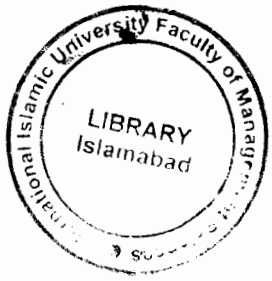
TCP PERFORMANCE USING SPLITTING OVER THE LEO SATELLITE LINK

Supervised by
Prof. Dr. Khalid Rashid
Dr. Tauseef-ur-Rehman



Developed by
Maleeha Saeed

Department of Computer Science,
International Islamic University, Islamabad.
(2004)

International Islamic University Islamabad

Department of Computer Science

Date: 23RD JUNE 2004

Final Approval

Acc. No. (FMS) T-1173

It is certified that we have read the project report submitted by Maleeha Saeed (85-CS/MS/02) and it is our judgment that this project is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for the MS Degree in Computer Science.

External Examiner
Dr. Mohammad Qasim Rind
Ex- Director General,
Establishment Division,
Government of Pakistan,
Islamabad, Pakistan.

Internal Examiner
Dr. Sikandar Hayat Khayal
Head,
Department of Computer Sciences,
International Islamic University,
Islamabad, Pakistan.

Supervisors
Prof. Dr. Khalid Rashid
Dean,
Faculty of Applied Sciences, &
Faculty of Management Sciences,
International Islamic University,
Islamabad, Pakistan.

Dr. Tauseef-ur-Rehman
Head,
Department of Telecommunication Engineering,
International Islamic University,
Islamabad, Pakistan.

Dedicated to

My Beloved Parents

A dissertation submitted to the

Department of Computer Science,

International Islamic University, Islamabad

as a partial fulfillment of the requirements

for the award of the degree of

MS of Computer Science

DECLARATION

We hereby declare that this software neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software and completed the report entirely on the basis of our personal efforts made under the sincere guidance of our teachers. If any part of the system is proved to be copied out or found to be reported, we shall stand by the consequences. No portion of the work presented in this report has been submitted in support of any application for another degree or qualification of this or any other university or institute of learning.

Maleeha Saeed
(85-CS/MS/02)

ACKNOWLEDGEMENT

We would like to take this opportunity to pay our humble gratitude to Almighty Allah, Who enabled us to complete this project. After that we would like to thank our helpful teachers, Prof. Dr. Khalid Rashid and Dr. Tauseef-ur-Rehman.

Maleeha Saeed
(85-CS/MS/02)

PROJECT IN BRIEF

Project Title:	TCP Performance Using Splitting Over the Satellite Link.
Objectives:	Check the of TCP Performance Using Splitting Over the Satellite Link.
Undertaken By:	Maleeha Saeed
Supervised By:	Dr. Khalid Rashid Dean Faculty of Applied Sciences, & Faculty of Management Sciences, International Islamic University, Islamabad, Pakistan. Dr. Tauseef-ur-Rehman Head, Department of Telecommunication Engineering, International Islamic University, Islamabad.
Tools And Technologies Used:	Network Simulator-2
Operating System:	Windows 9x, 2000, NT, XP
System Used:	x86 Family 6 Model 8 Stepping 10 AT/AT Compatible 122, 356 KB RAM.
Date Started:	1st September 2003
Date Completed:	May 2004

ABSTRACT

World wide usage of the internet is currently growing at a faster rate, resulting in a huge demand for the transmission of internet data via satellite. Satellite system features clearly shows that it can also used for mobile internet applications because it has remarkable ability to cover huge areas for data transmission. In other words satellite gives better approach for broadband transmission. To achieve this successful goal, these satellite systems must show their abilities for the internet facility with the unforgettable support of internet protocols e.g TCP. The failure of TCP is due to high error rate and high delays. Due to movement of LEO satellite system causes more errors. One way to control this problem is to divide TCP connections into segments, or splitting the connection. This paper takes above describe approach, splitting to explore the use of a TCP Proxy on board a satellite for the purpose of enhancing end-to-end TCP performance subdividing end-to-end application paths into separate TCP connections – uplink and downlink.

Maleeha Saeed
(85-CS/MS/02)

List of Contents

List of Figures

List of Abbreviations

TCP:	Transmission Control Protocol.
NS:	Network Simulator.
NS-2:	Network Simulator-2.
LEO:	Low Earth Orbit Satellite.
RTT:	Round Trip Time.
ACK:	Acknowledgement.
GEO:	Geo-Stationary Earth Orbit Satellite.
BER:	Bit Error Rate.
WISE:	Wireless IP Suite Enhancer.
WLP:	Wireless Link Protocol.
STCP:	Sharing TCP.
STP:	Satellite Transport Protocol.
TCPW:	TCP Westwood.
BWE:	Bandwidth Estimate.
AODV:	Ad-hoc On-Demand Distance Vector.
DSR:	Dynamic Source Routing.
MANET:	Mobile Ad Hoc Network.
TCP/IP:	Transmission Control Protocol/Internet Protocol.
ISL:	Inter Satellite Link.
GSL:	Ground to Satellite Link.
FACK:	Forward ACK.
SACK:	Selective ACKs.

Chapter 1

INTRODUCTION

1 INTRODUCTION

The idea of ubiquitous broadband access to the Internet continues to expand. Many people have grown to depend upon the internet and wish to extent this access to other areas of their lives. So satellite systems either currently operating or in various stages of development can easily fulfil the requirements. Because satellite has a number of important applications in both the commercial and military markets. A satellite has the unique advantage that it provides an instant communications infrastructure to almost anywhere in the world. And other advantages of satellite communications are natural broadcast capabilities, and the ability to reach remote and geographically adverse locations at relatively low cost.

In the TCP/IP suite, the connection-oriented transport protocol is the transmission control protocol (TCP) and the service it offers to users-through application protocols-is known as the reliable stream transport service. It provides a set of congestion control mechanisms to ensure the reliable delivery of data, and to adjust the data transmission according to network conditions.

The TCP/IP protocol suite that forms the basis of the Internet was designed to operate over an extremely large range of environments. But due to some characteristics of satellite performance of TCP is degraded. The large propagation delay characteristic of satellites can severely impact TCP performance. TCP connection experiences relatively frequent losses from link level errors, TCP performance suffers as a result.

Some solutions involve changes to the protocol mechanisms to accommodate the properties of satellite links. These modifications change the basic error-control and flow-control strategies to improve performance. Another class of solutions requires changes to the architecture of the network. In these cases, intermediaries perform processing on behalf of TCP endpoints to the greater benefit of performance. One such technique involves subdividing connections into terrestrial and space segments, or "splitting" the connection.

There are basic two ways to change the protocol mechanism. For improving end-to-end performance of TCP with the help of Splitting mechanism we can convert it into another version of TCP. Another way to improve end-to-end performance of TCP, simply convert TCP into another new protocol.

Improving end-to-end performance of TCP we use the concept of Splitting mechanism in LEO satellite system, which is our proposed idea. The existing protocol is related to GEO satellite system.

1.1 SATELLITE COMMUNICATION:

A satellite is a specialized wireless receiver/transmitter that is launched by a rocket and placed in orbit around the earth. Communications satellites have been around since 1958. A communication satellite is space craft that orbits the earth and relays messages, radio, telephone and television signals. Stations on the ground, called earth stations, transmit signals to the satellite, which the relays the signal to other earth stations.

The first artificial satellite, launched by Russia (then known as the Soviet Union) in the late 1950s, was about the size of a basketball. It did nothing but transmit a simple Morse code signal over and over. In contrast, modern satellites can receive and re-transmit thousands of signals simultaneously, from simple digital data to the most complex television programs [1].

The latter half of the 1990s has seen a resurgence of interest in satellite-based data networks. Satellite communication systems have long been one of the hallmarks of advanced communications technology, with their remarkable and distinctive ability to link most of the populated areas of the earth. Yet, until recently, the satellite communication industry had increasingly begun to look more like a dinosaur, with competition from fiber optic and terrestrial wireless networks steadily eating away at the industry's most profitable markets.

Worldwide usage of the Internet is currently growing at an exponential rate, resulting in a dramatic increase in the demand for the transmission of Internet data via satellite [2]. The primary differences between terrestrial and satellite connectivity are the link latency, error rate, and asymmetry.

Satellites are very useful for communication because of the following reasons. They have natural broadcast capability. They can reach geographically remote areas and places where lack terrestrial communication infrastructure. This also make them useful to reach mobile users [3].

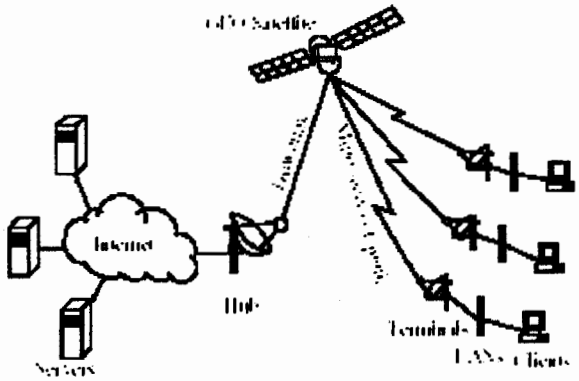


Figure 1.1 (Network Scenario)

1.2 LOW EARTH ORBIT SATELLITES (LEO):

LEOs are usually located from 700km from the earth to 1500 or 2000km. The low altitude of the orbit makes the transmission time for a link between a ground station and the satellite much shorter than it is for GEOstationary satellites (GEOs). Another advantage of low altitude is that less power is needed for the connection which in turn means that antennas used at the ground stations can be very small. So it is possible to use a device not bigger than an cellular phone which is quite important if the network should be used for things like mobile telephony.

The velocity of low earth orbit satellites moving around the globe is about 25000km/h, so the orbital period is around 100min, depending on their height. This also means that a

LEO satellite is visible from a single point on the earth only for a short period of time (3-12min). To achieve global coverage a high number of satellites is required. Most LEO satellite systems use polar orbits, each with a certain number of satellites. Due to the earth's rotation beneath them, every satellite covers the whole surface within several orbital cycles.

The famous global LEO satellite system working, Iridium, uses 66 satellites (it's name is derived from element 77, which is iridium, since 77 satellites were planned originally).

Another global low earth orbit satellite system, is Teledesic. This was planned to consist of 840 satellites, later this number was reduced to 288. Teledesic was also have the capability to route network data.

Satellites in low orbits can only cover a small earth surface area at a time, if a minimum elevation angle should be ensured. The elevation angle is the angle under which the satellite can be seen from the earth surface. The bigger the angle (maximum is 90 degrees in the zenith) the lower is the error rate during data transfer due to dust, rain and longer transmission paths through the atmosphere.

The area which is covered by a single satellite with certain minimum elevation angle taken in consideration is called its footprint. The footprint between two adjacent satellites in each orbit is overlapping, and so are the footprints of two adjacent orbits. If one inscribes a hexagon into each footprint, then you speak of the effective footprint of the satellite and you can cover the surface of the earth with them without any gaps.

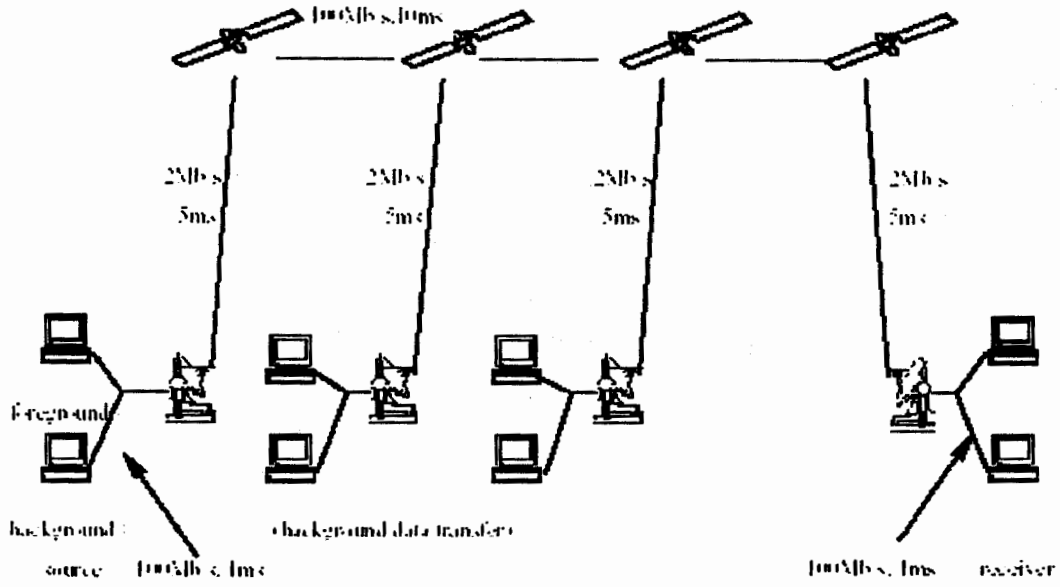


Figure 1.2 (LEO Satellite System)

There are two classes of low earth orbit satellites:

1.2.1 LITTLE LEOS:

Which are used for non-real time communications, sometimes also referred to as real enough-time service. Applications for these are data transfer like paging or everything that can be done by electronic mail. Little LEO are quite small with a weight of 50-100kg.

1.2.2 BIG LEOS:

Where as big LEOs provide voice transfer (cellular telephony) or even networking.
It contains the weight of up to 500kg and a diameter of not more then a few meters.

1.3 TRANSMISSION CONTROL PROTOCOL (TCP):

The TCP/IP protocol suite that forms the basis of the Internet was designed to operate over an extremely large range of environments [2].

In the TCP/IP suite, the connection-oriented transport protocol is the transmission control protocol (TCP) and the service it offers to users-through application protocols-is known as the reliable stream transport service. It provides a set of congestion control mechanisms to ensure the reliable delivery of data, and to adjust the data transmission according to network conditions.

