t_{sl}(m-1)$], and a transfer with M-1 losses separated by (M-2) average distances (D_{ave}) [$t_{M\_loss}(D_{ave})$].

$$D_{ave} = \frac{N_{packets\_left}}{N_{loss\_indications}} = \frac{N - m + 1}{M - 1}$$

$$t_{ml}(N) = E\{t_{sl}(m-1)\} + (M-2)E\{t_{M\_loss}(D_{ave})\} \tag{23}$$

It is clear that short transfer models are much more complex than long transfer models. This is primarily due to the fact that the time short transfers take is in the order of few rounds. To attain an accurate estimate of the average transfer time, several factors have to be taken into consideration, for example: the delayed acknowledgements timer expiration effect, the connection setup time, and the time spent slow start. Such factors were reasonably ignored in some long transfer models [28] as they do not affect the performance estimate significantly because the TCP flow spends most of the time in the congestion avoidance phase.

4.3 Comparison between TCP Performance Models

This section provides a comparison between the predictions of Padhye's and Sikdar's models based on the results in [32]. Table 4.2 shows the relative prediction error (with respect to the *ns* simulation results performed in [32]) calculated as |Model Prediction – Simulation Result| / (Simulation Result) for the two models where *p* denotes the packet error/loss probability. Results show that Skidar's model is most accurate most of the time with the exception for high error rates where Padhye's model provides more accurate results.

Table 4.2: Relative error comparison of various TCP models (Short Transfers) (Results are adapted from [32])

Model	p=1%	P=3%	p=5%	p=8%	p=10%
Sikdar	0.15	0.41	0.25	0.51	1.06
Padhye	1.05	1.35	0.77	0.52	0.42

Similar analysis is performed for long transfers as shown in Table 4.3. In this case, Padhye's model provides more accurate predictions most of the time except for very low error rates. Padhye's model is a relatively simple model which makes it very attractive to be extended to account for various effects such as ARQ [25], bit error rate, etc.

Table 4.3: Relative error comparison of various TCP models (Long Transfers) (Results are adapted from [32])

Model	p=1%	P=3%	p=5%	p=8%	p=10%
Sikdar	0.78	1.82	0.38	4.75	16.12
Padhye	0.96	1.22	0.35	2.08	10.57

Finally, this chapter summarized known results concerning TCP performance taking into consideration various operational aspects such as the channel's probability of error, the effect of the acknowledgement timer expiration, etc. Nevertheless, the operation of TCP over MLPPP needs to be addressed. Since, in this thesis, a MLPPP connection is formed by inverse-multiplexing multiple wireless links, each link is susceptible to call-drops. A single link loss due to a call-drop results in a TCP timeout, as will be discussed in Chapter 5. Thus, in order to predict TCP performance

under MLPPP, the probability density function of the time interval between call drops is estimated empirically. Then, this probability density function is used to extend Padhye's model to account for the effect of call drops on TCP transfers running over MLPPP connections (see Chapter 6).

Chapter 5. Towards the Development of a Call Drop Probability Density Function

The performance of TCP over MLPPP over Iridium has been experimentally examined in [4] and [14]. Experiments were carried out to transfer data from Kansas to Greenland using the system described in Chapter 7. TCP transfers are carried out using IPERF and packets are captured using TCPDUMP. The captured packets are studied using the TCPTrace package which provides detailed set of TCP related graphs. In this chapter, the reasons why call drops occur are discussed, the relationship between call drops and timeouts is investigated, and finally a probabilistic model for the time difference between call drops is developed.

5.1 The Call Drop Event

A call drop is the event of losing an established connection suddenly. Such event is not favorable from the user's point of view. Operators should always keep the call dropping rate as low as possible to satisfy their customers' expectations. Call drops are common in cellular systems and in circuit switched wireless systems in general. In this work, a call drop event means that one of the MLPPP channels loses its connection. Once a call drop is detected, the connection is automatically re-established by the link management software, discussed in Chapter 7. Nevertheless, in the interim, several packets are lost. There are various reasons that

might lead to call drops such as low signal-to-noise ratio (SNR), handover failure, network signaling failures, etc. In this work, two key reasons that may result in call drops are investigated: SNR and handovers.

The developed link management software, described in Chapter 7, records the call drop information (modem, time of occurrence). A module has been included in the management software to acquire SNR measurements every 4 seconds. According to [34], however, the modems used for obtaining measurements provide delayed measurements during handovers. In other words, if the modem is continuously (every 2 seconds) probed for SNR measurements, it returns the SNR measurement after 4 seconds (relative to the probing time). However, when a handover takes place, the modem returns its SNR measurement after 6 (or more) seconds. Thus, the delay property was utilized in order to detect handovers by keeping record of the measurement intervals. SNR measurements for 48 hours were performed. Then, the time differences between the call-drop measurements were calculated and sorted. After investigating the measurements, it was clear that all measurements with return periods longer than 6 seconds were separated by multiples of 10 minutes which correspond to the Iridium inter-satellite handover event. However, not all handover events were detected due to hardware limitations of the modems used. Thus, a simple linear interpolation was used to predict the missing handover events. Once the handovers table is compiled, it was matched with the call-drops table for each modem

in order to correlate call drops with the corresponding handover failures. A sample of the measurements is shown in Table 5.1.

Table 5.1: Sample Handover Measurements from Greenland

Event Time	Measured/Interpolated
7/16/2004 13:21	Predicted
7/16/2004 13:31	Observed
7/16/2004 13:41	Predicted
7/16/2004 13:51	Predicted
7/16/2004 14:01	Predicted
7/16/2004 14:12	Observed
7/16/2004 14:22	Predicted
7/16/2004 14:32	Observed
7/16/2004 14:42	Predicted
7/16/2004 14:52	Observed

In order to estimate the effect of the inter-satellite handovers on the call-drop process, 182 call drops were observed on a seven-modem MLPPP connection. The call drop times were then matched with the prepared handover time measurements table (Table 5.1). Here, exhaustive search, data manipulation and matching operations were performed using the MySQL database engine along with the Sun JAVA programming language. The results of the matching operations showed that 10% of the call drops are due to handovers. This low effect of handovers on call drops suggests that the call drop process is not a periodic random process.

For completeness, it is noteworthy to state that the SNR measurements showed that if the link's SNR gets below 0 dB for more than 16 sec, a call drop will occur. In general, it was also observed that operation below 2 dB for more then 20 sec results in call drops. Such observations, however, can not be directly used for modeling call drops as the information about the signaling layer is not published.

5.2 Call Drops Effect on TCP Performance

Multiple long TCP transfers have been performed using IPERF. TCP dumps were collected and analyzed using the TCPTrace tool. The plots generated by TCPTrace were then processed to add call drop information to the TCPTrace graph files. According to the results of [4] and [8], and the outcome of more than 50 experiments carried out as part of this work, all call drops for various number of modems in the MLPPP bundle (2 – 8) modems resulted in TCP timeouts. The reason why a call drop leads to a timeout is not obvious. Many factors are involved for the timeout occurrence and some of these factors are hardware/software dependent. It is certain, however, that a call drop leads to losing sufficient number of packets which result in a timeout most of the time. Moreover, a link loss may result in the dropping of several acknowledgements for packets that were successfully received. It is known [18] that multiple losses lead timeouts in TCP Reno. For TCP SACK to timeout, half the packets in a congestion window need to be dropped (when operating in the congestion avoidance phase) [18]. In a long delay network (given the size of: modem buffers, serial card buffers, and MLPPP buffers), the number of lost packets is the

sum of all the packets sent before detecting the call drop plus all the packets stored in the sender's and the receiver's buffers. For example, suppose that a MLPPP connection formed with the following parameters,

Datalink and physical layer parameters

Number of Modems = 4

Modem transmission rate (R) = 300 bytes/sec

Modem buffer size (B_M) = 4.5KB

Serial card buffer size (B_S)= 1.5 KB

MLPPP buffer (B_P) = 1.5 KB

Average time before detecting a call drop on the physical layer (T_L) = 10 sec

TCP parameters

Maximum TCP window size W_{max} = 22.5 KB

RTT (propagation delay) = 1 sec

TCP segment size (P) = 1.5 KB

Here, the number of packets lost before detecting a call-drop indicates the average time of detecting the loss of the physical link. Prior to this time, the upper layers assume that the data is being transmitted, while in reality the data is getting lost. Assuming that a call drop only causes the packets stored at the receiver's modem buffer to get lost (while the other buffers at the receiver's end are assumed to deliver their packets correctly), then the total packet loss is the sum of the total packets

assumed to be transmitted while in fact being lost plus all the packets stored in the transmitter's buffers (modem's and serial-card's buffers) and the packets stored in the receiving modem's buffer as,

Total packets lost = $\frac{1}{P}$ [T_L R + B_M + B_S + B_M] = $(1.5)^{-1}$ [10*0.3 + 4.5 + 1.5 + 4.5]

= 9 packets

Total number of bytes lost = 13.5 KB > $\frac{1}{2}W_{max}$. Therefore, it is obvious that even TCP SACK will timeout if a call drop occurs. Note that a very conservative value of T_L was used in this example. In the deployed Iridium system this time is at least of the order of 30 seconds which assures that if a call drop occurs then a TCP timeout event will take place.

Tests of MLPPP connections using a range from 2 to 8 Iridium modems also showed a strong correlation between call drops and timeouts. Moreover, the tests showed that timeouts due to link errors were negligible. Figure 5.1 shows a sample time sequence graph, obtained using TCPTrace, with call drops indicated by vertical lines.

Thus, after performing a wide range of TCP over MLPPP measurements, it became clear that in order to study the effect of call drops on TCP performance, it is necessary to model the call-drop process. To do that, the first step is to estimate the probability density function (pdf) of the time difference between call drops for a given MLPPP bundle, as will be shown in Section 5.3.