1.3.1 TCP ENHANCEMENTS:

In the last ten years, a large number of proposed enhancements and changes to the basic TCP structure have been put forward. Each of these offers its specific features. Some involves modifications and additions to the protocol itself, while others involve implementation changes. At the same time, the Internet community remains sensitive to the broad effect that changes can carry, particularly with respect to congestion control. In spite of the large number of proposals, very few TCP varieties have gained broad acceptance. Some of more commonly known include Tahoe, Reno, Vegas and SACK [4].

The traditional Tahoe version of TCP includes the basic slow-start and congestion avoidance mechanisms, and incorporates the Fast Retransmit method of avoiding time-out by retransmitting data upon receiving a duplicate acknowledgement. In case a packet is lost and the TCP sender does not receive the ACK, it times out and retransmits the packet, reduces its congestion window to one and enters slow-start. If three or more duplicate ACKs are received in a row, it is the strongly indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire [4].

In most respects similar to Tahoe, the TCP Reno implementation incorporates the Fast Recovery mechanism. Fast Recovery operates within congestion avoidance instead of going into slow-start in response to the Fast Retransmit condition. Thus, the sender halves the congestion window rather than sending it back to one, as in TCP Tahoe, and thereafter increases occur incrementally. In fact, since the receiver can only generate the duplicate ACK when another segment is received, there must still be data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow-start [4].

A variant of Reno, called New Reno, modifies its behaviour when receiving new ACKs in the Fast Recovery phase. In order to exit Fast Recovery, the sender must receive an ACK for the highest sequence number sent [4].

TCP Vegas takes a different approach by beginning retransmit early when the time difference between a send and its corresponding ACK exceeds the time-out value. Vegas implementations also include a congestion control mechanism that will increase/decrease the congestion window relative to the difference between a sampled throughput measurement and the value of the window size normalized by the round-trip time measurement [4].

Other schemes like TCP SACK and FACK show extremely good performance on very lossy links. In TCP SACK, when the receiver holds non-contiguous data, it sends duplicate ACKs bearing SACK options to inform the sender which segments have been correctly received. TCP FACK uses the additional information provided by the SACK option to keep an explicit measure of outstanding data in the network and is able to regulate it to be within one segment size of the congestion window [4].

1.4 TCP SPLITTING:

TCP splitting is an approach that uses a gateway at the periphery of the satellite network to convert TCP traffic into an intermediate protocol that is well suited for the satellite

environment. On the other end of the satellite link, the protocol will be converted back to TCP. In some instances the protocol converter simply converts between versions of TCP. This approach can be used to extend the usefulness of the satellite.

A second approach to TCP splitting involves using a protocol other than TCP on the satellite segment of the link. The second protocol may be specifically tailored for use in a satellite environment. By replacing TCP with another protocol, the performance may be dramatically enhanced. The only drawback to this type of approach is that it must be possible to look at the TCP headers. This means that it will not work with encryption techniques that encrypt the transport header unless the gateway is a trusted system [2].

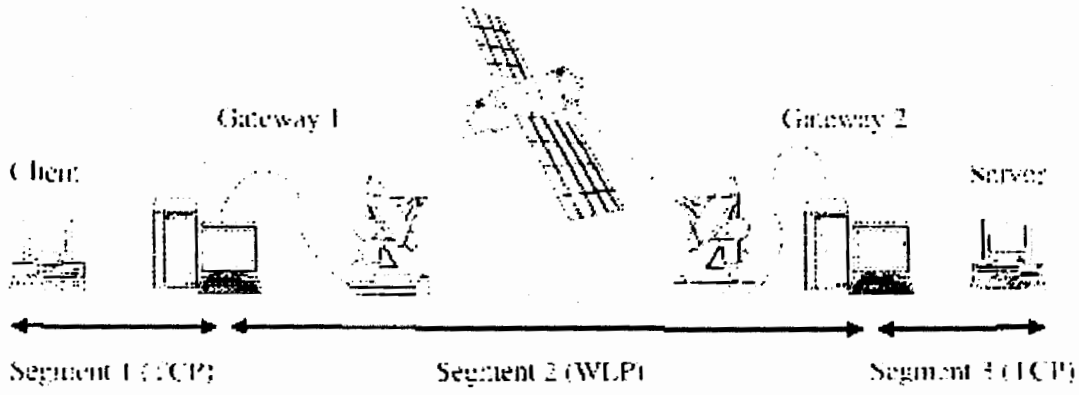


Figure 1.3 (Splitting Mechanism)

1.5 NETWORK SIMULATOR-2 (NS-2):

NS is a discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

NS began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. NS is open source package that has always included substantial contributions from other researchers.

1.5.1 NS INSTALLATION:

Version:

NS evolves through version 1, version 2, and the most up-to-date version is 2.27. The versions available to download on NS page are from 2.1b3 to 2.1b8a for all-in-one package. You may download any version that is best fit your need. The newer the version is, the more modules and features it has. But it does not mean the newer, the better -- you may not need the newly added parts at all.

For NS source core, you could get as early as version 2.0(which is rarely used now). Link to download version 1 of NS is also there in case some researchers may need to use that version.

Platform:

NS supports Unix, Linux, and Windows-95/98/2000/NT. Unix is highly preferred, since you will experience less problems in using NS2 on Unix than on Windows.

Components:

The main components of NS-2 are as below:

- Tcl release 8.3.2 (required component)
- Tk release 8.3.2 (required component)
- Otcl release 1.0a7 (required component)
- TclCL release 1.0b11 (required component): simulation interface
- NS release 2.27 (required component): simulation code core
- Nam release 1.0a10 (optional component): animation tool
- Xgraph version 12 (optional component): graphic tool

Requirements:

To build NS you need a computer and a C++ compiler. NS is fairly large. The all-in-one package requires about 250MB of disk space to be built. Building NS from pieces can save some disk space.

1.5.2 NS FUNCTIONS:

The main functions of NS-2 are given below:

- Creating the event scheduler
- Creating network
- Computing routes
- Creating connection
- Create queuing scheme
- Creating traffic
- Inserting errors
- Tracing

1.5.4 GETTING START WITH NS:

NS is a large and complicated package.

To start and get some intuitive sense of NS, you could run some sample scripts, from simple ones to complicate ones. The ones with Nam animation might give you more feeling on NS. You could also run the samples in the area you are especially interested in.

To learn NS, NS tutorials on the web are good resource. They walk you through the key NS development processes in an incremental way, and educate you about key concepts in NS. They could help you get basic understanding in a short period time.

After preparing you with basic knowledge of NS, studying NS manual is an important and inevitable step if you want to make your own agents and complicated simulation schemes.

NS is open source and used by many researchers who already built up rich knowledge and experience with NS. They would like to share their experience and help each other. To get help from them and learn the lessons from other people's experience, you could subscribe NS mailing list.

1.5.5 MAKING CHANGES IN NS:

C++ makes the core part of NS. In NS, C++ objects have hierarchical and cross relations among each other. Changing in one part might have potential impact on other parts, so it has to be very careful. To add new C++ program or class in NS, you need to change the Makefile and recompile it. To change existing C++ code, you always need to recompile it.

Otcl is a script language. The overhead to change it is less compared to changing C++ code. To add new Otcl program or class in NS Otcl library, you need to source it, change the Makefile and recompile it, but to add your own testing Otcl scripts, you could directly run it without changing Makefile and recompiling them. To change existing Otcl code, you always need to recompile, and might source them if necessary. But to change your test scripts, most of the time you can directly run [5].

Chapter 2

LITERATURE REVIEW

2 LITERATURE REVIEW

Before starting research on splitting technique we have searched many research papers on splitting technique through Internet. The only properly discussed research papers we came across during our literature review are as follows:

- J. Scott Stadler et. al have done a lot of research on the TCP/IP protocol suite and finally find that TCP shows a poor performance when it is used for satellite link. So after considering splitting mechanism, with the help of MIT Lincolin Laboratory they have developed the "Wireless IP Suite Enhancer (WISE)" which dramatically improves the performance of TCP/IP when it is extended via a satellite link. At last in 1998 they have published "Performance Enhancement for TCP/IP on a Satellite Channel", research paper. This brings a new idea of TCP enhancer that significantly improves performance in a satellite environment.

The WISE approach consists of software that is added to gateways at the periphery of the wireless segment of the network. WISE operates by transparently splitting the TCP connection into three segments, client to gateway, gateway to gateway via the wireless link, and gateway to server.

The gateway to gateway connection, however, uses a special Wireless Link Protocol (WLP) developed according to the physical characteristics of the wireless link at hand. The WISE software is responsible for converting TCP to WLP upon entering the wireless sub-network and back to TCP upon exiting [2].

- Xu Xin et. al have discussed their idea about splitting in "Performance Analysis of Transport Protocol in Satellite Network". Slow start algorithm wastes a lot of bandwidth on satellite networks. Because of large delay*bandwidth product it takes a large time to increase the congestion window to fill the link and hence effectively utilize the bandwidth. Delayed ACKs also cause wasted bandwidth during the slow start phase. One method to deal with this is to increase the initial value of congestion window.

TCP uses timeouts to detect lost segments. When the timer expires the TCP retransmits the data and performs congestion control by setting ssthresh to half that of the window and making window 1 and then it starts the slow start. This algorithm uses three duplicate ACKs to trigger retransmission of the lost segments of data and the sender can retransmit them without waiting for the timeout. This is called *fast retransmit*. Now it adjusts the window size, which is called *fast recovery*. The value of ssthresh is set to half the value of the window, the window size is halved. The window size now is increased by one segment for each duplicate ACK it receives. When an ACK for the retransmitted packet is received the congestion window is restored to ssthresh which is equal to half the original value of the window size when the congestion was detected.

When multiple packets of data are lost the TCP at sender waits for the timeout and determines which segments have to be retransmitted. During this time, the data segments and their acknowledgements are lost from the network. As there are no ACKs coming in the sender invokes slow start and restarts transmission which is very time consuming and degrades the TCP performance.

One solution, Satellite Transport Protocol (STP) is a devised to optimize the transmit performance in the networks exhibiting high latency, high error rate and asymmetric [3].

- M. Gerla et. al have published their idea in "TCP via Satellite Constellations". In the last some years, a large number of proposed enhancements and changes to the basic TCP structure have been put forward. Each of these offers its specific features. Some involves modifications and additions to the protocol itself, while others involve implementation changes. At the same time, the Internet community remains sensitive to the broad effect that changes can carry, particularly with respect to congestion control. In spite of the large number of proposals, very few TCP varieties have gained broad acceptance. Some of more commonly known include Tahoe, Reno, Vegas and SACK.

The traditional Tahoe version of TCP includes the basic slow-start and congestion avoidance mechanisms, and incorporates the Fast Retransmit method of avoiding time-

out by retransmitting data upon receiving a duplicate acknowledgement. In case a packet is lost and the TCP sender does not receive the ACK, it times out and retransmits the packet, reduces its congestion window to one and enters slow-start. If three or more duplicate ACKs are received in a row, it is the strongly indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.

In most respects similar to Tahoe, the TCP Reno implementation incorporates the Fast Recovery mechanism. Fast Recovery operates within congestion avoidance instead of going into slow-start in response to the Fast Retransmit condition. Thus, the sender halves the congestion window rather than sending it back to one, as in TCP Tahoe, and thereafter increases occur incrementally. In fact, since the receiver can only generate the duplicate ACK when another segment is received, there must still be data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow-start.

A variant of Reno, called New Reno, modifies its behaviour when receiving new ACKs in the Fast Recovery phase. In order to exit Fast Recovery, the sender must receive an ACK for the highest sequence number sent.
TCP Vegas takes a different approach by beginning retransmit early when the time difference between a send and its corresponding ACK exceeds the time-out value. Vegas implementations also include a congestion control mechanism that will increase/decrease the congestion window relative to the difference between a sampled throughput measurement and the value of the window size normalized by the round-trip time measurement.

Other schemes like TCP SACK and FACK show extremely good performance on very lossy links. In TCP SACK, when the receiver holds non-contiguous data, it sends duplicate ACKs bearing SACK options to inform the sender which segments have been correctly received. TCP FACK uses the additional information provided by the SACK

option to keep an explicit measure of outstanding data in the network and is able to regulate it to be within one segment size of the congestion window.

They have also discussed, about major differences when a GEO network or a LEO network is considered for TCP traffic delivery. In case of GEO networks we are generally concerned with a TCP/IP service scenario for fixed terminals located at user premises. Therefore the main problems are connected with high latency and low signal-to-noise ratios, and measures are needed to reduce their impacts on throughput performance. In fact, the round-trip GEO satellite delay is around half a second. On the other hand, even when only signal-to-noise ratio effects are considered, BER is typically not better.

LEO networks present a more complex behaviour. In this case our service are those for the ubiquitous user and for the mobile user. Round-trip transmission delay is on the order of a few ten milliseconds, so it is in line with terrestrial networks. However, delay variation could be significant as it ranges typically between 13ms and 50ms for an orbital height of 1000km. Packets can be lost also due to shadowing and blockage, especially in urban environments [4].

- M. Luglio et. al have discussed that large delays lengthen the duration of the "Slow Start" interval because the slow start process depends upon the RTT of the connection. They also analyze that lossy characteristic of satellite link creates more problems for TCP. So finally they proposed a technique which involves subdividing connections into terrestrial and space segments, or "Splitting" the connection.

In this paper, "TCP Performance Using Splitting over the Satellite Link", they have examined the idea of adding transport services to the satellite in order to improve the end-to-end performance of TCP over satellite link. We find that the "Splitting" of TCP connections into separate uplink and downlink components yields significant performance improvements. Moreover, as the connection suffers from additional problems from mobility and multiple satellites, the on-board cache makes a greater.

The proposed solutions involve changes to the TCP flow-control mechanisms like New Reno and Peach. TCP peach replaces Slow Start and Fast Recovery with Sudden Start and Rapid Recovery. SACK also limits waste of satellite resources upon retransmissions by changing TCP's error-control mechanism. A modification to the Fast Recovery algorithm was developed, called TCP Westwood [6].

- M. Luglio et. al show their research in "On Board Satellite "Split TCP" Proxy". They introduce a forwarding (Proxy) agent on the satellite maintains two separate split TCP connections for each end point of the TCP session. By splitting the TCP connection on board, we can increase the speed of error recovery and we can reduce the propagation delay on each link.

Subdividing the connection into earth-to-space connections results in a further reduction in windows sizes by approximately one-half. So the reduction in window size comes at the expense of memory and processing on board the satellite.

On-board forwarding agent, residing on the satellite, this agent receives data and acknowledgements from uplinks while sending, retransmitting and acknowledging packets on downlinks. More specifically, when the forwarding agent receives a valid packet on the uplink connection, it acknowledges this packet on the uplink connection and places the packet in the cache to be sent on the downlink. Consistent with the TCP sender, this packet remains in the cache until acknowledged by the downlink connection. At this point, the packet has successfully returned to earth and may be removed from the cache. Packets both lost on the downlink and subsequently corrupted in memory result in unrecoverable packets. In the semantics of TCP, this in effect represents a connection failure [7].

- Jing Peng et. al have written their idea in "Improving TCP Performance over Long Delay Satellite Link". They have implemented sharing TCP state information for the purpose of improving TCP performance over long delay satellite links. Sender side modification has been made to the TCP protocol. The resulting TCP is called Sharing

TCP (STCP). In STCP, information about the channel between a host-pair is shared among sequential and concurrent connections. Concurrent connections are under better control so that the possibility of congestion losses is reduced. STCP, provides a new mechanism to allocate the network capacity among concurrent connections in a relatively fair manner.

STCP is based on the idea of sharing TCP state information proposed in RFC 2140. The design goal of STCP is to speed up similar connections and to coordinate concurrent connections by sharing the information about the network condition. In STCP, TCP manager maintains a list of ECBs. When opening a new connection, TCP manager searches the list for an existing ECB associated with the destination IP address. If no ECB is found, a new ECB is created for that destination IP address.

Simulation results show that this solution performs better than standard TCP when there exist information about the same channel discovered by previous connections [8].

- P. Loreti et. al have discussed "Satellite Systems Performance with TCP-IP Applications". The access to the internet in the presence of wide range mobility represents one of the key issues for future telecommunication systems. In this paper, we investigated the performance of various TCP schemes in satellite environments characterized by variable propagation conditions. They evaluated different architectures (LEO, single hop, full) in representative fixed/mobile terminal scenarios.

The simulations show that TCP Westwood is able to outperform TCP Reno and SACK in the presence of random errors or shadowing. The faster recovery algorithm used by TCP Westwood helps it to recover quickly from packet errors [9].

- Mario Gerla et. al have published their paper named "TCP Westwood: Congestion Window Control using Bandwidth Estimation". TCP Westwood (TCPW), a new protocol with a sender-side modification of the window congestion control scheme. TCP

Westwood controls the window using end-to-end rate estimation in a way which is totally transparent to routers and to the destination.

In TCP Westwood the sender continuously computes the connection BandWidth Estimate (BWE). Thus, BWE is equal to the rate at which data is delivered to the TCP receiver. The estimate is based on the rate at which ACks received and on their payload. After a packet loss indication, (i.e, reception of 3 duplicate ACKs, or timeout expiration). , the sender resets the congestion window and the slow start threshold based on BWE. More precisely, cwin=BWE*RTT [10].

- Joseph Ishac et. al has discussed the mechanism of Spoofing in "Satellite and Terrestrial Network Analysis". Spoofing involves the transparent splitting of a network connection between the source and destination by some entity within the network path. The performance gain of spoofing is less beneficial for smaller sized transfers than gains obtained when transferring large files.

The objective of spoofing involves isolating the long-latency link by introducing a middle agent which splits the TCP connection. The middle agent, or "spoofer", takes on the personality of both parties. The responsibility of the spoofer is to intercept, cache, and acknowledge data received by the sender and then forward that data to the receiver.

As a result of their simulation, they find that spoofing is indeed beneficial for large file transfers. For small transfer sizes, spoofing greatly increased the throughput seen by the sender, but was much less beneficial for throughput observed at the receiver. Since a majority of data sent across networks is small, spoofing will not provide much advantage to a standard home user. However, benefits to web servers and other content providers would be significant. Also, spoofing allows for data to accumulate at the spoofer, increasing the number of dropped data packets, which also degrades the receivers perceived performance [11].

- Milenko Petrovic et. al has discussed routing protocols in "Routing Protocols for Ad Hoc Networks". There also exist routing protocols that contain both a pro-active and on-demand component. Such protocols are termed hybrid.

Operation of the AODV (Ad-hoc On-Demand Distance Vector) can be divided in two functions – route discovery and route maintenance. During protocol initialization, neighbours are discovered. A node sends Hello message on its interface and receives Hello messages from its neighbours. This process repeats periodically to determine neighbour connectivity. When a route is needed to some destination, the protocol starts route discovery. The source sends Route Request Message to its neighbours. If a neighbour has no information on the destination, it will send message to all of its neighbours and so on.

DSR uses a modified version of source routing. Operation of the protocol can be divided in two functions – route discovery and route maintenance. Route discovery operation is used when routes to unknown hosts are required. Route maintenance operation is used to monitor correctness of established routes and to initiate route discovery if a route fails.

OLSR provides loop-free operation. This protocol is based on link state routing algorithm. It is an optimized version of link state routing for ad hoc networks. The optimization tries to reduce number of broadcasts during flooding of link state information.

FSR (Fisheye State Routing) is based on link state routing. The goal of FSR is to reduce flooding used in disseminating link state information when connectivity changes.

LANMAR (Landmark Ad hoc Routing) is a combination of link state and distance vector protocols . It borrows from FSR and Landmark routing protocols. It is best suited to networks where group mobility applies. LANMAR uses concepts of landmarks to reduce size of routing tables and effectively handle changes in topology resulting from node mobility. A group of nodes that are in close proximity of each other (i.e. direct

communication is possible) have a designated landmark. The landmark node has its Landmark flag set to ON. Nodes that have the same landmark are within the same scope. Routing table at each node contains only routes to nodes in the same scope, and routes to all landmarks. This makes LANMAR suitable for large networks.

In DSDV (Destination-Sequenced Distance Vector) each node keeps a list of all nodes with next hops on the path to a particular node, just like in a standard distance vector protocol. Nodes exchange updates whenever change in topology is detected by some node. Each update packet has a sequence number. Sequence numbers are used to guarantee loop-free operation. Sequence number is assigned by the originating node. When a node receives an update packet, it checks the sequence number in that packet. If information in the packet is older that what the node has in its routing tables, then the packet is discarded. Otherwise, information is entered in the topology database, and routing tables are appropriately updated. The update packet is then forwarded to all neighbour nodes (except the one from which the packet came). In addition, the node sends any new information from merging of information from the update packet. update can be delayed until best route arrives.

TBRPF (Topology Broadcast based on Reverse-Path Forwarding) is a link state protocol. Its main goal is to improve dissemination of link state information through controlled flooding. The way it tries to control flooding is to minimize number of messages flooded, by building a minimum spanning tree of the network (at each node) and sending link state messages by following the minimum spanning tree path. That way every node receives update message exactly once. Link state information is sent whenever a change in topology at a node occurs [12].

- GDSG-APAC-MANET have discussed their idea for mobile Ad Hoc networks

in "Mobile Ad Hoc Networks for the Military" and also discuss the meaning of MANET. Mobility-the mobile nodes in the network will follow some form of synthetic or observed mobility model with direction ranging from random to predictable, and velocity ranging from static to fast moving. Ad-hoc topology-the network should have no fixed

infrastructure, with the topology of the network being defined by the positions of the mobile nodes at a particular time. Wireless network-mobile ad-hoc networks are, by virtue of their characteristics, wireless [13].

- Xiaoyan Hong et. al have their idea about "Scalable Routing Protocols for Mobile Ad Hoc Networks". The routing protocols we intend to include in the survey fall into three categories flat routing protocols, hierarchical routing approaches, and GPS augmented geographical routing schemes. A network that does not rely on a fixed infra structure and works in a shared wireless media. Such a network, called a mobile adhoc network (MANET), is a self-organizing and self-configuring multi-hop wireless network, where the network structure changes dynamically due to member mobility [14].

Chapter 3

PROBLEM DOMAIN
&
PROPOSED SOLUTION

3 PROBLEM DOMAIN AND PROPOSED SOLUTION:

A satellite is a specialized wireless receiver/transmitter that is launched by a rocket and placed in orbit around the earth. A communication satellite is space craft that orbits the earth and relays messages, radio, telephone and television signals. Stations on the ground, called earth stations, transmit signals to the satellite, which the relays the signal to other earth stations. The TCP/IP protocol suite that forms the basis of the Internet was designed to operate over an extremely large range of environments. But due to some characteristics of satellite, performance of TCP is degraded. Improving end-to-end performance of TCP we use the concept of Splitting mechanism in LEO satellite system, which is proposed idea.

3.1 PROBLEM DOMAIN:

The properties of satellite that can degrade TCP performance are given below:

3.1.1 LARGE DELAYS:

The large propagation delay characteristic of satellites can severely impact TCP performance. First of all, large delay lengthens the duration of the "Slow Start" interval because the slow start process depends upon the Round Trip Time (RTT) of the connection. The TCP source sends up to one window of packets; TCP controls the sending packet rate by adjusting the size of the sender's window. Upon entering the slow start, TCP begins with a window of one segment and doubles this window approximately once for every RTT (assuming the receiver acknowledges each packet). In fact, most implementation of TCP do not send a separate acknowledgement of every received packet, and thus the window grows more slowly. In any case, the rate at which the window increases depends upon RTT. Large RTT's also affect the rate at which the window grows during congestion avoidance.

There are common aspects, but also major differences when a GEO network or a LEO network is considered for TCP traffic delivery. In case of GEO networks we are generally concerned with a TCP/IP service scenario for fixed terminals located at user premises. Therefore the main problems are connected with high latency and low signal-to-noise ratios, and measures are needed to reduce their impacts on throughput performance. In fact, the round-trip GEO satellite delay is around half a second. On the other hand, even when only signal-to-noise ratio effects are considered, BER is typically not better [4].

LEO networks present a more complex behaviour. In this case our service are those for the ubiquitous user and for the mobile user. Round-trip transmission delay is on the order of a few ten milliseconds, so it is in line with terrestrial networks. However, delay variation could be significant as it ranges typically between 13ms and 50ms for an orbital height of 1000km. Packets can be lost also due to shadowing and blockage, especially in urban environments [4].

3.1.2 LOSSY LINKS:

The congestion-control mechanism of TCP relies upon lost packets as indicators of congestion. TCP congestion-control assumes packet are normally lost as a result of overflowing queues serving bottleneck links and infrequently lost for non-congestion reasons such as link errors and other types of corruption. While this assumption holds true for wired networks, it fails for many wireless links including satellites. Thus, when a TCP connection experiences relatively frequent losses from link level errors, TCP performance suffers as a result. In fact, the impact of random errors upon performance increases with the size of the window. This occurs both as a consequence of and increased probability of multiple losses in one RTT-often leading to timeouts-and the increased duration of slow start. So far satellite links, with both higher losses and delays, random losses severely impact performance.

3.2 PROPOSED SOLUTION:

Improving end-to-end performance of TCP we Split TCP connections, which is proposed idea.

3.2.1 SPLIT TCP CONNECTIONS:

Split:

SPLIT scheme uses an intermediate host to divide a TCP connection into two separate TCP connections. The implementation avoids data copying in the intermediate host by passing the pointers to the same buffer between the two TCP connections. A variant of the SPLIT approach that was investigated, SPLIT-SMART, uses a SMART-based selective acknowledgement scheme on the wireless connection to perform selective retransmissions. There is a little chance of reordering of packets over the wireless connection since the intermediate host is only one hop away from the final destination [15].

Approaches:

The following the approaches of Split TCP connections:

- **TCP Spoofing:**

Spoofing involves the transparent splitting of a network connection between the source and destination by some entity within the network path. The performance gain of spoofing is less beneficial for smaller sized transfers than gains obtained when transferring large files [11].

The objective of spoofing involves isolating the long-latency link by introducing a middle agent which splits the TCP connection. The middle agent, or "spoofer", takes on the

personality of both parties. The responsibility of the spoofer is to intercept, cache, and acknowledge data received by the sender and then forward that data to the receiver [11].

- **TCP Splitting**:

TCP splitting is an approach that uses a gateway at the periphery of the satellite network to convert TCP traffic into an intermediate protocol that is well suited for the satellite environment. On the other end of the satellite link, the protocol will be converted back to TCP. In some instances the protocol converter simply converts between versions of TCP. This approach can be used to extend the usefulness of the satellite.

A second approach to TCP splitting involves using a protocol other than TCP on the satellite segment of the link. The second protocol may be specifically tailored for use in a satellite environment. By replacing TCP with another protocol, the performance may be dramatically enhanced. The only drawback to this type of approach is that it must be possible to look at the TCP headers. This means that it will not work with encryption techniques that encrypt the transport header unless the gateway is a trusted system [2].

- **Web Caching:**

Split connection at cache misses.

Chapter 4

DESIGNING

4 DESIGNING

Designing is actually a multi-step process that focuses on four distinct attributes of a program-data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins.

4.1 NETWORK SIMULATOR-2 (NS-2):

NS is a discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

NS began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. NS is open source package that has always included substantial contributions from other researchers.