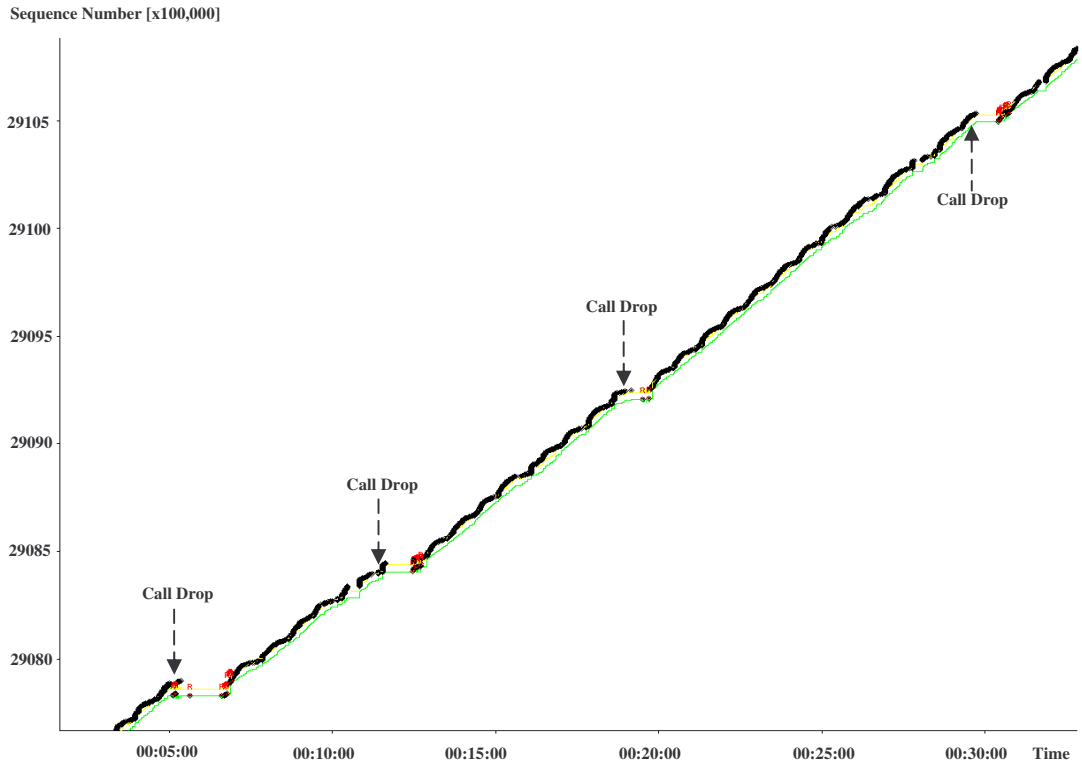

Figure 5.1: A sample TCP trace
[Experiment on 7/17 of a 33 min file transfer using IPERF from Greenland to ITTC at the University of Kansas]

5.3 The Development of the Call Drops Probability Density Function

Several tests were performed from the field (in Greenland) and the laboratory (in Kansas) in order to collect sufficient number of call drop events. Call drops were collected in a per link basis and stored in a MySQL database. A simplified structure of the MySQL table used is shown in Table 5.2, where modems were labeled after state names.

Table 5.2: A Simplified Structure of the Call Drops Table

Modem ID	Drop Time
Kansas	7/16/2004 13:51
Nebraska	7/16/2004 15:37

Measurements of 394 call-drops were collected as part of the scientific field experiments carried out between a research site in Greenland and the research laboratory at the University Kansas. In order to study call drops for the whole MLPPP bundle, the time difference between call drops (Δ) for a single link is investigated. This inter-call drop time difference (ICTD) is a function of the network. Thus in order to estimate the pdf of per link ICTD, a proper number of call drop events (394 events in this case) over multiple long connections was collected. Then, the ICTD was calculated. Results showed that the ICTD of the Greenland-Kansas connections followed an exponential distribution as shown in Figure 5.2.

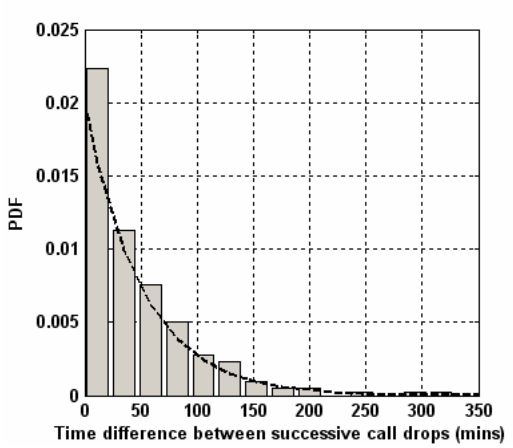

Figure 5.2: Inter-Call Drop Time Difference PDF [Greenland-Kansas 2004 Measurements]

It is noteworthy to summarize the procedure of estimating the ICTD probability density function. Once sufficient data samples were collected, a histogram of the data samples is developed. Then, the data bins of the histogram are scaled by the area under the curve so that the graph constitutes a valid pdf. Next, using a curve fitting tool, an approximate (exponential) distribution is obtained. Afterwards, the chi-square goodness-of-fit test is applied to the estimated distribution along with the data bins. Finally, the rate of the exponential distribution (β) is adjusted so that the goodness-of-fit is satisfied for the chosen significance level (5%) and the number of bins (15).

Since the per link time difference between call drops follows an exponential distribution, then the per link call-drop process is Poisson with a rate of β. Assuming n independent and identical links, it follows that the random process characterized by the time difference between call drops for the whole bundle can be modeled by merging the n Poisson random processes into a single Poisson random process with a rate of ($\lambda = n\beta$). The time difference between call drops for the whole bundle (Z^{CD}) is given by

$$f_{Z^{CD}}(t) = \begin{cases} \lambda e^{-\lambda t} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

The modeling of the bundle's pdf is summarized in Figure 5.3. The average call drop rates per modem β were studied by establishing MLPPP connections in the laboratory and by analyzing the call-drop measurements obtained from the field tests in Greenland. The average call drop rates are given in Table 5.3.

Table 5.3: Call Drop Rates (β) for Field Measurements

Connection	Kansas-Greenland	Kansas-Kansas
Per Link Drop Rate (β)	1/50 mins	1/52 mins

The main reason for the difference in the call drop rates in both cases may be explained by the fact that the transmitters and the receivers for the Kansas-Kansas MLPPP connection communicate through a single satellite, where as two different satellites connect the Kansas and Greenland sites. Thus for the second case (Kansas-Greenland connection), call drops due to handover failures, signaling failures, weather differences, etc, will increase the call drop rate.

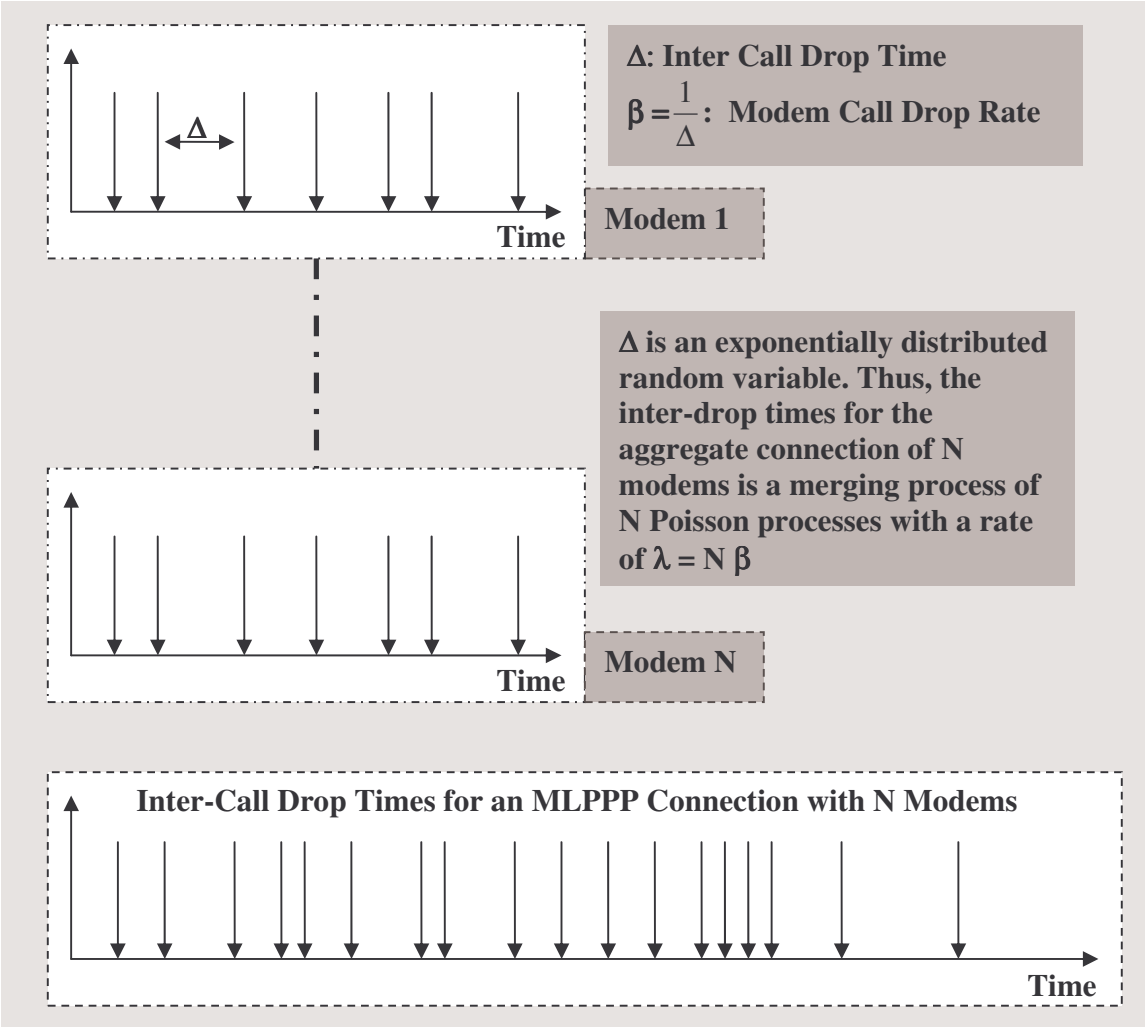

Figure 5.3: Modeling of the Call Drops PDF of a MLPPP Bundle

Now, that the call drops distribution of the bundled connection has been established, the next step is to use this result in order to estimate the TCP transfer time for large files taking the call drops effect into account.

Chapter 6. Long File TCP Transfer Time Analysis

This chapter focuses on TCP over MLPPP, specifically the evaluation of the effect of call drops on TCP performance taking the empirical estimate of the probability density function (pdf) of the call-drop process (developed in Chapter 5) into consideration. Using the estimated call-drop model, the development in [15] is extended to account for call drops. Then, the proposed TCP model (that takes call drops into account) is experimentally validated by field measurements using the Iridium network. Afterwards, TCP performance over multiple Inmarsat connections is predicted using the proposed model by varying the call drop rate, packet loss probability and the timeout interval. Finally, the effect of the ARQ on the RTT and on the packet loss rate is analyzed.

6.1 Problem Definition and Background Assumptions

The goal of this chapter is to derive an estimate of the transfer latency for a TCP connection running over MLPPP. This section explains various assumptions necessary to incorporate the call-drops effect into Padhye's model [15]. In addition to Padhye's model assumptions, the following factors need to be taken into account:

1. Each wireless link runs a physical layer reliability assurance mechanism such as automatic-repeat-request (ARQ) discussed [25] and [26] to compensate for wireless errors (see Figure 6.1). Thus, packet losses due to wireless errors

only cause retransmissions (resulting in a halving of the congestion window) but not timeouts. Here, the ARQ is assumed to have a maximum number of retrials before it delivers a corrupt packet. The delay incurred at the maximum number of retrials is assumed to be lower than the timeout value. Hence, timeouts are assumed to be solely due to call drops and the probability of packet losses visible at the TCP layer is very small.