4.1.1 THE CLASS SIMULATOR:

The overall simulator is described by a Tcl class Simulator. It provides a set of interfaces for configuring a simulation and for choosing the type of event scheduler used to drive the simulation. A simulation script generally begins by creating an instance of this class and calling various methods to create nodes, topologies, and configure other aspects of the simulation. A subclass of Simulator called OldSim is used to support *ns* v1 backward compatibility.

4.1.2 TCP AGENTS IN NS:

This section describes the operation of the TCP agents in *ns*. There are two major types of TCP agents: one-way agents and a two-way agent. One-way agents are further subdivided into a set of TCP senders (which obey different congestion and error control

techniques) and receivers ("sinks"). The two-way agent is symmetric in the sense that it represents both a sender and receiver. It is still under development. Basically it covers most files matching the regular expression ~*ns*/tcp*.{cc, h}.

The one-way TCP sending agents currently supported are:

- Agent/TCP - a "tahoe" TCP sender
- Agent/TCP/Reno - a "Reno" TCP sender
- Agent/TCP/Newreno - Reno with a modification
- Agent/TCP/Sack1 - TCP with selective repeat (follows RFC2018)
- Agent/TCP/Vegas - TCP Vegas
- Agent/TCP/Fack - Reno TCP with "forward acknowledgment"

The one-way TCP receiving agents currently supported are:

- Agent/TCPSink - TCP sink with one ACK per packet
- Agent/TCPSink/DelAck - TCP sink with configurable delay per ACK
- Agent/TCPSink/Sack1 - selective ACK sink (follows RFC2018)
- Agent/TCPSink/Sack1/DelAck - Sack1 with DelAck

The two-way experimental sender currently supports only a Reno form of TCP:

- Agent/TCP/FullTcp

4.1.3 NODE BASICS:

The basic primitive for creating a node is

```
set ns [new Simulator]
$ns node
```

The instance procedure node constructs a node out of more simple classifier objects . The Node itself is a standalone class in OTcl. However, most of the components of the node are themselves TclObjects.

4.1.4 THE CLASSIFIER:

The function of a node when it receives a packet is to examine the packet's fields, usually its destination address, and on occasion, its source address. It should then map the values to an outgoing interface object that is the next downstream recipient of this packet. In *ns*, this task is performed by a simple *classifier* object. Multiple classifier objects, each looking at a specific portion of the packet forward the packet through the node. A node in *ns* uses many different types of classifiers for different purposes.

A classifier provides a way to match a packet against some logical criteria and retrieve a reference to another simulation object based on the match results. Each classifier contains a table of simulation objects indexed by *slot number*. The job of a classifier is to determine the slot number associated with a received packet and forward that packet to the object referenced by that particular slot. The C++ class Classifier (defined in ~*ns*/classifier.h) provides a base class from which other classifiers are derived.

4.1.5 SIMPLE LINK:

The class Link is a standalone class in OTcl, that provides a few simple primitives. The class SimpleLink provides the ability to connect two nodes with a point to point link. *ns* provides the instance procedure simplex-link{} to form a unidirectional link from one node to another. The link is in the class SimpleLink.

4.1.6 C++ AND OTCL SEPARATION:

C++ is for data and per packet action. The overhead of modifying C++ code is more compared to that of script languages.

Otcl is for control and periodic or triggered action. Otcl is interpreting language and script driven, so it is easy to change.

The advantage to have 2 languages is to achieve compromise between composibility and speed.

The disadvantage to have 2 languages is long learning curve and hard to debug.

4.1.7 OTCL LINKAGE IN NS:

NS is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy). The two hierarchies are closely related to each other; from the user's perspective. The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass. user instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important.

Concept Overview:

On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important.

NS meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. ns (via `tclcl`) provides glue to make objects and variables appear on both langauges.

Code Overview:

The code to interface with the interpreter resides in a separate directory, `tclcl`. The rest of the simulator code resides in the directory, `ns-2`. We will use. There are a number of classes defined in *~tclcl/*. We only focus on the six that are used in *ns*- The Class Tcl, contains the methods that C++ code will use to access the interpreter. The class TclObject, is the base class for all simulator objects that are also mirrored in the compiled hierarchy. The class TclClass, defines the interpreted class hierarchy, and the methods to permit the user to instantiate TclObjects. The class TclCommand, is used to define simple global interpreter commands. The class EmbeddedTcl, contains the methods to load higher level built in commands that make configuring simulations easier. Finally, the class InstVar, contains methods to access C++ member variables as OTcl instance variables.
The procedures and functions can be found in *~tclcl/*Tcl.{cc, h}, *~tclcl/*Tcl2.cc, *~tclcl/*tcl-object.tcl, and, *~tclcl/*tracedvar.{cc, h}. The file *~tclcl/*tcl2c++.c is used in building *ns*.

Class Tcl:

The class Tcl encapsulates the actual instance of the OTcl interpreter, and provides the methods to access and communicate with that interpreter. The methods described in this section are relevant to the *ns* programmer who is writing C++ code.

Class TclObject:

Class TclObject is the base class for most of the other classes in the interpreted and compiled hierarchies. Every object in the class TclObject is created by the user from within the interpreter. An equivalent shadow object is created in the compiled hierarchy. The two objects are closely associated with each other. The class TclClass, contains the mechanisms that perform this shadowing. Refer an object as a TclObject1. By this, we refer to a particular object that is either in the class TclObject, or in a class that is derived from the class TclObject. If it is necessary, we will explicitly qualify whether that object is an object within the interpreter, or an object within the compiled code. In such cases, we will use the abbreviations "interpreted object", and "compiled object" to distinguish the two. And within the compiled code respectively.

In most cases, access to compiled member variables is restricted to compiled code, and access to interpreted member variables is likewise confined to access via interpreted code; however, it is possible to establish bi-directional bindings such that both the interpreted member variable and the compiled member variable access the same data, and changing the value of either variable changes the value of the corresponding paired variable to same value. The binding is established by the compiled constructor when that object is instantiated; it is automatically accessible by the interpreted object as an instance variable. *ns* supports five different data types-reals, bandwidth valued variables, time valued variables, integers, and booleans.

For every TclObject that is created, *ns* establishes the instance procedure, cmd{}, as a hook to executing methods through the compiled shadow object. The procedure cmd{} invokes the method command() of the shadow object automatically, passing the arguments to cmd{} as an argument vector to the command() method. The user can invoke the cmd{} method in one of two ways: by explicitly invoking the procedure, specifying the desired operation as the first argument, or implicitly, as if there were an instance procedure of the same name as the desired operation.

Class TclClass:

This compiled class (class TclClass) is a pure virtual class. Classes derived from this base class provide two functions:

- Construct the interpreted class hierarchy to mirror the compiled class hierarchy.
- Provide methods to instantiate new TclObjects.

Each such derived class is associated with a particular compiled class in the compiled class hierarchy, and can instantiate new objects in the associated class. As an example, consider a class such as the class RenoTcpClass. It is derived from class TclClass, and is associated with the class RenoTcpAgent. It will instantiate new objects in the class RenoTcpAgent. The compiled class hierarchy for RenoTcpAgent is that it derives from TcpAgent, that in turn derives from Agent, that in turn derives (roughly) from TclObject.

Class TclCommand:

This class (class TclCommand) provides just the mechanism for *ns* to export simple commands to the interpreter, that can then be executed within a global context by the interpreter. There are two functions defined in ~*ns*/misc.cc: ns-random and ns-version. These two functions are initialized by the function init_misc(void), defined in ~*ns*/misc.cc; init_misc is invoked by Tcl_AppInit(void) during startup.

Class EmbeddedTcl:

NS permits the development of functionality in either compiled code, or through interpreter code, that is evaluated at initialization. For example, the scripts ~*tclcl*/tcl-object.tcl or the scripts in ~*ns*/tcl/lib. Such loading and evaluation of scripts is done through objects in the class EmbeddedTcl. The easiest way to extend *ns* is to add OTcl code to either ~*tclcl*/tcl-object.tcl or through scripts in the ~*ns*/tcl/lib directory. Note that, in the latter case, *ns* sources ~*ns*/tcl/lib/ns-lib.tcl automatically, and hence the

programmer must add a couple of lines to this file so that their script will also get automatically sourced by *ns* at startup. As an example, the file ~*ns*/tcl/mcast/srm.tcl defines some of the instance procedures to run SRM. In ~*ns*/tcl/lib/ns-lib.tcl.

Class InstVar:

This section describes the internals of the class InstVar. This class defines the methods and mechanisms to bind a C++ member variable in the compiled shadow object to a specified OTcl instance variable in the equivalent interpreted object. The binding is set up such that the value of the variable can be set or accessed either from within the interpreter, or from within the compiled code at all times [16].

Chapter 5

DEVELOPMENT

5 DEVELOPMENT

During development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how the design will be translated into a programming language, and how testing will be performed.

5.1 TOOLS AND TECHNOLOGIES:

The tools and technologies used in this project are as follows:

5.1.1 NETWORK SIMULATOR-2 (NS-2):

NS is a discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

NS began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. NS is open source package that has always included substantial contributions from other researchers.

NS Installation:

- **Version:**

NS evolves through version 1, version 2, and the most up-to-date version is 2.1b8a. The versions available to download on NS page are from 2.1b3 to 2.1b8a for all-in-one package. You may download any version that is best fit your need. The newer the version is, the more modules and features it has. But it does not mean the newer, the better -- you may not need the newly added parts at all.

For NS source core, you could get as early as version 2.0(which is rarely used now). Link to download version 1 of NS is also there in case some researchers may need to use that version.

- **Platform:**

NS supports Unix (FreeBSD, SunOS, Solaris), Linux, and Windows-95/98/2000/NT. Unix is highly preferred, since you will experience less problems in using NS2 on Unix than on Windows.

- **Components:**

The main components of NS-2 are as below:

- Tcl release 8.3.2 (required component)
- Tk release 8.3.2 (required component)
- Otcl release 1.0a7 (required component)
- TclCL release 1.0b11 (required component): simulation interface
- NS release 2.1b8a (required component): simulation code core
- Nam release 1.0a10 (optional component): animation tool
- Xgraph version 12 (optional component): graphic tool

- **Requirements:**

To build NS you need a computer and a C++ compiler. NS is fairly large. The all-in-one package requires about 250MB of disk space to be built. Building NS from pieces can save some disk space.

NS Functions:

The main functions of NS-2 are given below:

- Creating the event scheduler
- Creating network
- Computing routes
- Creating connection
- Create queuing scheme

- Creating traffic
- Inserting errors
- Tracing

NS Development:

- **C++ and Otcl Separation:**

C++ is for data and per packet action. The overhead of modifying C++ code is more compared to that of script languages.

Otcl is for control and periodic or triggered action. Otcl is interpreting language and script driven, so it is easy to change.

The advantage to have 2 languages is to achieve compromise between composibility and speed.

The disadvantage to have 2 languages is long learning curve and hard to debug.

Getting Start with NS:

NS is a large and complicated package.

To start and get some intuitive sense of NS, you could run some sample scripts, from simple ones to complicate ones. The ones with Nam animation might give you more feeling on NS. You could also run the samples in the area you are especially interested in.

To learn NS, NS tutorials on the web are good resource. They walk you through the key NS development processes in an incremental way, and educate you about key concepts in NS. They could help you get basic understanding in a short period time.

After preparing you with basic knowledge of NS, studying NS manual is an important and inevitable step if you want to make your own agents and complicated simulation schemes.

NS is open source and used by many researchers who already built up rich knowledge and experience with NS. They would like to share their experience and help each other. To get help from them and learn the lessons from other people's experience, you could subscribe NS mailing list.

Making Changes in NS:

C++ makes the core part of NS. In NS, C++ objects have hierarchical and cross relations among each other. Changing in one part might have potential impact on other parts, so it has to be very careful. To add new C++ program or class in NS, you need to change the Makefile and recompile it. To change existing C++ code, you always need to recompile it.

Otcl is a script language. The overhead to change it is less compared to changing C++ code. To add new Otcl program or class in NS Otcl library, you need to source it, change the Makefile and recompile it, but to add your own testing Otcl scripts, you could directly run it without changing Makefile and recompiling them. To change existing Otcl code, you always need to recompile, and might source them if necessary. But to change your test scripts, most of the time you can directly run [5].

Simple Topology in Ns:

This script defines a simple topology of four nodes, and two agents, a UDP agent with a CBR traffic generator, and a TCP agent. The output is two trace files, out.tr and out.nam. When the simulation completes, it will attempt to run a nam visualisation of the simulation on your screen.

 # The preamble
set ns [new Simulator] ;*# initialise the simulation*
 # Predefine tracing
set f [open out.tr w]

```
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
# Some agents.
set udp0 [new Agent/UDP] ;# A UDP agent
$ns attach-agent $n0 $udp0 ;# on node $n0
set cbr0 [new Application/Traffic/CBR] ;# A CBR traffic generator agent
$cbr0 attach-agent $udp0 ;# attached to the UDP agent
$udp0 set class_ 0 ;# actually, the default, but. . .
set null0 [new Agent/Null] ;# Its sink
$ns attach-agent $n3 $null0 ;# on node $n3
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
puts [$cbr0 set packetSize_]
puts [$cbr0 set interval_]
# A FTP over TCP/Tahoe from $n1 to $n3, flowid 2
set tcp [new Agent/TCP]
$tcp set class_ 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
set ftp [new Application/FTP] ;# TCP does not generate its own traffic
$ftp attach-agent $tcp
$ns at 1.2 "$ftp start"
```

$ns connect $tcp $sink

$ns at 1.35 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

TCP Agents in Ns:

This section describes the operation of the TCP agents in *ns*. There are two major types of TCP agents: one-way agents and a two-way agent. One-way agents are further subdivided into a set of TCP senders (which obey different congestion and error control techniques) and receivers ("sinks"). The two-way agent is symmetric in the sense that it represents both a sender and receiver. It is still under development. Basically it covers most files matching the regular expression ~*ns*/tcp*.{cc, h}.

The one-way TCP sending agents currently supported are:

- Agent/TCP - a "tahoe" TCP sender
- Agent/TCP/Reno - a "Reno" TCP sender
- Agent/TCP/Newreno - Reno with a modification
- Agent/TCP/Sack1 - TCP with selective repeat (follows RFC2018)
- Agent/TCP/Vegas - TCP Vegas
- Agent/TCP/Fack - Reno TCP with "forward acknowledgment"

The one-way TCP receiving agents currently supported are:

- Agent/TCPSink - TCP sink with one ACK per packet
- Agent/TCPSink/DelAck - TCP sink with configurable delay per ACK
- Agent/TCPSink/Sack1 - selective ACK sink (follows RFC2018)
- Agent/TCPSink/Sack1/DelAck - Sack1 with DelAck

The two-way experimental sender currently supports only a Reno form of TCP:

- Agent/TCP/FullTcp

- **Simple Configuration:**

```
set ns [new Simulator] ;# preamble initialization
set node1 [$ns node] ;# agent to reside on this node
set node2 [$ns node] ;# agent to reside on this node
set tcp1 [$ns create-connection TCP $node1 TCPSink $node2 42]
$tcp set window_ 50 ;# configure the TCP agent
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 0.0 "$ftp start"
```

This example illustrates the use of the simulator built-in function create-connection. The arguments to this function are: the source agent to create, the source node, the target agent to create, the target node, and the flow ID to be used on the connection. The function operates by creating the two agents, setting the flow ID fields in the agents, attaching the source and target agents to their respective nodes, and finally connecting the agents (i.e. setting appropriate source and destination addresses and ports). The return value of the function is the name of the source agent created.

TCP data source-the TCP agent does not generate any application data on its own; instead, the simulation user can connect any traffic generation module to the TCP agent to generate data. Two applications are commonly used for TCP: FTP and Telnet. FTP represents a bulk data transfer of large size, and telnet chooses its transfer sizes randomly from tcplib (see the file tcplib-telnet.cc.

- **Other Configuration Parameters:**

In addition to the window_ parameter, the TCP agent supports additional configuration variables. Each of the variables described in this subsection is both a class variable and an instance variable. Changing the class variable changes the default value for all agents that are created subsequently. Changing the instance variable of a particular agent only affects the values used by that agent. For example,

```
Agent/TCP set window_ 100 ;# Changes the class variable
```

$tcp set window_ 2.0 ;# *Changes window_for the $tcp object only*

The default parameters for each TCP agent are:

Agent/TCP set window_ 20 ;# *max bound on window size*

Agent/TCP set windowInit_ 1 ;# *initial/reset value of cwnd*

Agent/TCP set windowOption_ 1 ;# *cong avoid algorithm (1: standard)*

Agent/TCP set windowConstant_ 4 ;# *used only when windowOption != 1*

Agent/TCP set windowThresh_ 0.002 ;# *used in computing averaged window*

Agent/TCP set overhead_ 0 ;# *!=0 adds random time between sends*

Agent/TCP set ecn_ 0 ;# *TCP should react to ecn bit*

Agent/TCP set packetSize_ 1000 ;# *packet size used by sender (bytes)*

Agent/TCP set bugFix_ true ;# *see explanation*

Agent/TCP set slow_start_restart_ true ;# *see explanation*

Agent/TCP set tcpTick_ 0.1 ;# *timer granulatiry in sec (.1 is NONSTANDARD)*

Agent/TCP set maxrto_ 64 ;# *bound on RTO (seconds)*

Agent/TCP set dupacks_ 0 ;# *duplicate ACK counter*

Agent/TCP set ack_ 0 ;# *highest ACK received*

Agent/TCP set cwnd_ 0 ;# *congestion window (packets)*

Agent/TCP set awnd_ 0 ;# *averaged cwnd (experimental)*

Agent/TCP set ssthresh_ 0 ;# *slow-stat threshold (packets)*

Agent/TCP set rtt_ 0 ;# *rtt sample*

Agent/TCP set srtt_ 0 ;# *smoothed (averaged) rtt*

Agent/TCP set rttvar_ 0 ;# *mean deviation of rtt samples*

Agent/TCP set backoff_ 0 ;# *current RTO backoff factor*

Agent/TCP set maxseq_ 0 ;# *max (packet) seq number sent*

- **The Base TCP Sender (Tahoe TCP):**

The "Tahoe" TCP agent Agent/TCP performs congestion control and round-trip-time estimation in a way similar to the version of TCP released with the 4.3BSD "Tahoe" UN'X system release from UC Berkeley. The congestion window is increased by one

packet per new ACK received during slow-start and is increased for each new ACK received during congestion avoidance.

- **The Reno TCP:**

The Reno TCP agent is very similar to the Tahoe TCP agent, except it also includes *fast recovery*, where the current congestion window is "inflated" by the number of duplicate ACKs the TCP sender has received before receiving a new ACK. A "new ACK" refers to any ACK with a value higher than the highest seen so far. In addition, the Reno TCP agent does not return to slow-start during a fast retransmit. Rather, it reduces sets the congestion window to half the current window and resets ssthresh_ to match this value.

- **NewReno TCP:**

This agent is based on the Reno TCP agent, but which modifies the action taken when receiving new ACKS. In order to exit fast recovery, the sender must receive an ACK for the highest sequence number sent. Thus, new "partial ACKs" (those which represent new ACKs but do not represent an ACK for all outstanding data) do not deflate the window (and possibly lead to a stall, characteristic of Reno).

- **Vegas TCP:**

This agent implements "Vegas" TCP.It was contributed by Ted Kuo.

- **Sack TCP:**

This agent implements selective repeat, based on selective ACKs provided by the receiver. It follows the ACK scheme, and was developed with Matt Mathis and Jamshid Mahdavi.

- **Fack TCP:**

This agent implements "forward ACK" TCP, a modification of Sack.

- **The Base TCP Sink:**

The base TCP sink object (Agent/TCPSink) is responsible for returning ACKs to a peer TCP source object. It generates one ACK per packet received. The size of the ACKs may be configured. The creation and configuration of the TCP sink object is generally performed automatically by a library call.

5.1.2 RATIONAL ROSE 98:

Rational Rose is a tool to make different diagrams. The benefits provided by the Rational Rose are as follows:

Features:

- It gives the facility to make use case diagram, class diagram, conceptual model, component diagram and deployment diagram.
- It provides basic structures for these diagrams such as classes, actors, entities etc as well as their labeling and relationships.

We have used Rational Rose to create use case diagram and class diagram for our project.

5.2 PROCEDURAL DESIGN:

In procedural design, we have discussed our codes of our project.

5.2.1 SATELLITE NETWORKING IN NS:

Geostationary Satellites:

Geostationary satellites orbit the Earth at an altitude of 22,300 miles above the equator. The position of the satellites is specified in terms of the longitude of the nadir point (sub-satellite point on the Earth's surface). In practice, geostationary satellites can drift from their designated location due to gravitational perturbations– these effects are not modeled in *ns*.

Low-Earth-Orbiting Satellites:

Polar orbiting satellite systems, such as Iridium and the proposed Teledesic system, can be modelled in *ns*. Basic constellation definition Includes satellite altitude, number of satellites, number of planes, number of satellites per plane. Orbit inclination can range continuously from 0 to 180 degrees (inclination greater than 90 degrees corresponds to retrograde orbits). Inter-satellite spacing within a given plane is fixed. Inter-satellite (ISL) links For polar orbiting constellations, intraplane, interplane, and crossseam ISLs can be defined.

Intraplane ISLs exist between satellites in the same plane and are never deactivated or handed off. Interplane ISLs exist between satellites of neighboring co-rotating planes. These links are deactivated near the poles (above the "ISL latitude threshold" in the table) because the antenna pointing mechanism cannot track these links in the polar regions. Like intraplane ISLs, interplane ISLs are never handed off. Crossseam ISLs may exist in a constellation between satellites in counter-rotating planes (where the planes form a so-called "seam" in the topology). GEO ISLs can also be defined for constellations of geostationary satellites.

Ground to satellite (GSL) links means multiple terminals can be connected to a single GSL satellite channel. GSL links for GEO satellites are static, while GSL links for LEO channels are periodically handed off.

Currently, if the (LEO) satellite serving a terminal drops below the elevation mask, the terminal searches for a new satellite above the elevation mask.

Using the Satellite Extensions:

- **Nodes and Node Positions:**

There are two basic kinds of satellite nodes: *geostationary* and *non-geostationary* satellite nodes. In addition, *terminal* nodes can be placed on the Earth's surface. Each of these three different types of nodes is actually implemented with the same class SatNode object, but with different position, handoff manager, and link objects attached. The position object keeps track of the satellite node's location in the coordinate system as a function of the ellapsed simulation time. This position information is used to determine link propagation delays and appropriate times for link handoffs.

Each SatNode and Position object is a split OTcl/C++ object, but most of the code resides in C++. The following types of position objects exist:

Position/Sat/Term A terminal is specified by its latitude and longitude. Latitude ranges from [-90,90] and longitude ranges from [-180,180], with negative values corresponding to south and west, respectively. As simulation time evolves, the terminals move along with the Earth's surface. The node generator can be used to create a terminal with an attached position object as follows:

```
$ns node-config -satNodeType terminal
(other node config commands go here...)
set n1 [$ns node]
$n1 set-position $lat $lon; # in decimal degrees
```

Position/Sat/Geo, a geostationary satellite is specified by its longitude above the equator. As simulation time evolves, the geostationary satellite moves through the coordinate system with the same orbital period as that of the Earth's rotation. The longitude ranges from [-180,180] degrees. Two flavors of geostationary nodes exist:
"geo" (for processing satellites) and "geo-repeater" (for bent-pipe satellites). The node generator can be used to create a geostationary satellite with an attached position object as follows:

```
$ns node-config -satNodeType geo (or ''geo-repeater'')
(other node config commands go here...)
set n1 [$ns node]
$n1 set-position $lon; # in decimal degrees
```

Position/Sat/Polar, a polar orbiting satellite has a purely circular orbit along a fixed plane in the coordinate system; the Earth rotates underneath this orbital plane, so there is both an east-west and a north-south component to the track of a polar satellite's footprint on the Earth's surface. Strictly speaking, the polar position object can be used to model the movement of any circular orbit in a fixed plane.

Satellite orbits are usually specified by six parameters: *altitude*, *semi-major axis*, *eccentricity*, *right ascension of ascending node*, *inclination*, and *time of perigee passage*. The polar orbiting satellites in *ns* have purely circular orbits, so simplify the specification of the orbits to include only three parameters: *altitude*, *inclination*, and *longitude*, with a fourth parameter *alpha* specifying initial position of the satellite. Altitude is specified in kilometers above the Earth's surface, and inclination can range from [0,180] degrees, with 90 corresponding to pure polar orbits and angles greater than 90 degrees corresponding to "retrograde" orbits. The *ascending node* refers to the point where the footprint of the satellite orbital track crosses the equator moving from south to north. In this simulation model, the parameter longitude of ascending node specifies the earth-centric. The fourth parameter, alpha, specifies the initial position of the satellite along this orbit. Finally, a fifth parameter, plane, is specified when creating polar satellite

nodes– all satellites in the same plane are given the same plane index. The node generator used to create a polar satellite with an attached position object as follows:

```
$ns node-config -satNodeType polar
(other node config commands go here...)
set n1 [$ns node]
$n1 set-position $alt $inc $lon $alpha $plane
```

- **Satellite Links:**

Satellite links resemble wireless links. Each satellite node has one or more satellite network interface stacks, to which channels are connected to the physical layer object in the stack. Satellite links differ from *ns* wireless links in two major respects-the transmit and receive interfaces must be connected to different channels, and there is no ARP implementation.

Network interfaces can be added with the following instproc of Class Node/SatNode:

```
$node add-interface $type $ll $qtype $qlim $mac $mac_bw $phy
```

The add-interface instproc returns an index value that can be used to access the network interface stack later in the simulation. By convention, the first interface created on a node is attached to the uplink and downlink channels of a satellite or terminal. The following parameters must be provided:

Type-the following link types can be indicated: geo or polar for links from a terminal to a geo or polar satellite, respectively, gsl and gsl-repeater for links from a satellite to a terminal, and intraplane, interplane, and crossseam ISLs. The type field is used internally in the simulator to identify the different types of links, but structurally they are all very similar.

ll-the link layer type (class LL/Sat is currently the only one defined).

qtype-the queue type (e.g., class Queue/DropTail). Any queue type may be used however, if additional parameters beyond the length of the queue are needed, then this instproc may need to be modified to include more arguments.

qlim-the length of the interface queue, in packets.

mac-the MAC type. Currently, two types are defined: class Mac/Sat– a basic MAC for links with only one receiver (i.e., it does not do collision detection), and Class Mac/Sat/UnslottedAloha– an implementation of unslotted Aloha.

mac_bw-the bandwidth of the link is set by this parameter, which controls the transmission time how fast the MAC sends.

phy-the physical layer– currently two Phys (Class Phy/Sat and Class Phy/Repeater) are defined. The class Phy/Sat just pass the information up and down the stack. A radio propagation model could be attached at this point. The class Phy/Repeater pipes any packets received on a receive interface straight through to a transmit interface.