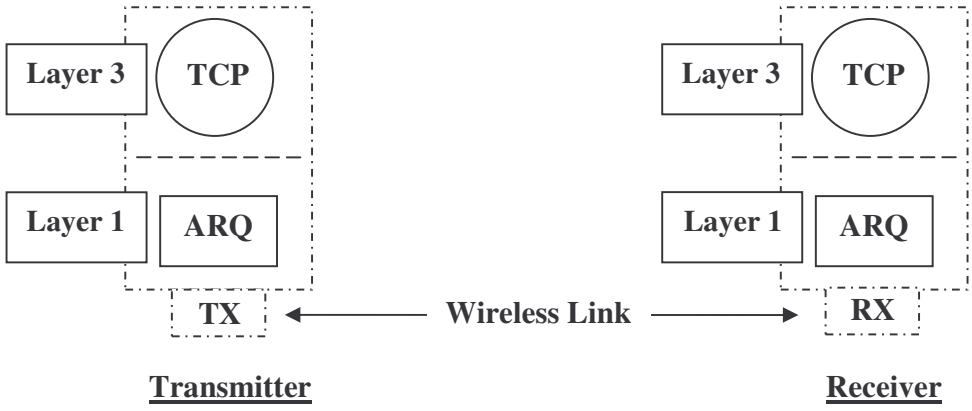

Figure 6.1: TCP running over lower layer ARQ

2. The link is restored before TCP leaves the slow start phase after experiencing a timeout caused by a call-drop. This is assumption is due to the fact that Padhye's model [15], extended here, does not consider slow start.
3. Delayed acknowledgements are assumed, leading to a window increase every 1/b packets (usually b =2).
4. The proposed derivation is meant to be a general derivation independent of any software implementations. Thus, no primary links are assumed. (A primary link is a link that its loss causes the whole bundle to be dropped).

5. Since timeouts are assured in the case of call drops, the difference between TCP versions has minor impact. Thus, the proposed method can be used to predict the performance of other TCP versions such as SACK and Tahoe.
6. Network parameters are not captured in this model due to the difficulty in validating the assumptions. For example, the number of satellite hops may impact the call drop rate due to the increased network signaling between satellites as the number of satellite hops increase.
7. In the case of a long file transfer, it is assumed that the flow runs in the congestion avoidance mode until a loss or a timeout occur.
8. In order to avoid unnecessary modeling of some implementation specific parameters (i.e., MLPPP software and modem hardware parameters), the RTT variations due to MLPPP and ARQ buffering are not modeled. Instead, the long term average RTT is used for the model prediction purposes.

6.2 TCP Model Mathematical Derivation

Based on the discussion presented in Section 4.2.3, the goal of this section is to derive a formula that considers the call-drop events in the estimation of the TCP transfer latency over a long delay MLPPP connection with average TCP throughput (B) packets/s. The TCP transfer latency for f_s bytes given the TCP maximum segment size (MSS) can be written as:

$$T_d = \left\lceil \frac{f_s}{MSS} \right\rceil B \quad \text{(sec)} \tag{24}$$

Here, B is estimated by extending the TCP latency model in [15] to include the effect of call drops. In this model, as in [15], the TCP flow is viewed as a complex periodic random process (see Figure 6.2). Each period (S) consists of two intervals: a data transfer interval (Z^D) and a timeout interval (Z^{TO}), i.e., $S=Z^D+Z^{TO}$. Since the limits of the TCP transfer period (S) are defined by call-drop events, then S is assumed to follow the call drop distribution previously denoted as Z^{CD}.

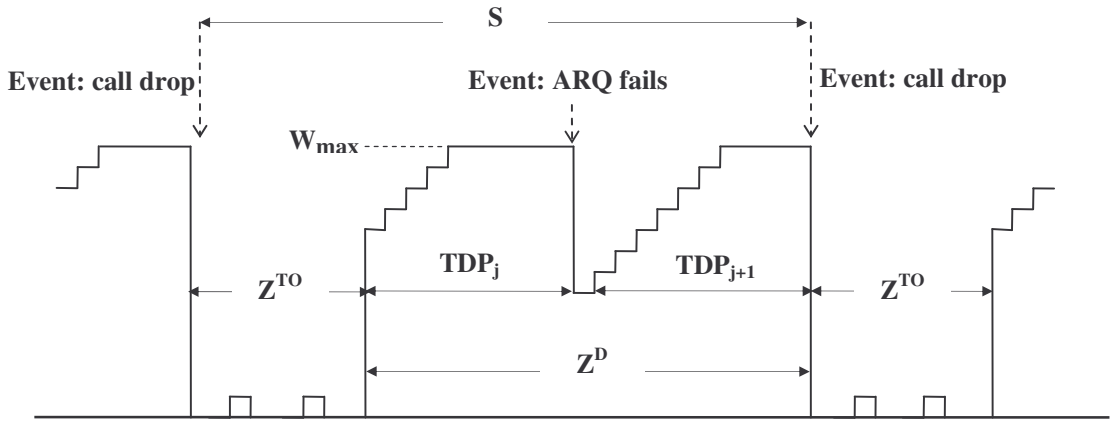

Figure 6.2: TCP Flow Period (adapted and modified from [15])

The data transfer period consists of n triple duplicate acknowledgement periods (TDP). Each TDP has a length of (A) during which (Y) packets are transmitted. In this case, it is assumed that those losses are mainly due to link errors that were not recovered by the physical layer ARQ. On the other hand, the timeout period Z^{TO} represents the time that the flow spends before it enters the slow start phase after a timeout. The number of packets sent during successive timeouts is given by R.

In order to estimate B, the TCP throughput (packets/s) in the absence of timeouts (B_{NT}), i.e., with no call drops, is considered first. B_{NT} can be written as:

$$B_{NT} = \frac{E\{Y\}}{E\{A\}} \tag{25}$$

It was shown in [15] that the mean number of packets sent during a TDP period given the packet loss probability (p) is:

$$E\{Y\} = \frac{1-p}{p} + E\{W_u\} \tag{26}$$

The average unconstrained congestion window size $E\{W_u\}$ was derived in [15] in terms of (p) as:

$$E\{W_u\} = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \tag{27}$$

Since the window size is usually restricted to a maximum value (W_{max}), then in the case considered here where the low packet error rate is small, it is reasonable to assume that $E\{W_u\} = W_{max}$.

The mean TDP period of length (A) given the average round trip time (RTT) was derived in [15] as,

$$E\{A\} = \begin{cases} RTT\left(\frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{2+b}{6}\right)^2} + 1\right) & E\{W_u\} < W_{\max} \\ RTT\left(\frac{b}{8}W_{\max} + \frac{1-p}{p.W_{\max}} + 2\right) & E\{W_u\} \geq W_{\max} \end{cases} \tag{28}$$

Substituting (26), (27) and (28) into (25) and rearranging gives,

$$B_{NT} = \begin{cases} \dfrac{\dfrac{1-p}{p} + \dfrac{2+b}{3b} + \sqrt{\dfrac{8(1-p)}{3bp} + \left(\dfrac{2+b}{3b}\right)^2}}{RTT\left(\dfrac{2+b}{6} + \sqrt{\dfrac{2b(1-p)}{3p} + \left(\dfrac{2+b}{6}\right)^2} + 1\right)} & E\{W_u\} < W_{\max} \\ \dfrac{\dfrac{1-p}{p} + W_{\max}}{RTT\left(\dfrac{b}{8}W_{\max} + \dfrac{1-p}{p.W_{\max}} + 2\right)} & E\{W_u\} \geq W_{\max} \end{cases} \quad (29)$$

If M represents the number of packets sent during a period S, then the throughput (B) (in the case of call drops) is given as,

$$B = \frac{E\{M\}}{E\{S\}} = \frac{E\{M\}}{1/\lambda} \quad (30)$$

It was shown in [3] that if T_o is the initial period for timeout then:

$$E\{Z^{TO}\} = T_0 \frac{1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + +32p^6}{1-p} \quad (31)$$

The value of M can be expressed as the product of the number of the TDP periods (n) and the number of packets sent in each TDP period plus the number of packets sent during Z^{TO}. Assuming that n and Y are statically independent, the mean value of M is given by:

$$E\{M\} = E\{n\}E\{Y\} + E\{R\} \quad (32)$$

The mean value of n can be obtained by the ratio of the means of Z^D and A as follows:

$$E\{n\} = \frac{E\{Z^D\}}{E\{A\}} = \frac{E\{S\} - E\{Z^{TO}\}}{E\{A\}} = \frac{1/\lambda - E\{Z^{TO}\}}{E\{A\}} \quad (33)$$

It was also shown in [15] that the number of packets, R, sent during Z^{TO} is given by,

$$E\{R\} = \frac{1}{1-p} \tag{34}$$

Substituting (8) and (10) into (7) gives,

$$B = \frac{E\{n\}E\{Y\} + E\{R\}}{1/\lambda} = \frac{\frac{1/\lambda - E\{Z^{TO}\}}{E\{A\}}E\{Y\} + E\{R\}}{1/\lambda} \tag{35}$$

But $B_{NT}=E\{Y\}/E\{A\}$. So, by substituting (29), (31) and (34) into (35) and rearranging one gets,

$$B = \left[1 - \lambda T_0 \frac{1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + +32p^6}{1-p}\right] B_{NT} + \frac{\lambda}{1-p} \tag{36}$$

6.3 Model Validation

The proposed TCP latency model (eq. 36) was tested using eight-Iridium links (2.4 kbps each) connecting the field site and the laboratory during the summer of 2004. Several long file transfers have been performed using Iperf and gftp client. The results, as shown in Table 6.1, agree with the predictions of the model. The variance in the results may be due to RTT variations caused by the physical layer ARQ and the MLPPP queuing.

Table 6.1: File Transfers from Greenland to the University of Kansas (Summer 2004),T0=60s, p = 5E^{-4},β = 1/50 min^{-1},MSS=1448, RTT=19s,Wmax= 47.9KB

File Size (MB)	**1.38**	**5.62**	**20.6**	**35.7**
Measured Transfer Time (min)	11	46	180	315
Predicted Transfer Time (min)	12.5	51	187	324

Model predictions were also tested for a various number of links [note here that W_{max} is a function of bandwidth]. The results agreed with the field measurements (see Table 6.2).

Table 6.2: File Transfers from the Greenland to the University of Kansas during summer 2004, T0=60s, p = $5E^{-4}$, β = 1/50 min^{-1}, MSS = 1448 Bytes, RTT=19s

Number of Links	**3**	**4**	**5**	**6**	**7**	**8**
File Size (MBytes)	4.82	0.85	1.91	1.39	3.40	1.40
W_{max} (KBytes)	16.1	22.0	28.3	34.7	41	47.9
Measured Time (min)	96	15	21	13	30	12
Prediction (min)	98.1	13.4	24.1	15.4	33.2	12.7

Next, the model will be validated at different call dropping rates. Thus in order to perform this test, a software module was built and added to the developed link management software as shown in Figure 6.3. The added software module generates call drops according to a Poisson process for any given dropping rate. Thus, one can increase the call dropping rate of the MLPPP bundle over the base provided by the Iridium system. Several 10 MB file transfers were conducted over six modems in the MLPPP bundle in the laboratory. The rates used are 1/52 min (the base Iridium system call-dropping rate), 1/30 min, and 1/20 min. The results, illustrated in Figure 6.4, indicate that the higher call dropping rate leads to an exponential increase in TCP transfer latency. The proposed model agrees well with the measurements' mean. The Iridium network call dropping rate is slightly lower than that of Greenland since the transmitters and the receivers are at the same location as discussed in Chapter 5.