An ISL can be added between two nodes using the following instproc:

$ns add-isl $ltype $node1 $node2 $bw $qtype $qlim

This creates two channels (of type Channel/Sat), and appropriate network interfaces on both nodes, and attaches the channels to the network interfaces. The bandwidth of the link is set to bw. The linktype (ltype) must be specified as either intraplane, interplane, or crossseam.
A GSL involves adding network interfaces and a channel on board the satellite and then defining the correct interfaces on the terrestrial node and attaching them to the satellite link, as follows:

$node add-gsl $type $ll $qtype $qlim $mac $bw_up $phy

satellite, it restarts the timer and tries again later. If any link changes occur, the routing agent is notified. The elevation mask and handoff timer interval are settable via OTcl:

```
HandoffManager/Term set elevation_mask_ 10; # degrees
HandoffManager/Term set term_handoff_int_ 10; # seconds
```

In addition, handoffs may be randomized to avoid phase effects by setting the following variable:

```
HandoffManager set handoff_randomization_ 0; # 0 is false, 1 is true
```

Crossseam ISLs are the only type of ISLs that are handed off. The criteria for handing off a crossseam ISL is whether or not there exists a satellite in the neighboring plane that is closer to the given satellite than the one to which it is currently connected. Again, a handoff timer running within the handoff manager on the polar satellite determines when the constellation is checked for handoff opportunities. Crossseam ISL handoffs are initiated by satellites in the lower-numbered plane of the two. It is therefore possible for a transient condition to arise in which a polar satellite has two crossseam ISLs (to different satellites). The satellite handoff interval is again settable from OTcl and may also be randomized:

```
HandoffManager/Sat set sat_handoff_int_ 10; # seconds
HandoffManager/Sat set latitude_threshold_ 70; # degrees
```

The handoff manager checks the latitude of itself and its peer satellite upon a handoff timeout; if either or both of the satellites is above latitude_threshold_ degrees latitude (north or south), the link is deactivated until both satellites drop below this threshold.

```
HandoffManager/Sat set longitude_threshold_ 10; # degrees
```

If the two satellites are closer together in longitude than longitude_threshold_ degrees, the link between them is deactivated. This parameter is disabled by default– all defaults for satellite-related bound variables can be found in *~ns*/tcl/lib/ns-sat.tcl.

- **Routing:**

Upon each topology change, a centralized routing genie determines the global network topology, computes new routes for all nodes, and uses the routes to build a forwarding table on each node. Currently, the slot table is kept by a routing agent on each node, and packets not destined for agents on the node are sent by default to this routing agent. For each destination for which the node has a route, the forwarding table contains a pointer to the head of the corresponding outgoing link. The routing genie is a class SatRouteObject and is created and invoked with the following OTcl commands:

```
set satrouteobject_ [new SatRouteObject]
$satrouteobject_ compute_routes
```

Where the call to compute_routes is performed after all of the links and nodes in the simulator have been instantiated. Like the Scheduler, there is one instance of a SatRouteObject in the simulation, and it is accessed by means of an instance variable in C++. For example, the call to recompute routes after a topology change is:

```
SatRouteObject::instance().recompute();
```

It is possible to compute routes using only the hop count and not the propagation delays; in order to do so, set the following default variable to "false":
SatRouteObject set metric_delay_ "true"

Finally, for very large topologies (such as the Teledesic example), the centralized routing code will produce a very slow runtime because it executes an all-pairs shortest path algorithm upon each topology change even if there is no data currently being sent. To speed up simulations in which there is not much data transfer but there are lots of

satcllites and ISLs, one can disable *handoff-driven* and enable *data-driven* route computations. With data-driven computations, routes are computed only when there is a packet to send, and furthermore, a single-source shortest-path algorithm (only for the node with a packet to send) is executed instead of an all-pairs shortest path algorithm. The following OTcl variable can configure this option (which is set to "false" by default):

SatRouteObject set data_driven_computation_ "false"

- **Trace Support:**

Tracefiles using satellite nodes and links are very similar to conventional *ns* tracing. Special SatTrace objects (class SatTrace derives from class Trace) are used to log the geographic latitude and longitude of the node logging the trace (in the case of a satellite node, the latitude and longitude correspond to the nadir point of the satellite). For example, a packet on a link from node 66 to node 26 might normally be logged as:

+ 1.0000 66 26 cbr 210 ------- 0 66.0 67.0 0 0

but in the satellite simulation, the position information is appended:

+ 1.0000 66 26 cbr 210 ------- 0 66.0 67.0 0 0 37.90 -122.30 48.90 -120.94

In this case, node 66 is at latitude 37.90 degrees, longitude -122.30 degrees, while node 26 is a LEO satellite whose sub-satellite point is at 48.90 degrees latitude, -120.94 degrees longitude (negative latitude corresponds to south, while negative longitude corresponds to west).

One addition is the Class Trace/Sat/Error, which traces any packets that are errored by an error model. The error trace logs packets dropped due to errors as follows, for example:

e 1.2404 12 13 cbr 210 ------- 0 12.0 13.0 0 0 -0.00 10.20 -0.00 -10.00

It may happen that a satellite node generates a packet that it cannot forward (such as in sat-mixed.tcl). This will show up as a drop in the tracefile with a destination field set to -2, and the coordinates set to -999.00:

d 848.0000 14 -2 cbr 210 ------- 1 14.0 15.0 6 21 0.00 10.00 -999.00 -999.00

This indicates that node 14, in trying to send a packet to node 15, could not find an available route. To enable tracing of all satellite links in the simulator, use the following commands *before* instantiating nodes and links:

```
set f [open out.tr w]
$ns trace-all $f
```

Then use the following line after all node and link creation (and all error model insertion, if any) to enable tracing of all satellite links:

```
$ns trace-all-satlinks $f
```

Specifically, this will put tracing around the link layer queues in all satellite links, and will put a receive trace between the mac and the link layer for received packets. To enable tracing only on a specific link on a specific node, one may use the command:

```
$node trace-inlink-queue $f $i
$node trace-outlink-queue $f $i
```

Where is the index of the interface to be traced. The implementations of the satellite trace objects can be found in *~ns*/tcl/lib/ns-sat.tcl and *~ns*/sattrace.{cc,h}.

- **Nam Support:**

Nam is not currently supported. Addition of nam for satellite is open to interested contributors.

- **Integration with Wired and Wireless Code:**

This section describes the capabilities and limitations of that code. The satellite code (and the wireless code) normally performs all routing in C++, while the traditional ns code uses a mix of OTcl and C++ code. For backward compatibility reasons, it is difficult to fully integrate both the wired and wireless code. The strategy for integrating wireless and wired code has been to define a special gateway node (called a "base-station"), to use hierarchial routing, and to locate a single base-station node in the wireless network with a network stack located in both the wireless and the wired subnet. Because routing is not fully integrated, the topology of the simulation is limited to only one gateway node per wireless subnet (i.e., a packet cannot enter the wireless network from one wired gateway and leave via another).

The satellite/wired code integration takes a different strategy. By selecting the node configuration $ns node-config-wiredRouting ON option, the C++ routing in the satellite code is turned off, and instead, all satellite topology changes lead to upcalls into the OTcl code. As a result, the link_ array in OTcl is manipulated according to all topology changes, and OTcl-based routing can occur. The penalty for doing this is a much longer execution time for larger simulations (such as Teledesic), but for smaller simulations, the difference is not as noticeable. An example script detailing the use of this new option is shown in *~ns*/tcl/ex/sat-wired.tcl, and a similar test in the satellite test suite exercises this code. Additionally, all of the satellite example scripts in *~ns*/tcl/ex directory can be converted to OTcl routing by using the $ns node-config -wiredRouting ON option.

The wired routing option for satellite has only been tested with (the default) static routing: $ns rtProto Static.The code triggers a global routing table update upon any satellite topology change. The option data_driven_computation_ can not be set to "true" when wiredRouting is ON. Note that the enabling or disabling of data_driven_computation_ can give subtle differences in simulation output since routes are computed at different times (while propagation delays are continuously changing). This effect can be seen by toggling this parameter in the Iridium example script *~ns*/tcl/ex/sat-iridium.tcl.

In the trace file, when a packet is dropped due to "no route to host" (such as when there

is a topology change), the trace looks a bit different depending on whether wiredRouting is turned OFF or ON. In the former case, there is one line per drop, with the destination labelled as "-2". In the latter case, there are three events (enque "+", deque "-", and drop "d") corresponding to the same packet, and the destination is shown as "-1".

In rare cases, there may be warning messages during the execution indicating "node out of range." This can occur if a node becomes disconnected in the topology and then another node tries to send a packet to it. For example, try enabling wiredRouting in the file *~ns*/tcl/ex/sat-mixed.tcl. This occurs because the routing table is dynamically sized upon topology change, and if a node becomes disconnected it may not have any entries inserted in the routing table (and hence the routing table is not grown to accommodate its node number). This warning should not affect actual trace output.

- **Example Scripts:**

Example scripts can be found in the *~ns*/tcl/ex directory, including:

- Sat-mixed.tcl A simulation with a mixture of polar and geostationary satellites.
- Sat-wired.tcl Similar to the previous script, but shows how to connect wired nodes to a satellite simulation.
- Sat-repeater.tcl Demonstrates the use of a simple bent-pipe geostationary satellite, and also so error models.
- Sat-aloha.tcl Simulates one hundred terminals in a mesh-VSAT configuration using an unslotted Aloha MAC protocol with a "bent-pipe" geostationary satellite. Terminals listen to their own transmissions (after a delay), and if they do not successfully receive their own packet within a timeout interval, they perform exponential backoff and then retransmit the packet. Three variants exist: basic, basic_tracing, and poisson. These variants are described further in the header comments of the script.
- Sat-iridium.tcl Simulates a broadband LEO constellation with parameters similar to that of the Iridium constellation (with supporting scripts sat-iridium-links.tcl, sat-iridium-linkswithcross.tcl, and sat-iridium-nodes.tcl).
- Sat-teledesic.tcl Simulates a broadband LEO constellation with parameters similar to

those proposed for the 288 satellite Teledesic constellation (with supporting scripts sat-teledesic-links.tcland sat-teledesic-nodes.tcl).

In addition, there is a test suite script that tries to exercise a lot of features simultaneously, it can be found at *~ns*/tcl/test/testsuite-sat.tcl.

- **Commands at a Glance:**

Following is a list of commands related to satellite networking:

$ns_ node-config -satNodeType <type>

This node configuration declares that the subsequent new nodes created will be of type <type>, where <type> can be one of the following: geo, geo-repeater, polar, terminal.

Other required fields for satellite nodes (for setting up initial links and channels) are as follows:

$ns_ node-config -llType <type>
$ns_ node-config -ifqType <type>
$ns_ node-config -ifqLen <length>
$ns_ node-config -macType <type>
$ns_ node-config -channelType <type>
$ns_ node-config -downlinkBW <value>
(note– satNodeType geo-repeater only requires specifying the channelType– all other options are disregarded.)
$ns_ satnode-polar <alt> <inc> <lon> <alpha> <plane> <linkargs> <chan>

This a simulator wrapper method for creating a polar satellite node. Two links, uplink and downlink, are created along withtwo channels, uplink channel and downlink channel. <alt> is the polar satellite altitude, <inc> is orbit inclination w.r.t equator, <lon> is the longitude of ascending node, <alpha> gives the initial position of the satellite along

this orbit, <plane> defines the plane of the polar satellite. <linkargs> is a list of link argument options that defines the network interface (like LL, Qtype, Qlim, PHY, MAC etc).

$ns_ satnode-geo <lon> <linkargs> <chan>

This is a wrapper method for creating a geo satellite node that first creates a satnode plus two link interfaces (uplink and downlink) plus two satellite channels (uplink and downlink). <chan> defines the type of channel.

$ns_ satnode-geo-repeater <lon> <chan>

This is a wrapper method for making a geo satellite repeater node that first creates a satnode plus two link interfaces (uplink and downlink) plus two satellite channels (uplink and downlink).

$ns_ satnode-terminal <lat> <lon>
This is a wrapper method that simply creates a terminal node. The <lat> and <lon> defines the latitude and longitude respectively of the terminal.

$ns_ satnode <type> <args>

This is a more primitive method for creating satnodes of type <type> which can be polar, geo or terminal.

$satnode add-interface <type> <ll> <qtype> <qlim> <mac_bw> <phy>

This is an internal method of Node/SatNode that sets up link layer, mac layer, interface queue and physical layer structures for the satellite nodes.

$satnode add-isl <ltype> <node1> <node2> <bw> <qtype> <qlim>

This method creates an ISL (inter-satellite link) between the two nodes. The link type (inter, intra or cross-seam), BW of the link, the queue-type and queue-limit are all specified.

$satnode add-gsl <ltype> <opt_ll> <opt_ifq> <opt_qlim> <opt_mac> <opt_bw> <opt_phy> <opt_inlink> <opt_outlink>

This method creates a GSL (ground to satellite link). First a network stack is created that is defined by LL, IfQ, Qlim, MAC, BW and PHY layers. Next the node is attached to the channel inlink and outlink [16].

5.3 ACTIVITY DIAGRAM:

It gives the pictorial representation of algorithm. Activity Diagram is used to represent activities. Basic need is that we want to make procedural design in UML. Operations in sequence are represented in activity diagram. Activity diagrams are useful when we want to describe a behavior which is parallel or when we want to show how behaviors in several use cases interact.

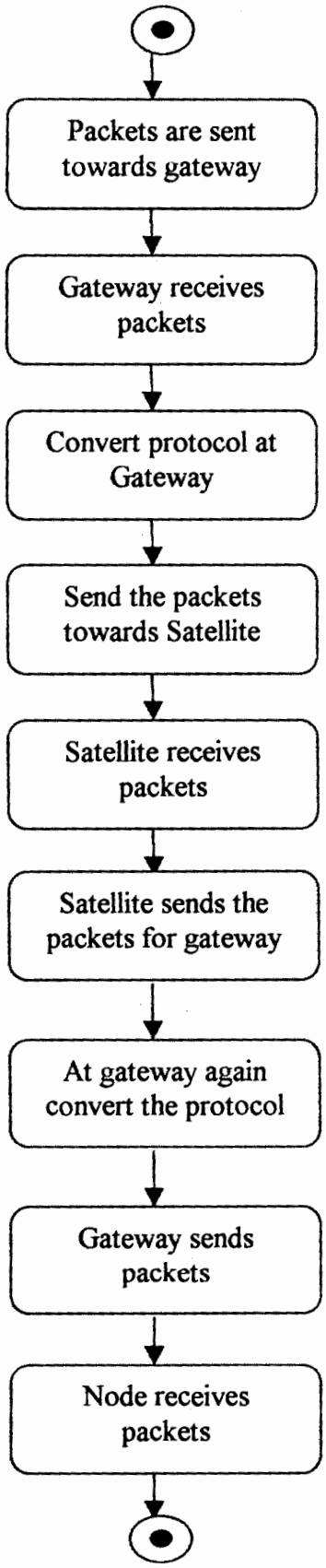


Figure 5.1 (Activity Diagram of Protocol Conversation Module)

Chapter 6

TESTING

6 TESTING:

System testing is an essential step for the development of a reliable and error-free system. Testing is a process of executing a program with the explicit intension of finding errors but this does not mean to embarrass the programmer or fail the product but the positive intention is to remove as many problems from the system as possible. A test case is a set of data items that the system processes as normal input. Good testing involves much more than just running the program a few times to see whether it works. A successful test is the one that finds error.

6.1 OBJECT ORIENTED TESTING STRATEGIES:

Testing begins with unit testing then progress towards integration testing and ends with system testing. In unit testing single modules are tested first. Once they are tested individually they are integrated into a program structure and tested again to find errors due to interfacing of different modules. Finally the system as a whole is tested.

6.2 TYPES OF TESTING:

We conducted the following type of testing to make the software stable and error free.

6.2.1 CODE INSPECTION:

Review and walk through.

6.2.2 UNIT TESTING:

All the modules of the project are first tested individually.

6.2.3 INTEGRATION TESTING:

After all the modules tested individually they are combined to form the final product. All the links and paths were tested. This testing should be done at several levels. E.g. tests of two or three objects, dozens of objects and thousands of them are all needed.

6.2.4 BLACK BOX TESTING:

The software was checked for graphical user interface and measures taken that expected output is generated.

6.2.5 SYSTEM TESTING:

The software was checked as a whole.

6.2.6 BETA TESTING:

Used by outsiders rather than developers often makes up for lack of imagination about possible error paths by testers.

6.2.7 PORTABILITY TESTING:

Test should be applied across the range of systems on which the software may execute. Tests may employ suspected non-portable constructions at the compiler, tool, language, operating system or machine level.

6.3 EVALUATION:

Evaluation of the software is carried out to check the stability and usability of the product being developed. We took measures to ensure that the developed software becomes effective and our research work is a new paradigm in research world. Some of the features of the software are given below:

6.3.1 EFFICIENCY AND EFFECTIVENESS:

The product developed is effective and efficient.

6.3.2 ACCURACY AND RELIABILITY:

The simulator provides reliable and accurate results.

6.3.3 SCALABILITY:

The product is scalable.

6.4 TEST CASE DESCRIPTION:

Following are the test cases:

Test Case ID:	Test 1	**Test Engineers:**	Maleeha Saeed
Objective:	To check nodes creation for polar satellites		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Nodes are successfully created. ■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 2(a)	**Test Engineers:**	Maleeha Saeed
Objective:	To check the nodes configuration.		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Problem occurred. ☐ **Pass** ☐ **Fail** ■ **Not Executed**			

Test Case ID:	Test 2(b)	**Test Engineers:**	Maleeha Saeed
Objective:	To check the nodes configuration.		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result:			
Nodes configured successfully.			
■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 3	**Test Engineers:**	Maleeha Saeed
Objective:	To check the plane of polar satellites		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result:			
The planes for polar satellites are successfully defined.			
■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 4	**Test Engineers:**	Maleeha Saeed
Objective:	To check the positions of polar satellites		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result:			
The positions of polar satellites are successfully defined.			
■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 5	Test Engineers:	Maleeha Saeed
Objective:	To check the next_ variable for polar satellites		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result:			
The setting of next_ variable is successfully done.			
■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 6	Test Engineers:	Maleeha Saeed
Objective:	To check the positions of terminals		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result:			
The positions of terminals are successfully defined.			
■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 7	Test Engineers:	Maleeha Saeed
Objective:	To check Gsl links creation		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result:			
Links successfully created.			
■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 8	**Test Engineers:**	Maleeha Saeed
Objective:	To check links creation		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Links successfully created. ■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 9	**Test Engineers:**	Maleeha Saeed
Objective:	To check the wired nodes creation		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Wired nodes are successfully created. ■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 10(a)	**Test Engineers:**	Maleeha Saeed
Objective:	To check the protocol conversion		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Problem occurred. ☐ **Pass** ☐ **Fail** ■ **Not Executed**			

Test Case ID:	Test 10(b)	**Test Engineers:**	Maleeha Saeed
Objective:	To check the protocol conversion		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Protocols are successfully converted. ■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Test Case ID:	Test 11	**Test Engineers:**	Maleeha Saeed
Objective:	To check the satellite routing		
Product/Version/ Module:	**TCP Performance Using Splitting Over the LEO Satellite Link**		
Environment:	**Windows 98**		
Date:	20th May 2004		
Result: Routing is successfully done. ■ **Pass** ☐ **Fail** ☐ **Not Executed**			

Chapter 7

RESULTS & DISCUSSIONS

Chapter 7

RESULTS & DISCUSSIONS

7 RESULTS AND DISCUSSION

In this section we will discuss our results by using simulation results.

7.1 SIMULATION ENVIRONMENT:

We have conducted our exploration of on-board splitting using the well-known ns-2 (Network Simulator) package. While the simulation tool already includes support for satellites, we have added several enhancements to the satellite channel model by incorporating mobility and shadowing.

7.1.1 LAND MOBILE SATELLITE CHANNEL MODEL:

Here, we use a physical-statistical land mobile satellite channel model. This model uses the geometrical projections of buildings surrounding the mobile terminal. In the model, height and width statistical distributions, and the existence or absence of the direct ray defines the state of the channel. This is either line-of-sight (LOS), when a ray exists, or shadowed, when no ray exists.

7.1.2 ON-BOARD FORWARDING AGENT MODEL:

In this study, we primarily focus upon splitting TCP connections using on-board satellite resources. To simulate this architecture, ns-2 was further enhanced with a new TCP forwarding agent. When attached to a satellite, this agent receives data and acknowledgements from the uplink connection while sending and retransmitting packets on the downlink connection. Because an unrestrained sender will quickly overflow the cache space, the forwarding agent limits the advertised window on the uplink connection to one-third of the total space in the cache, reducing this further as the cache fills. At the same time, the satellite limits the window on the downlink connection to one-third of the total space in the cache, limiting the amount of unacknowledged data sent on the downlink connection.

For this purpose, the standard TCP implementations of ns-2 were modified to allow for an advertised window that changes over the life of the connection. Limiting windows to one-third of the total cache space allows the satellite to store the "in-flight" packets on the uplink and downlink while leaving one window of additional space for backpressure to take effect.

7.2 SIMULATION SCENARIOS:

The results of our simulations fall into four broad categories:

- Single-Hop Unshadowed LEO: Two terminals connect via single LEO satellite.
- Single-Hop Unshadowed double LEO: Two terminals connect via two LEO satellites.
- Single-Hop Source-Shadowed LEO: As in the first case, but the TCP sender suffer from shadowing.
- Single-Hop Sink-Shadowed LEO: As in the first case, but the TCP receiver suffers from shadowing.

In these experiments, the single-hop scenarios include two terminals with the TCP sender in New York and the TCP receiver in San Francisco. The LEO satellite resides at –96W degrees longitude. In the double-LEO cases the TCP sender lies in Rome with the TCP receiver in Los Angeles. In this case, one LEO resides at –95W and one at zero degrees. The shadowed scenarios use the Land Mobile Satellite Channel as described previously. Here, the mobile terminal moves at 2 m/s. In all scenarios, including the shadowing scenarios, both uplink and downlink suffer from uniformly distributed Bit Error Rates (BER) of 10^{-6}, 10^{-7} or 10^{-8}. These bit errors were applied independently to both uplink and downlink for both data and acknowledgments.

In each scenario, we have considered the "goodput" of both TCP NewReno and TCP Westwood with NewReno enhancements. For the unshadowed cases, the throughput measures were determined using 100 FTP sessions lasting 31 seconds each. Because shadowing introduces greater statistical variation, the shadowed cases use 1000 FTP

sessions lasting 31 seconds each. All TCP connections used 1500-byte packets and the TCP window sizes were not limited by the buffer space of the source and sink nodes.

7.2.1 SINGLE-HOP UNSHADOWED LEO:

Figure 1 includes results for the case without shadowing. Here, losses occur as a result of the uniformly distributed BER. In part due to the low capacity of the link, TCP achieves reasonable performance even when faced with the relatively high error rate of 10^{-6}. For the "split" cases, TCP performance increases with the size of the forwarding cache until reaching a saturation point when the cache does not present a limiting factor. The relationship between performance and the size of the cache relates to the optimal window size for the link. In this scenario, both the uplink and downlink have 1 Mbit/s capacity and use 1500-byte packets.

Assuming a LEO one-way delay of 125 ms, the optimal window size for each link becomes$1 \cdot 10^6$ bit/s/(8·512 bits per packet)·2·0.125 s, or 21 packets. In our case, performance levels once the size of the cache rises above three times the optimal window size, or 63.

Figure 2 summarizes the simulation results for the unshadowed single-hop case. For the "Split" case, the forwarding agent used a "saturated" cache size, that is, a cache large enough for the performance to stabilize as seen in Figure 1. From this chart, splitting generally enhances performance especially with increasing bandwidth and error rates. In some cases, splitting achieves over a three-fold increase. Overall, Westwood benefits less from splitting and performs better than NewReno.

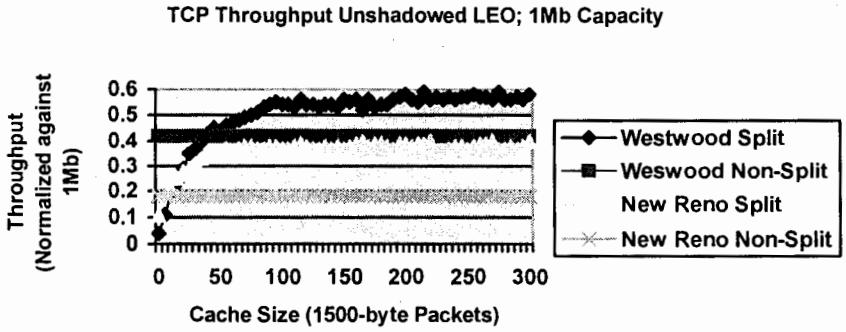


Figure 7.1 (Throughput vs. Cache Size for Unshadowed)

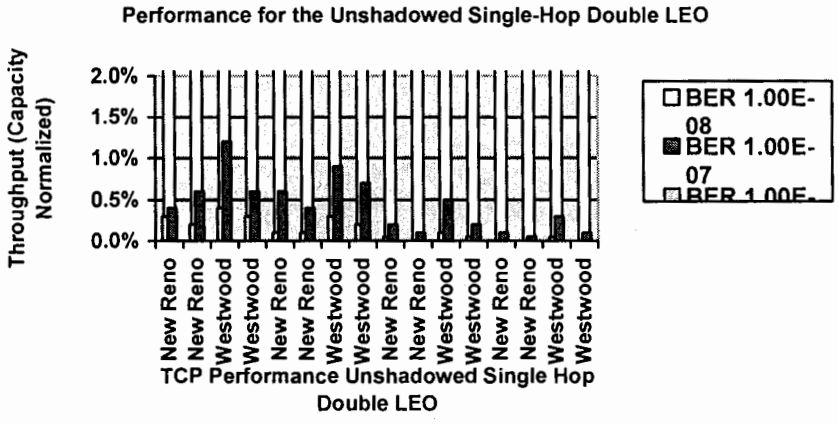


Figure 7.2 (Performance for Unshadowed Scenario)

7.2.2 SINGLE-HOP UNSHADOWED DOUBLE LEO:

Figure 3 shows TCP throughput for the Single-Hop double LEO scenario. Here, with increased delay, the connection achieves lower throughput overall and performance only levels with much larger cache sizes. Splitting also dramatically increases performance for saturated cache sizes.

Figure 4 summarizes the results for the Single-Hop Double LEO cases. For longer delays, TCP Westwood improves performance, with respect to the non split case, and outperforms NewReno substantially more than in the single-hop single LEO case. This holds especially true for large bandwidths.