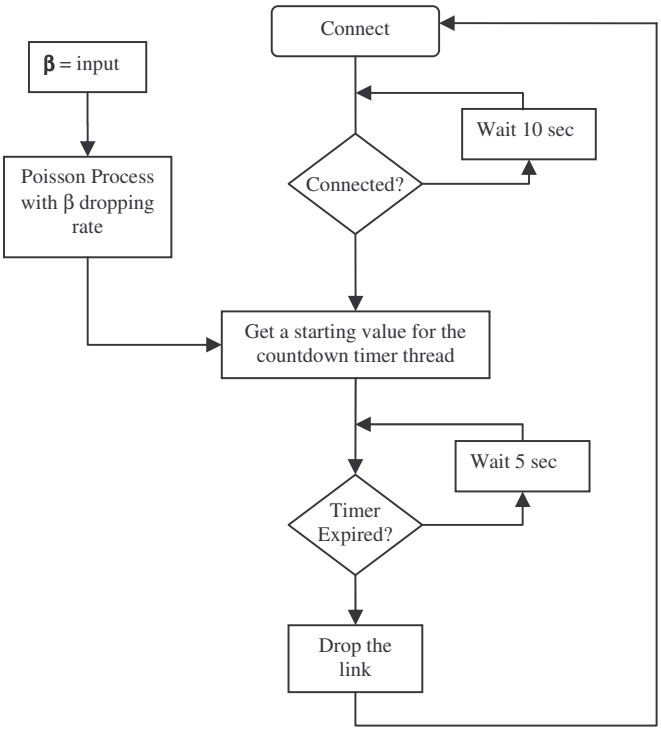

Figure 6.3: Call dropping module architecture

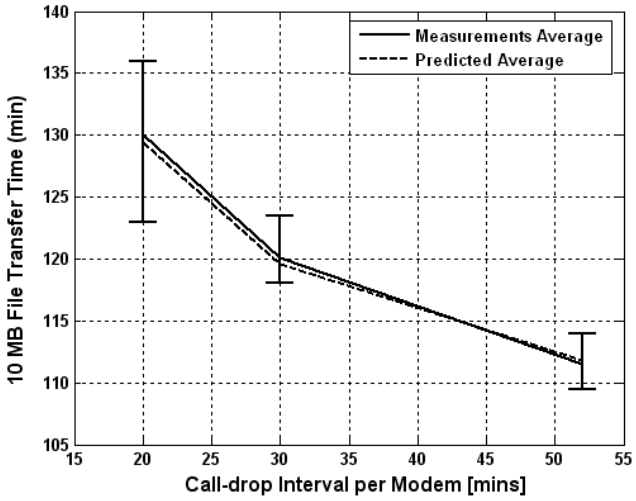

Figure 6.4: Model predictions for various call-drop rates [Observed average is based on eight measurements at each call dropping rate, error bars correspond to the 25% and the 75% percentiles, RTT = 18.2 sec, p = 5E-4

6.4 System Design Analysis

This section discusses the effects of both the packet loss probability (due to ARQ failure) and the of the timeout value due to RTT variations. Then, ARQ effect on the average RTT value is discussed.

6.4.1 The Effects of the Packet Loss Probability and the Timeout Value

In this section, the effects of the wireless errors and the impact of the timeout value are studied. It should be emphasized that in this study a wireless error refers to the errors that the physical layer ARQ could not handle. The effect of such errors is still appreciable in some cases even though they do not result in timeouts. This is due to the fact that a wireless error leads to the halving of the TCP congestion window. This effect is amplified for low bandwidth connections such as Iridium as it takes many (long time) RTTs before the connection reaches the maximum window limit again. For example, this time may be of the order of 30-40 seconds for an eight-modem connection (for a single packet loss). Frequent losses result in throughput impairment. Thus, the importance of an efficient physical layer ARQ that minimizes wireless errors as much as possible is emphasized.

Since the effect of call drops is significant when working under a long delay network, the parameters for Inmarsat GEO satellite network were used [36]. The number of modems in the bundle =4, file size =100MB, the per link bandwidth=128Kbps, RTT=0.61s (considering ARQ effect [5]), MSS=1KB and W_{max}=40KB. The range of

wireless errors with loss rates of (0.05% - 0.15%) is investigated. Figure 6.5 shows the effect of the packet loss probability on the performance [low values of p are considered because the proposed latency model assumes that losses are very low due to ARQ and no timeouts occur]. It is clear that a slight increase of the packet loss probability (over the studied range) results in approximately 25-minute increase in the transfer time. This emphasizes the importance of an efficient physical layer ARQ that minimizes wireless errors as much as possible.

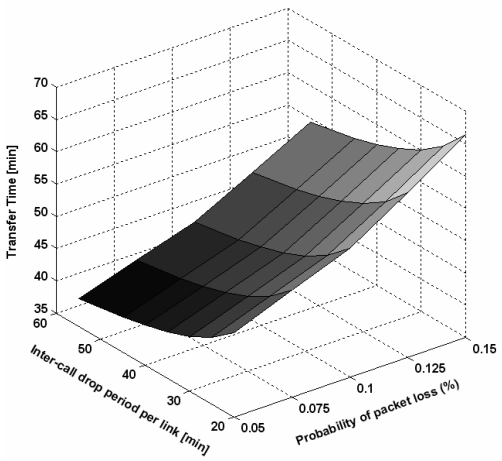

Figure 6.5: Effect of wireless errors on an Inmarsat MLPPP connection [Number of Modems = 4]

The timeout value is a significant factor that determines the impact of a call drop on the performance of a TCP flow. As discussed in Section 4.1.5, the initial timeout value is given by: $Timeout = Estimated\_RTT + 4 \times Deviation$. Thus in a connection with a significant RTT variance, the deviation term will dominate. In the case of point-to-point connections, the key reasons for the RTT variations are the physical layer ARQ and the MLPPP queuing. For example, a MLPPP packet might undergo

multiple ARQ retransmissions before it is received by the MLPPP layer. The worst case is when the MLPPP connection is running in the fragmentation mode, the failure of the ARQ on one link to transmit the MLPPP fragment results in discarding all the MLPPP fragments for that packet from all the other links. Thus, the timeout value is increased due to the significant RTT variations. As a reminder, it is assumed here that ARQ does not interfere with TCP operation.

Using the developed model (eq. 36), the effect of the timeout value has been studied for the same Inmarsat parameters in the range from 60 to 120 seconds as shown in Figure 6.6. The timeout value becomes an important factor when the call drop rate is high. This is due to the fact that frequent long timeouts will result in severe degradation of the TCP performance. Again, this adds another factor to consider when designing an ARQ algorithm. Moreover, this might be a reasonable motivation to optimize the current round-and-robin MLPPP frame distribution (i.e., on the modems) algorithm. Good ARQ and MLPPP frame distribution algorithms must always keep the RTT variance as low as possible in order to keep the value of the timeout as low as possible.

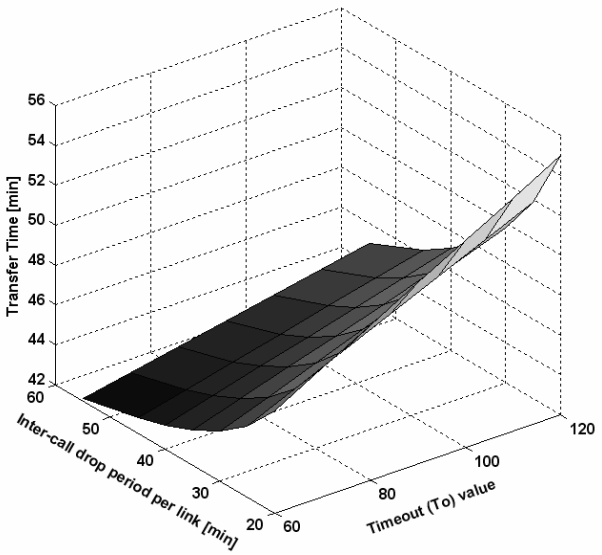

Figure 6.6: Effect of the timeout value on an Inmarsat MLPPP connection [Number of Modems = 4]

Finally, the results of the developed model suggest a potential advantage of modifying the operating system's kernel in a way that the kernel generates a triple acknowledgement once a call drop is detected. This instructs TCP to switch to the recovery mode instead of timing out. Thus, in this way, a call drop will only trigger a packet loss instead of resulting in a timeout. It is clear that implementing this technique requires that the kernel and/or the MLPPP drivers be modified to keep track of the received TCP acknowledgements. Consequently, the proposed model collapses back into Padhye's model [28] with the packet loss rate (p) approximately equal to the call dropping rate.

6.4.2 The Effect of the ARQ Operation on RTT

This section discusses the effect of ARQ on the performance of TCP, specifically, the impact of the ARQ operation on the TCP transfer latency. As discussed in Section 4.1.6, RTT varies due to ARQ variations (see Figure 4.6). The mean RTT is a function of the physical channel [25]. This can be explained by the fact that the number of retransmissions needed to deliver ARQ frames vary with the SNR seen at the receiver side. The physical channel state may vary with the mobile speed (m/s) and the data rate [27]. RTT models are of significant importance if one needs to develop short transfer models. Since the mean RTT is higher for wireless links, RTT models may be used to detect wireless links. ARQ must be designed such that it does not lead to sudden variations in RTT as this may result in unnecessary timeouts at the TCP layer [25].

In the future wireless broadband standards, such as in WiMax, the user terminal may change its data-rate according to the wireless channel conditions. Allowing variable mobile speeds and variable data rates would affect the ARQ performance and eventually would impact the observed RTT. The physical channel multi-path profile plays an important role in determining the effect of the ARQ on the RTT. The ARQ effect on the RTT may be described in terms of the mean RTT and the RTT variance. Such observations open the scope for a potential area of future work in this research.

Chapter 7. Software Design for the Iridium Link

Inverse-multiplexing multiple Iridium connections into a single MLPPP connection was first tested in Greenland in 2003 [4]. The connectivity between the laboratory and the field was established using 4 Iridium modems providing an aggregate rate of 9.6 kbps. In order to achieve higher data rates over the Iridium network, one needs to use more modems. The target of this work is to establish connectivity over eight modems. The original software developed in [4] was based on simple UNIX scripts. The software required several code modifications to perform trivial tasks such as changing the order of dialing the modems or even changing their numbers. Moreover, changing MLPPP parameters such as the LCP-echo-interval required changing all modem configuration files manually. In addition, the system lacked a logging mechanism which is vital for system administrators to keep track of the system operation events such as call drops and number of the call-establishment failures. Finally, there was no easy way for the end user to check the current status of the system (i.e. the status of each link in the MLPPP bundle). It is clear that running the system using the software in [4] requires a very experienced user.

Thus, the goal for this effort is to support an average throughput of 16 to 20 kbps using eight modems by providing simple software that does not require experienced users to operate it. The software should also provide graphical means for monitoring

the status of the individual links (up, dropped, etc). Finally, the system should keep a log of all the events per modem in a common format that can be easily analyzed by any commercial tool. This chapter discusses the development of the Iridium link management software and explores the design of the main components of management software.

7.1 Introduction

As shown in Chapter 2, the interaction between the two ends of the system is accomplished by using MLPPP to inverse-multiplex the application traffic into eight Iridium links. Figure 7.1 provides a general view of the developed system.

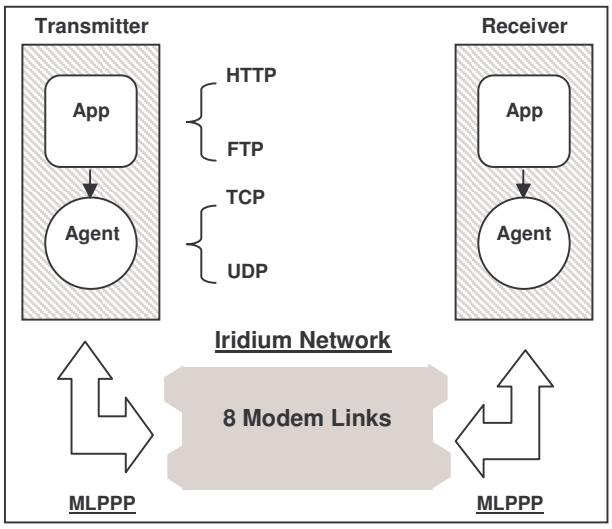

Figure 7.1: Overview of System Operation (adopted from [14])

Due to the nature of connectivity over multiple satellite links, links disappear due to call drops. Call drops might occur due to low signal strength, inter-satellite handover

failures, or due to any other signaling failure inside the satellite network. Since call drops are inevitable, a mechanism needs to be employed in order to bring those links up again once a call drop occurs. Moreover, the number of connection retrials in case of failed connection establishment attempts has to be limited in order to avoid disturbing the Iridium network.

The algorithm discussed in [4] can be extended as illustrated in Figure 7.2. In the following discussion, the terms modem and link are used interchangeably. Each physical modem starts in the connecting state. In this state, the modem dials the receiving end modem which is in the waiting state. If the MLPPP connection gets established by any modem, the link is considered to be in the connected state. The modem stays in the connected state until it experiences a call drop. Once a call drop occurs, the link moves into the dropped state and stays idle for a predefined period of time (T_d), once T_d seconds elapsed, the modem goes back into the connecting stage. If the modem does not connect (for example, if the signal strength was low, the satellite network was busy, etc), then it moves to the failed state and remains at this state for (T_f) seconds. Then, it goes back to the connecting state. If the maximum number of connection attempts (Max_Attempts) has been exceeded, the modem moves from the failed state to the blocked state. The modem stays idle (in the blocked state) for a relatively long period of time (approximately for one hour) before it goes back to the connecting state.

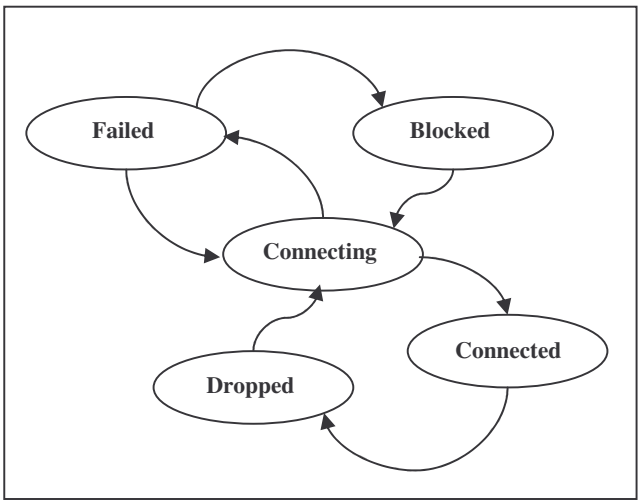

Figure 7.2: The State Diagram of an Iridium Modem

7.2 Link Management Software (KUMICS)

The proposed link management software, Kansas University Multilink Iridium Communication System (KUMICS), consists of three primary modules that interact with each other. Figure 7.3 illustrates the architecture of the KUMICS software. The number of modems supported by KUMICS ranges from 1 to 8 modems. The modems pool block represents the physical modems. The control software module handles the MLPPP bundling and link management. The graphical user interface (GUI) module provides a user friendly way to control the communication link. Each modem has a profile that contains it characteristics: modem name, number to dial, serial port, log file, etc. These files are stored in text formatted files in the modem profiles database. The connection database module represents the format and the structure of link states.

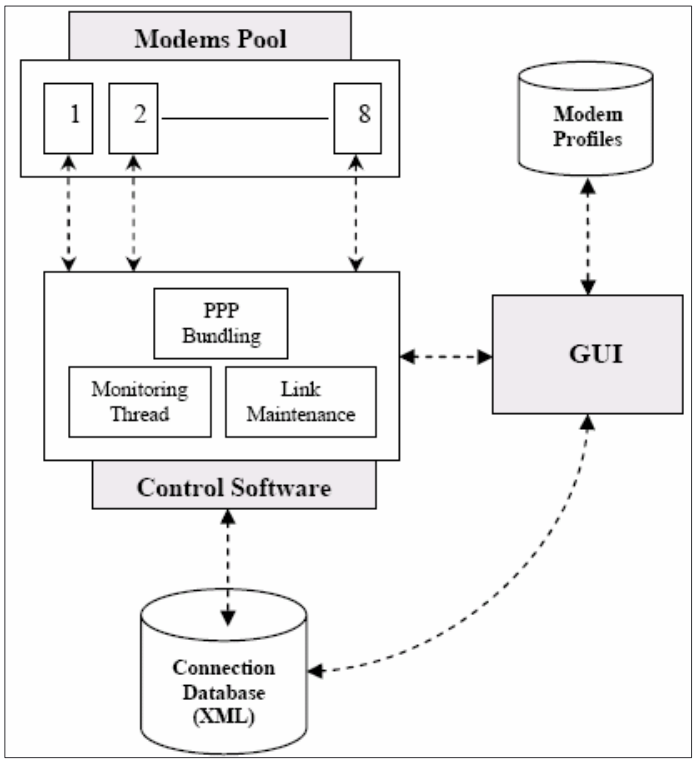

Figure 7.3: KUMICS Design Architecture

The connection database is stored in XML format. XML is a standard format that can be easily parsed and manipulated by commercial database engines (for example MS-SQL Server 2003 and MySQL). The XML schema of the database is shown in Figure 7.4. The heading of the document contains the connection start time (unified for all modems), modem name, modem index. The modem index is a sequence number used to identify the modem for a given connection. The index with value of one indicates the primary modem as will be discussed later.

```
<?xml version="1.0" encoding="utf-16"?>
<xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="connection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="modem name" type="xs:string" />
        <xs:element name="modem index" type="xs:int" />
        <xs:element name="connection_time" nillable="true"
type="xs:string" />
        <xs:element minOccurs="0" maxOccurs="unbounded"
name="entry">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="entry_id" type="xs:int" />
              <xs:element minOccurs="0" maxOccurs="unbounded"
name="event">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="state" type="xs:string" />
                    <xs:element name="time" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 7.4: XML Schema of the Connection Database

7.3 Control Module Details

The operation of the current implementation of MLPPP for Linux (pppd 2.4.3) is illustrated in Figure 7.5. The first modem that makes a connection creates the bundle and keeps track of who joins the bundle. In other words, the first connection process plays three roles: monitoring the PPP interface, creating the PPP bundle and

managing the connection of the first modem. The primary modem process keeps track of the MLPPP member links in the form of a small database.

The flow chart of the control software module, which controls the operation of multiple links and ensures a proper operation of the MLPPP bundle, is illustrated in Figure 7.6. The main process starts by selecting a primary modem. KUMICS selects the modem with the best call-drops history to be the primary modem. Afterwards, the primary modem is dialed. If the connection is successful, the secondary modems processes are spawned. If the maximum number of connection retrials for the primary modem is exceeded, the corresponding modem is blocked for N minutes and another modem is selected.

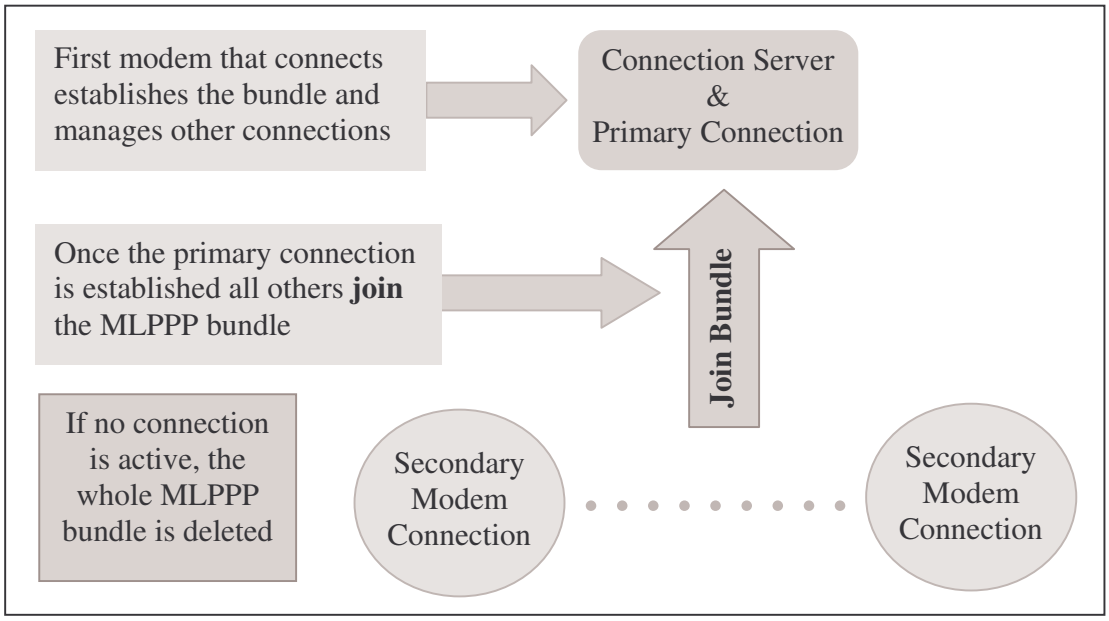

Figure 7.5: Operation of the PPP Daemon

Once the secondary modem processes are spawned, the main process enters the monitoring state. In this state, the main process watches the primary connection and at the same time monitors the MLPPP bundle. If the primary link drops, it reports that to the connection status database. If the bundle disappears (which happens if and only if all modems drop at the same time), the main process performs connection clean up and starts a new connection. It should be emphasized that the term “connection”, refers to the MLPPP connection once started by a primary link. The connection ends when the bundle disappears. Each connection is stored in the database separately.

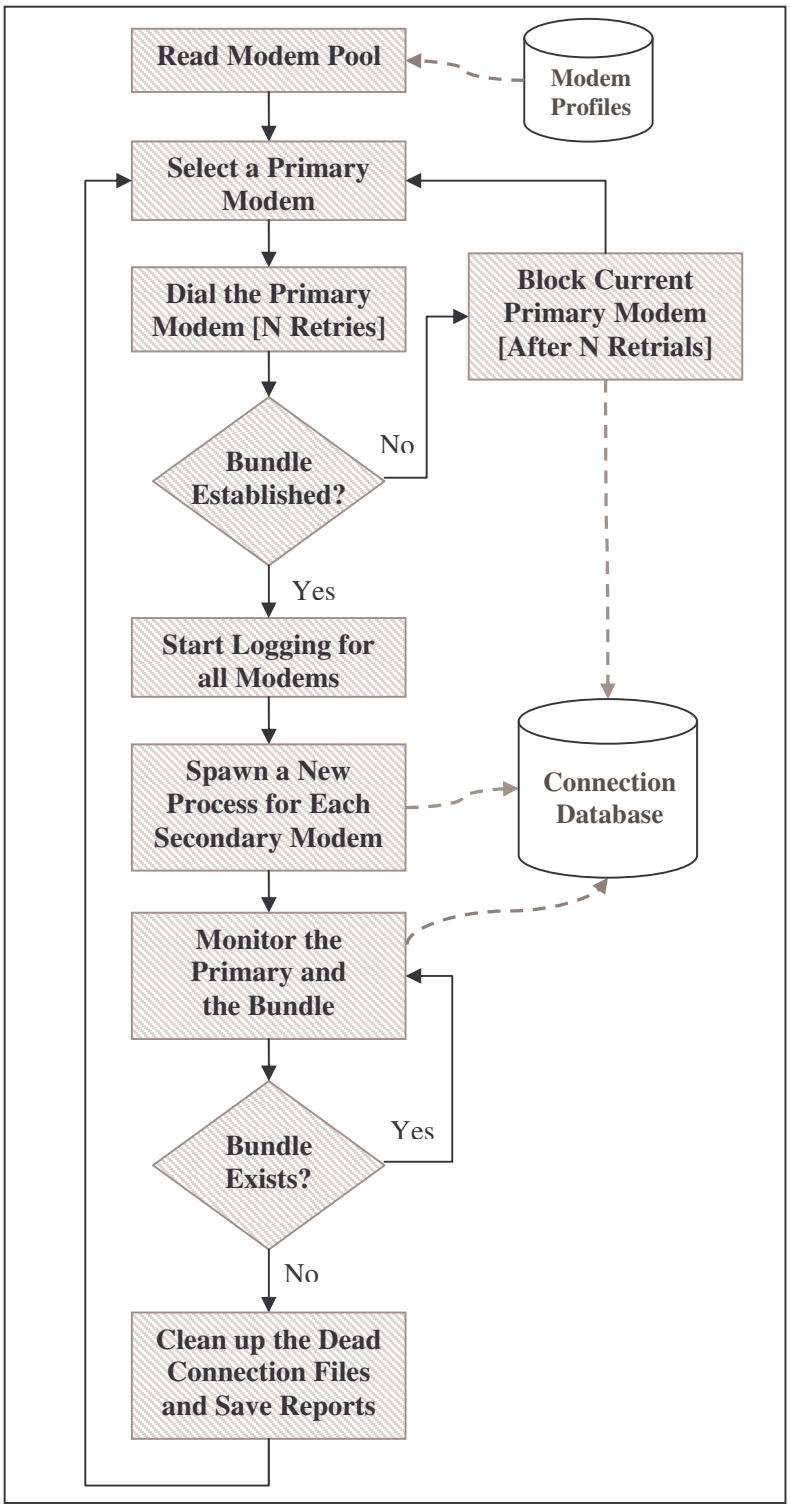

Figure 7.6: Control Software Management Algorithm

Figure 7.7 shows the secondary modems' connection algorithm. The secondary modems get connected in a similar faction to how the primary modem gets connected. The main difference here, is that if the primary link is lost it never gets connected again until the whole bundle drops (i.e, a new connection is re-established). This is a known problem in PPPD 2.3.4 package for Linux.

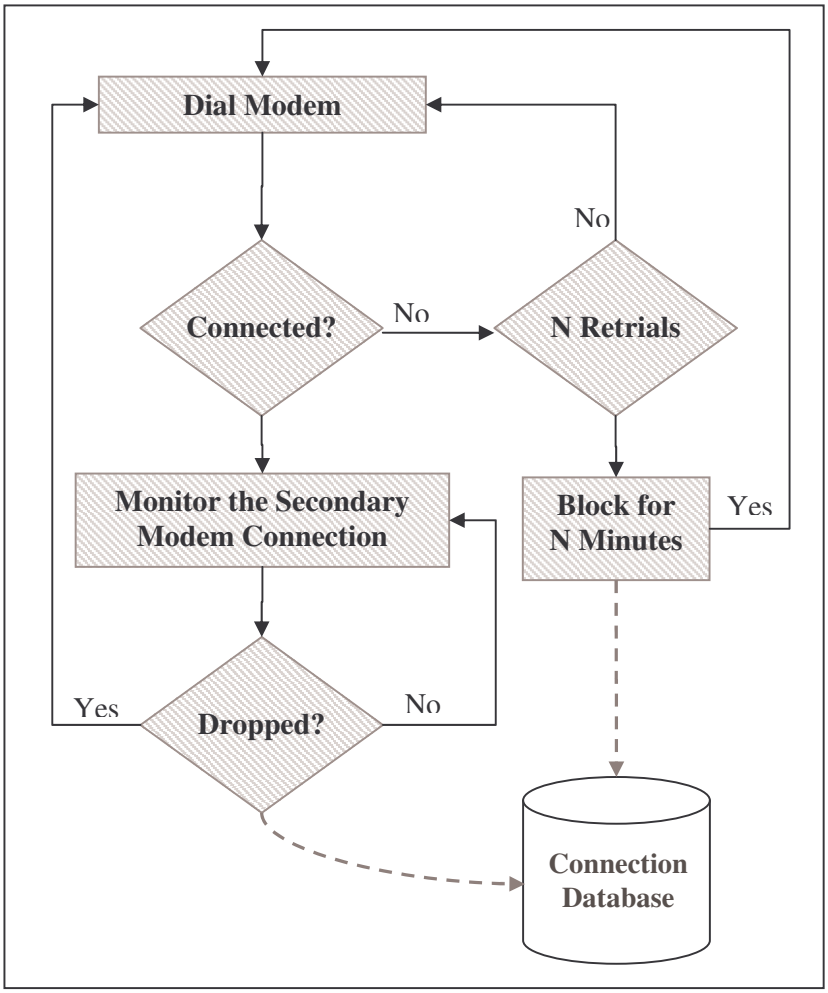

Figure 7.7: Secondary Modems' Connection Algorithm

7.4 Graphical User Interface Design Details

The graphical user interface (GUI), shown in Figure 7.8, handles several tasks that are required before loading the control module discussed in Section 7.3. After making the proper hardware connections to the corresponding serial ports, the GUI allows the user to perform configuration on a per-modem basis. The information required (see Figure 7.9) for each modem is: the modem name, the serial port that the modem connects to, the file name to be used for PPPD for this modem profile, and finally the phone number to dial.

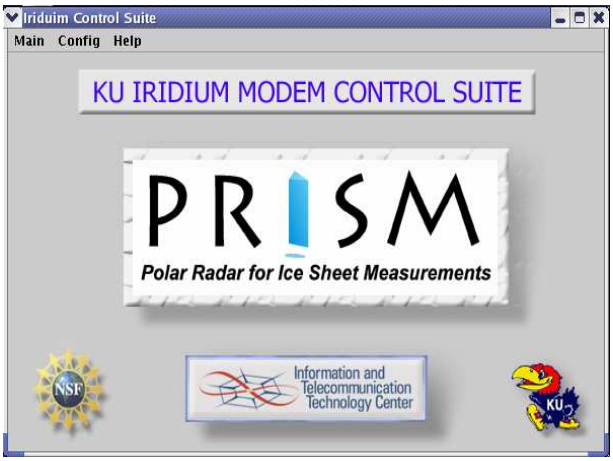

Figure 7.8: Main Window of the KUMICS' GUI

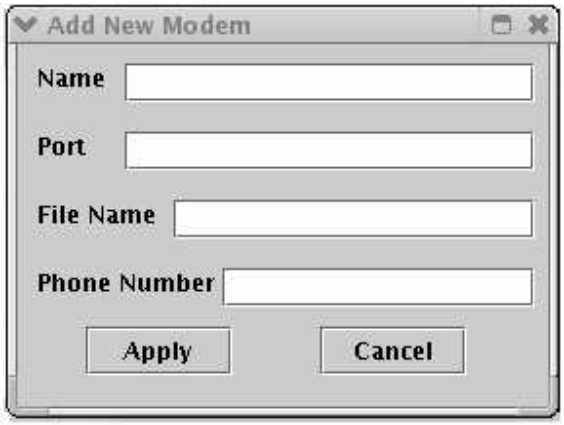

Figure 7.9: Creating a New Modem Profile

The next step is to select the modems to be used for the current connection as shown in Figure 7.10. KUMICS allows the user to have multiple modem profiles, thus, it is necessary to specify which modems' profiles to be used for the MLPPP connection.

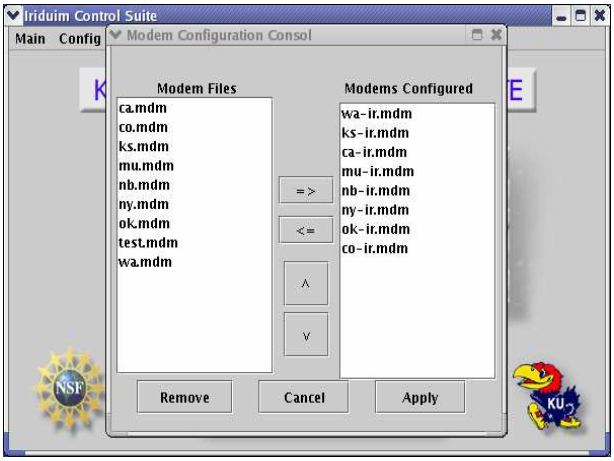

Figure 7.10: Currently Configured Modems

Next, one specifies the parameters to be used for the current connection: PPP username, PPP password, state of compression on the MLPPP layer, the state of the

Data After Voice (DAV) Mode, the number of modems of the currently configured modems to be used (this number is set to be the same as the number of the configured modems by default), and the MLPPP parameters (described in Chapter 3): PAP-Restart, LCP-Restart, LCP Max Configure, Keep-alive Interval, Keep-alive Retrials and Connect Delay.

Once all configurations have been created, one saves the current configurations in a connection profile file that can be retrieved later. The final step is to compile the current settings into PPPD configuration files. The GUI module contains a compiler sub-module that compiles user configurations using a pre-defined template into PPPD configuration files.

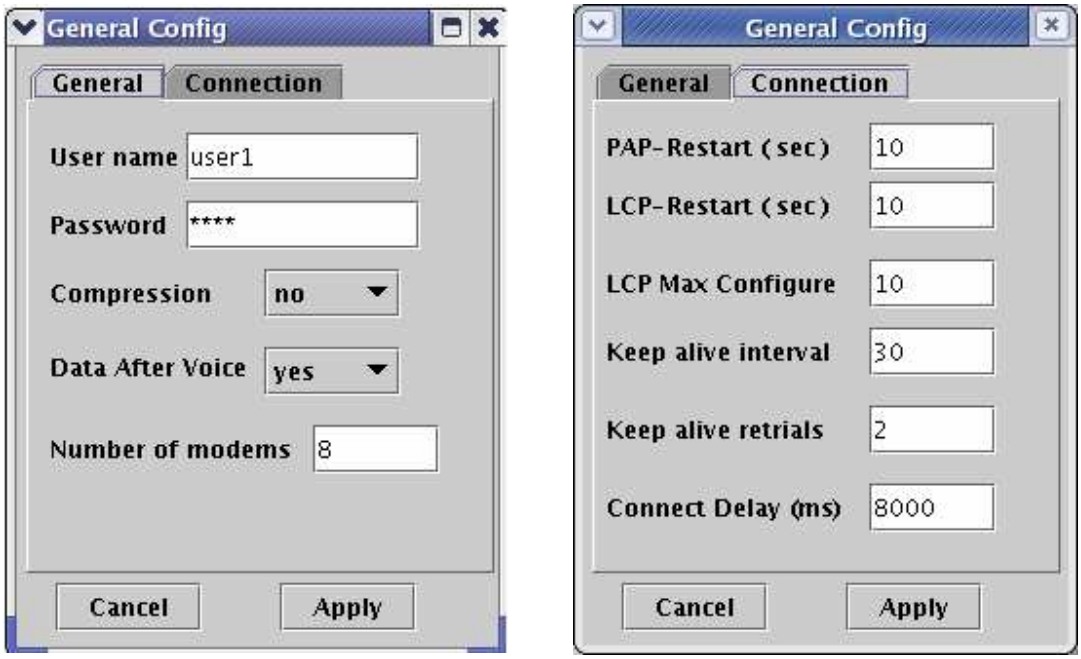

Figure 7.11: MLPPP Connection General

7.5 A Complete Integrated Image of KUMICS

Once the current configuration is compiled into working PPPD configuration files, one can start an Iridium connection by a simple click on the connect menu. In the background, the GUI module creates an instance for each modem and shows it on the screen. For the sake of this discussion, we will refer to these objects as the *modem objects*. The main function of these instances is to keep track of the current modem status and visually report it to the user using different colors for each state. Once the modem objects are created, the GUI launches a monitoring thread that refreshes the modem objects just created every few seconds in order to reflect the status of the control module. This information is passed in temporary files that are read by the modem objects once refreshed by the monitoring thread. Figure 7.12 shows a screen shot of the connection monitoring window that KUMICS provides where each state is represented by a different color. Modems were named after US state names.

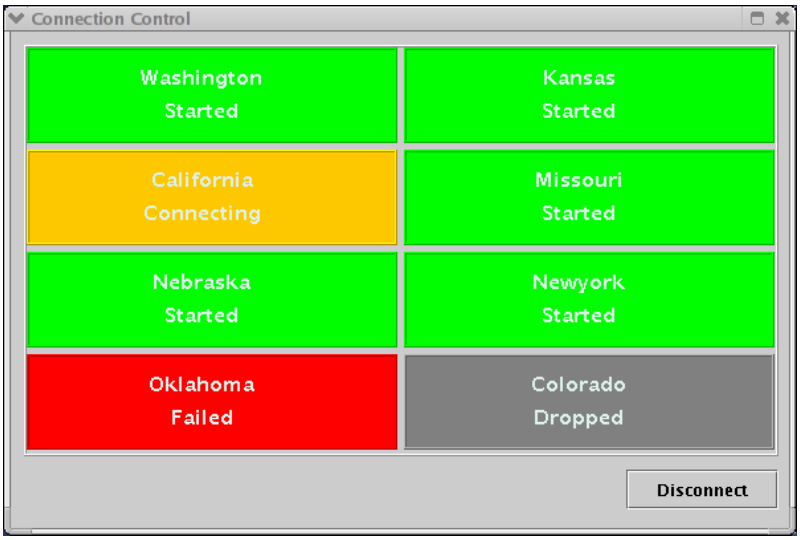

Figure 7.12: A Screen Shot of a Connection Monitoring Window

The control module, as discussed in Section 7.3, performs MLPPP bundle management, handles the reporting to the database module, and updates the status of the GUI module. Once disconnected, the database reports might be processed offline to extract some performance metrics such as the number of call drops and their distribution. It is noteworthy to state that the control software also keeps track of the average time between call drops for each modem and uses this information in the initial primary modem selection. This information can be a valuable resource that provides an instant metric for system administrators about the performance of each modem. The database reports provide the time at which each state is encountered. Figure 7.13 summarizes the integration of the three modules in KUMICS.

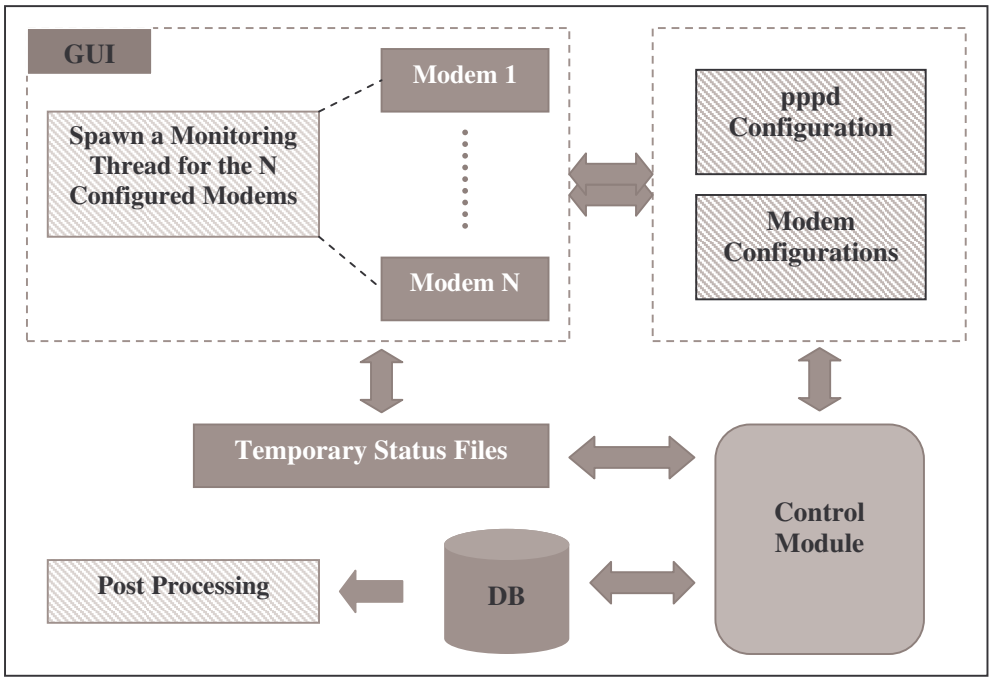

Figure 7.13: KUMICS' Modules Interaction for a Typical Connection

7.6 Experiences with KUMICS

KUMICS was first installed on the Iridium communication system as part of the scientific research performed in Greenland in the summer of 2004. The system provided a reliable MLPPP connection formed by eight Iridium modems. The system delivered full throughput (i.e., when all modems are connected) 87% of the time and maintained 97% of connectivity with at least one modem [37] as shown in Figure 7.14. In Figure 7.14, the term *primary call drops* refer to the times at which connectivity is completely lost (i.e., no modem is up).

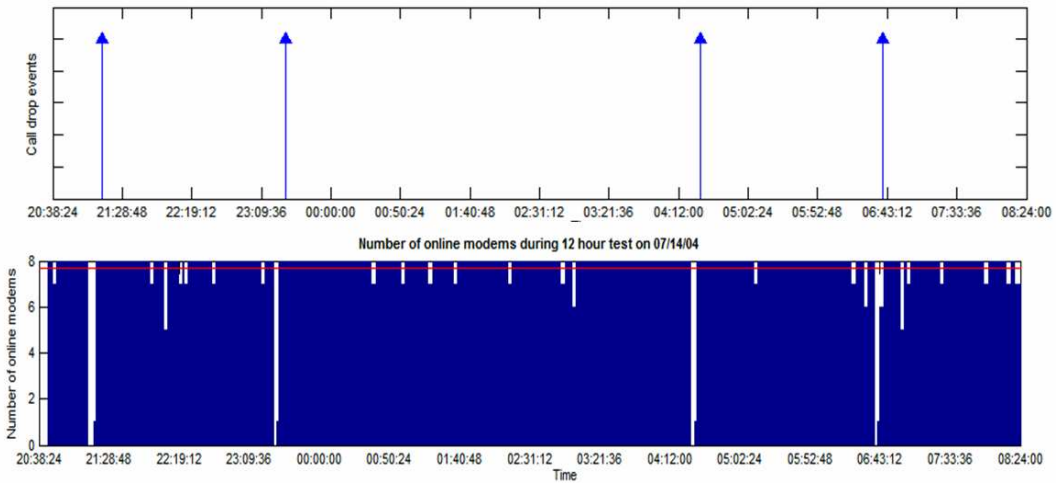

Figure 7.14: Reliability of the Iridium System
[Call drop pattern for an 8x8 Iridium test for 12 hrs,
Average time between call drops = 3 hrs]

The system was able to support the desired throughput (the product of the number of modems and the single modem rate of 2.4 kbps) as shown in Figure 7.15. More information about the system performance is available in [37].

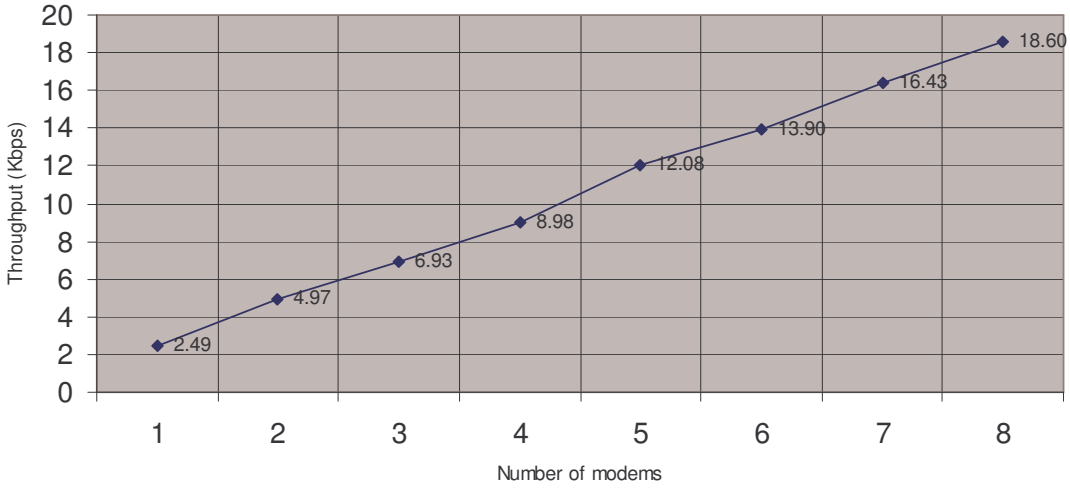

Figure 7.15: Throughput as a function of the number of modems

In summary, this chapter discussed the main elements of the link management software for MLPPP systems. The developed software package, KUMICS, consists of a graphical user interface (GUI), a link control module, and a link monitoring and reporting system. Tests showed that the system had at least one connected modem 97% of the time with a maximum throughput (8 modems) of 18.6 kbps.

Conclusions

Many researchers have studied TCP behavior for various applications and in various environments. For instance, models were provided for long transfers such as FTP transfers over wireless cellular networks. Recently, efforts resulted in successful utilization of the MLPPP protocol to provide connectivity in Polar regions, which is covered by 2.4 kbps Iridium satellite connections, by inverse-multiplexing eight Iridium links into one virtual MLPPP connection. It was observed that a call-drop on an individual Iridium link results in a TCP timeout.

In this thesis, we studied the time difference between call drops on a MLPPP connection and suggested that the call drop process can be modeled by a Poisson process. We used this knowledge to extend the TCP transfer latency model in [15] to capture the effect of call drops for TCP connections over long delay MLPPP links. Then, we used the Iridium network to validate the proposed TCP transfer latency model. We also confirmed that TCP performance can be enhanced by employing an efficient physical layer ARQ in order to minimize the number of wireless packet errors. Finally, we successfully designed, implemented and tested in the field a user friendly MLPPP link management software package. The developed system provided full connectivity 87% of the time and connectivity with at least one modem 97% of the time.

Future Work

The presented work in this thesis can be extended to predict the performance of TCP over multilink-PPP cellular systems. In cellular systems call drops are widely affected by handovers. Thus, a new call drop model needs to be developed. This effect may be challenging in the third generation (3G) cellular systems which implement Mobile IP. Handover failures modeling in the case of 3G systems with Mobile IP is more involved than in the case with handover modeling for circuit switched cellular systems like in GSM and IS-95. Nevertheless, once the call drop model is developed, the expected value between call drops is estimated from the probability density function of the time between call drops for the whole MLPPP bundle. One should always remember that the mobile speed plays an important role in the performance of the handover algorithms. The impact of the mobile speed on the operation of the physical layer ARQ needs to be investigated as the multi-path profile may change depending on the mobile speed.

The effect of the MLPPP queuing on the average and the variance of the RTT can be studied and incorporated to the developed model (eq. 36). The queuing algorithm used in the MLPPP implementation needs to be investigated and its performance needs to be evaluated. Moreover, choosing the size of the maximum received reconstructed unit (MRRU) for MLPPP needs to be studied. The effect of the size of

the MRRU can be incorporated in the TCP throughput prediction, (eq. 36), provided in this thesis.

Moreover, the operating system kernel can be modified in a way that it generates a triple acknowledgement once a call drop is detected. This instructs TCP to switch to the recovery mode instead of timing out. Thus, in this way, a call drop will only signal a packet loss instead of causing a timeout. Implementing this technique requires that the kernel and/or the MLPPP drivers keep track of the received TCP acknowledgements.

Furthermore, the transient properties of RTT for MLPPP connection should be addressed when developing models for short TCP transfers such as web-browsing. Although this aspect does not have significant impact on the predictions for long TCP transfers, it affects the accuracy of the predictions for short transfers. This is due to the fact that short transfers do not last for a long time and using the long term average of the RTT does not lead to acceptable predictions.

Finally, software enhancements for the link management software need to be performed whenever there is a new release of the MLPPP drivers' package [13] or a new feature offered by the satellite modems. The software might also be enhanced by supporting remote administration where users can monitor the status of the system using a remote terminal avoiding using complex Linux commands. Moreover, the system can be enhanced by providing a standby communications line (regular

telephone, cellular, or satellite) that is used to make a phone call to the system administrators in case of severe problems.

References

[1] “RFC 793 – Transmission Control Protocol”, 1981, web resource, available at: http://www.freesoft.org/CIE/RFC/793/index.htm.

[2] A. Jamalipour, “Low Earth Orbital Satellites for Personal Communication Networks”, Artech House, Boston.London, 1998.

[3] “About Inmarsat”, web resource at: http://about.inmarsat.com/satellites.aspx.

[4] A. Mohammad, “Multi-Link Iridium Satellite Data Communication System”, Master of Science thesis, University of Kansas, 2004.

[5] Y. Zhang, “Internetworking and Computing over Satellite Networks”, Kluwer Academic Publishers, 2003.

[6] R. Nelson, “Iridium: From Concept to Reality”, web resource at: http://www.aticourses.com/news/iridium.htm.

[7] “Iridium Satellite System Website”, web resource at: http://www.iridium.com/corp/iri_corp-understand.asp.

[8] S. Pratt, “An Operational and Performance Overview of the Iridium Low Earth Orbit Satellite system”, IEEE Communication Surveys, http://www.comsoc.org/pubs/surveys/, 2nd Quarter 1999, pp. 1-15.

[9] “RFC 1661 – The Point to Point Protocol (PPP)”, 94 at: http://www.faqs.org/rfcs/rfc1661.html.

[10] “MLPPP Introduction”, web resource available at: http://www.cisco.com.

[11] “RFC 1990 – The PPP Multilink Protocol (MP)”, 96 at: http://www.faqs.org/rfcs/rfc1990.html.

[12] P. Mackerras, “PPP Generic Driver and Channel Interface”, 2002, web resource available at: http://www.ibiblio.org/peanut/Kernel-2.6.10/networking/ppp_generic.txt.

[13] “PPPD 2.4.3 Software”, web resource available at: http://ppp.samba.org/.

[14] A. Mohammad, V. Frost, S. Zaghloul, “Overview, Performance and Reliability from summer 2004 SUMMIT, Greenland Field Experiments July 14-July 25, 2004”, web resource available at: http://www.ittc.ku.edu/~frost/KU_IRIDIUM_GREENLAND-FIELD-TESTS_2004.ppt.

[15] J. Padhye and V. Firoiu and D. Towsley and J. Krusoe, “Modeling TCP Throughput: A Simple Model and its Empirical Validation”, Proceedings of the ACM SIGCOMM '98, pp. 303-314, 1998.

[16] V. Jacobson, “Congestion Avoidance and Control”, Proc. ACM SIGCOMM, Stanford, CA, Aug. 1988, pp. 314–329.

[17] “RFC 2001 - TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, web resource available at: http://www.faqs.org/rfcs/rfc2001.html.

[18] B. Sikdar, S. Kalyanaraman and K. Vastola, “Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK”,IEEE/ACM Transactions on Networking, pp. 959-971,vol. 11, no. 6, Dec. 2003.

[19] V. Jacobson, “Modified TCP Congestion Avoidance Algorithm”, web resource available at: ftp://ftp.ee. lbl.gov/email/vanj.90apr30.txt.

[20] “New Congestion Avoidance algorithm”, web resource available at: http://www.opalsoft.net/qos/TCP-30.htm.

[21] K. Fall and S. Floyd, “Simulation-based comparisons of TAHOE, RENO, and SACK TCP,” ACM Comput. Commun. Rev., vol. 26, no. 3, pp. 5–21, 1996.

[22] W. Stevens, “TCP/IP Illustrated, Volume 1”, Addison-Wesley, 1994.

[23] N. Cardwell, S. Savage, and T. Anderson, “Modeling the performance of short TCP connections”, Technical Report, Washington University, 1998.

[24] L. Peterson and B. Davie, “Computer Networks: A Systems Approach”, 3rd Ed, Morgan Kaufmann, 2003.

[25] A-F Canton, T-Chaled, “End-to-end Reliability in UMTS:TCP over ARQ”, Globecom’2001, San Antonio, Nov 2001.

[26] D. Bertsekas and R. Gallager, “Data Networks”, 2nd Ed, Prentice Hall, 1992.

[27] M. Zorzi, R. R. Rao, L. B. Milstein, “Error Statistics in Data Transmission over Fading Channels”, IEEE Transactions on Communications, Vol. 46, No. 11, November 1998, pp. 1468-1476.

[28] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, “Modeling TCP Reno performance: A simple model and its empirical validation”, IEEE/ACM Trans. on Networking, vol. 8, no. 2, pp. 133-145, April 2000.

[29] N. Cardwell, S. Savage, and T. Anderson, “Modeling TCP latency”, in Proc. IEEE INFOCOM, Tel Aviv, Israel, Mar. 2000, pp. 1742–1751.

[30] B. Sikdar, S. Kalyanaraman and K. Vastola, “Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK”,IEEE/ACM Transactions on Networking, pp. 959-971,vol. 11, no. 6, Dec. 2003.

[31] Mark Allman and Vern Paxson, “On estimating end-to-end network path properties”, SIGCOMM ’99, August 1999.

[32] Yujian Peter Li, “Modeling Web/TCP Transfer Latency”, (Thesis) , Calgary, Alberta January 2004, web resource available at: http://pages.cpsc.ucalgary.ca/~yli/thesis.pdf.

[33] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, Lawrence Berkeley National Laboratory, web resource available at: http://www.icir.org/floyd/papers/sacks.pdf.

[34] “ISU AT Command Reference”, model: A3LA-D Iridium modems, version 2.1, web resource available at: ftp://ftp.nalresearch.com/Satellite%20Products/Standard%20Modems/A3LA-D/Release%203.2.3/A3LA-D%20Manuals/AT%20Command%20Reference%20V2.1.pdf.

[35] J. Devore, “Probability & Statistics for Engineering and the Sciences”, Brooks/Cole, 1982.

[36] M. Allman, C. Hayes, H. Kruse, S. Ostermann, “TCP Performance over Satellite Links”, 5th Intl Conference on Telecomm Systems, Tennessee, USA, ‘97.

[37] “ Experiences with 8 Modem Multi-Link Iridium Satellite Data Communication System-2004”, , web resource available at: http://www.ittc.ku.edu/~frost/KU_IRIDIUM_GREENLAND-FIELD-TESTS_2004.ppt.