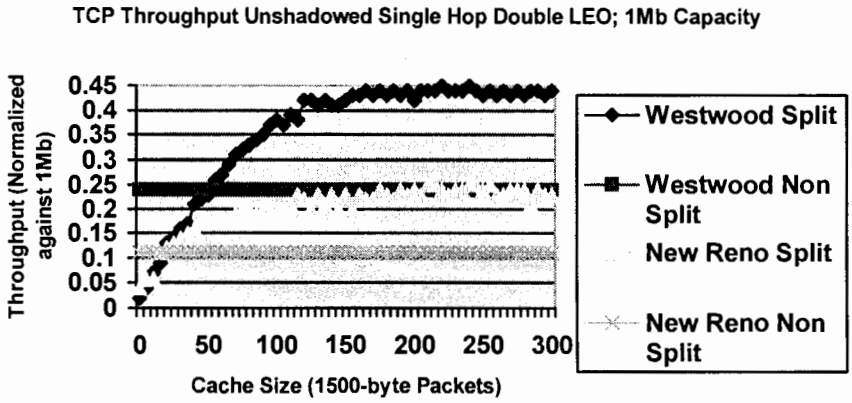


Figure 7.3 (Throughput vs. Cache Size for Unshadowed Single-Hop Double LEO)

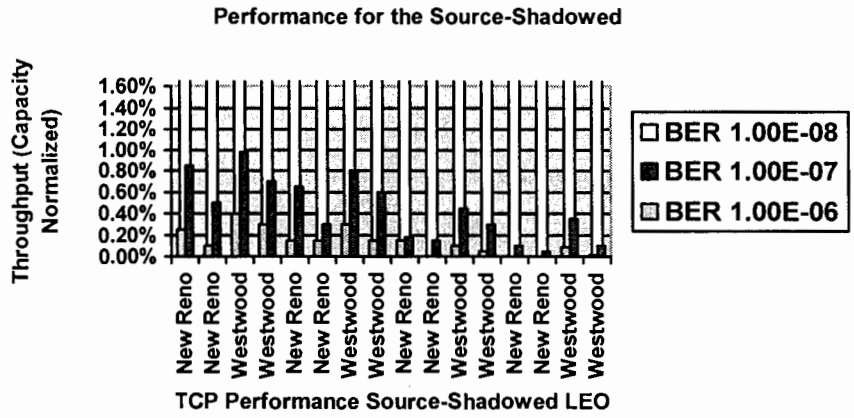


Figure 7.4 (Performance for the Unshadowed Single-Hop Double LEO)

7.2.3 SINGLE-HOP SOURCE AND SINK SHADOWED LEO:

Figure 5 summarizes the results for the source-shadowed terminal, that is, when the TCP senders moves and suffers from shadowing. Figure 6 summarizes the results for the sink shadowed case, that is, when the TCP receiver suffers from mobility and shadowing. These results indicate that sink-shadowing causes more harm to TCP than source shadowing. This may be explained by the fact that packets lost from sink shadowing consume transmission resources and satellite buffer space, while packets lost dues to source shadowing do not travel on the downlink or consume space on the satellite. Thus,

the relative improvement of splitting appears consistent between source- and sink-shadowed TCP connections.

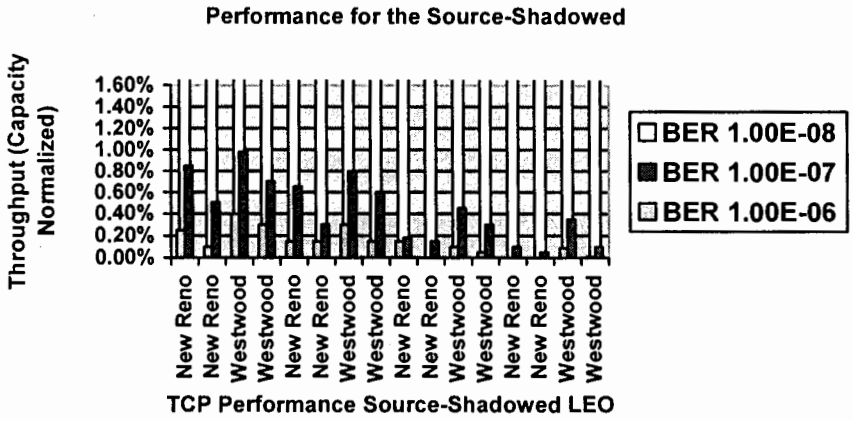


Figure 7.5 (Performance for Source-Shadowed)

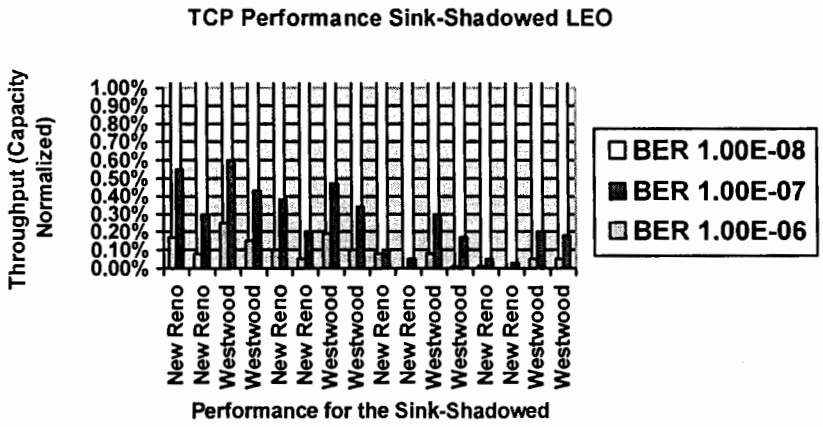


Figure 7.6 (Performance for Sink-Shadowed)

7.3 CONCLUSION:

Finally we have examined the idea of adding transport services to the satellite in order to improve the end-to-end performance of TCP over satellite links. We found that the "splitting" of TCP connections into separate uplink and downlink components yields significant performance improvements. Moreover, as the connection suffers from additional problems from mobility and multiple satellites, the on-board cache makes a greater impact.

Chapter 8

REFERENCES

8 REFERENCES

1. http://searchnetworking.techtarget.com

2. J. Scott Stadler, Jay Gelman; "Performance Enhancement for TCP/IP on a Satellite Channel"; MIT Lincoln Laboratory, Lexington, 1998.

3. Xu Xin, Tang Kai, Ma YiFei; "Performance Analysis of Transport Protocol in Satellite Network"; Institute of Communication Engineering, Nanjing 210016, China.

4. M. Gerla, M. Luglio, R. Kapoor, J. Stepanek, F. Vatalaro, M. A. Vazquez-Castro; "TCP Via Satellite Constellations"; USA, Italy, Spain.

5. http://www.cs.sjsu.edu/faculty/melody/NS2_Tutorial.htm.

6. M. Luglio, J. Stepanek, M. Gerla; "TCP Performance Using Splitting over the Satellite Link"; Computer Science Department, University of California Los Angeles, Boelter Hall, Los Angeles CA; USA.

7. M. Luglio, M. Y. Sanadidi, M. Gerla, Member, J. Stepanck, Student Member; "On-Board Satellite "Split TCP" Proxy"; IEEE.

8. Jing Peng, Peter Andreadis, Claude Belisle, Michel Barbeau; "Improving TCP Performance over Long Delay Satellite Links"; Communications Research Centre, Carleton University; Canada.

9. P. Loreti, M. Luglio, R. Kapoor, J. Stepanek, M. Gerla, F. Vatalaro, M. A. Vazquez-Castro; "Satellite Systems Performance with TCP-IP Applications"; Dipartimeno di Ingegneria Elettronica, Universita di Roma Tor Vergata, Computer Science Department, University of California Los Angeles, Deptarment de

Technologias de las Comunicaciones, Universidad Carlos III de Madrid; IEEE; 2001.

10. Mario Gerla, M. Y. Sanadidi, Ren Wang, Andrea Zanella, Claudio Casetti, Saverio Mascolo; "TCP Westwood: Congestion Window Control using Bandwidth Estimation"; UCLA Computer Science Department, Politecnico Di Torino, Politecnico Di Bari.

11. Joseph Ishac; "Satellite and Terrestrial Network Analysis"; Department of Electrical Engineering and Computer Science, Case Western Reserve University; 2001

12. Milenko Petrovic; "Routing Protocols for Ad Hoc Networks"; Universite York University; May 25,2001.

13. GDSG-APAC-MANET ver3.0; "Mobile Ad Hoc Networks for the Military"; Cisco Systems; 2003.

14. Xiaoyan Hong, kaixin Xu, Mario Gerla; "Scalable Routing Protocols for Mobile Ad Hoc Networks"; Computer Science Department, University of California, Los Angeles.

15. Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz; "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links";1997.

16. Kannan Varadhan; "The *ns* Manual"; UC Berkeley, LBL, USC/ISI, and Xerox PARC; December 6, 2003.

17. Nicolas Christin; "Building ns-2 on Cygwin [versions 2.26 and 2.1b9a(*)]"; UC Berkley; USA.

APPENDIX A

BUILDING NS-2 ON CYGWIN:

Appendix A contains the information about the installation of Cygwin.

Here are relatively simple instructions to get the ns-2 network simulator package to fully build on Cygwin for Microsoft Windows 9x/ME/NT/2000/XP. Make sure you have installed Cygwin with the UNIX text type. (It's the default during the Cygwin install procedure.) Using the DOS text type hasn't been tested, but is likely to lead to a number of errors in the validation tests. It's pretty easy to check which text type you are using, just at the prompt, do a

mount | grep textmode

and if you don't get anything (i.e., it returns an empty string), you should be fine. If the above command does return something, you are quite likely using the DOS text type, and you may be in trouble. You can change the text type by simply rerunning the Cygwin setup program and changing this option, without reinstalling the whole package. Also, make sure your Cygwin installation directory does not contain any spaces. Spaces in the root directory seem to be causing a lot of problems. In particular, `C:\Cygwin` (the default) is a good installation directory, `C:\Program Files\Cygwin` is not.

XFree86 binaries, sources and configuration files, that is, the `XFree86-base`, `XFree86-bin`, `XFree86-prog`, `XFree86-lib`, and `XFree86-etc` packages. They can easily be installed via the Cygwin `setup.exe` program if you haven't already installed them. If you are unfamiliar with XFree86 and/or Cygwin, we recommend you take a look at the Cygwin XFree86 User Guide, and the references therein.

The `make`, `patch` and `diff` utilities as well as `perl` (5.6.1 or above). Again, those can easily be installed via the Cygwin `setup.exe` program if you haven't already installed them. Those should already present in your Cygwin install, but make sure that you have the following set of utilities available: `gcc` (both `gcc` and `gcc-g++` packages - and version

3.x, not `gcc2`!), `awk`, `diff`, `tar`, `gzip`. If some of them are missing (which you can check from the command line by simply issuing the command and see if the shell tells you "command not found"), you must install them with the Cygwin `setup.exe` program before proceeding. This should already be present on your system, but make sure you have the `w32api` package installed on your system. You can verify this by issuing the command

```
cygcheck -c w32api | grep w32api
```

If this command returns an empty string, install the `w32api` package with the Cygwin `setup.exe` program before proceeding. To avoid potential problems, make sure you don't have spaces in your login name and/or the Cygwin installation directory.

In the following, it is important you type all shell commands the way they are given here. In particular, failure to respect the linebreaks (i.e., one command per line) will result in errors.

Put the `ns-allinone-2.26.tar.gz` file in your Cygwin home directory (typically something like `C:\cygwin\home\nicolas`). Put the `nam-1.9.configure` and `ns-allinone-2.26-cygwin.patch` files in your Cygwin home directory as well. Launch Cygwin, start an XFree86 server if you see fit (using `startx`). From your Cygwin (`bash`) prompt, run the following commands (each line corresponds to a separate command):

```
gzip -d -c ns-allinone-2.26.tar.gz | tar xvf -
mv nam-1.9.configure ns-allinone-2.26/nam-1.9/configure
cd ns-allinone-2.26
patch -p0 < ~/ns-allinone-2.26-cygwin.patch
```

You can then just install everything by doing a:

```
./install
```

Update your environment variables:

export NS_HOME=`pwd`

export

PATH=$NS_HOME/tcl8.3.2/unix:$NS_HOME/tk8.3.2/unix:$NS_HOME/bin:$PATH

export LD_LIBRARY_PATH=$NS_HOME/tcl8.3.2/unix:$NS_HOME/tk8.3.2/unix:\

$NS_HOME/otcl-1.0a8:$NS_HOME/lib:$LD_LIBRARY_PATH

export TCL_LIBRARY=$NS_HOME/tcl8.3.2/library

To avoid retyping these commands every time you login you can do the following, after having updated your environment variables as described above:

Type `pwd` at the command prompt.

Note what the shell returns: a directory of the form `/home/XXX/ns-allinone-2.26/` (where `XXX` is your login name).

Update your `~/.bashrc` file by adding the following commands to it:

export NS_HOME=/home/XXX/ns-allinone-2.26/

export

PATH=$NS_HOME/tcl8.3.2/unix:$NS_HOME/tk8.3.2/unix:$NS_HOME/bin:$PATH

export LD_LIBRARY_PATH=$NS_HOME/tcl8.3.2/unix:$NS_HOME/tk8.3.2/unix:\

$NS_HOME/otcl-1.0a8:$NS_HOME/lib:$LD_LIBRARY_PATH

export TCL_LIBRARY=$NS_HOME/tcl8.3.2/library

where you replace `/home/XXX/ns-allinone-2.26/` by whatever `pwd` returned. Make sure you respect the line breaks as shown above. (Optional, run the validation tests with cd ns-2.26; ./validate after having updated your `PATH`, `LD_LIBRARY_PATH`, and `TCL_LIBRARY` environment variables. Note that the regression suite should work on Cygwin, contrary to classical Win32 builds. Some tests may fail on old versions of Cygwin) [17].

APPENDIX B

A Splitting Approach To The Tcp Routing For Satellite Communication

*Maleeha Saeed** *Tauseef-ur-Rehman*** *tauseef@iiu.edu.pk* *Khalid Rasheed**** *drkhalid@iiu.edu.pk*

ABSTRACT

Worldwide usage of the Internet is currently growing at a faster rate, resulting in a huge demand for the transmission of Internet data via satellite. Satellite system features clearly shows that it can also used for mobile Internet applications because it has remarkable ability to cover huge areas for data transmission. In other words satellite gives better approach for broadband transmission. To achieve this successful goal, these satellite systems must show their abilities for the Internet facility with the unforgettable support of Internet protocols e.g. TCP. The failure of TCP is due to high error rate and high delays. By using larger bandwidth in GEO satellite system causes longer delay and more errors. One way to control this problem is to divide TCP connections into segments, or splitting the connection. This paper takes above describe approach, splitting to explore the use of a TCP Proxy on board a satellite for the purpose of enhancing end-to-end TCP performance subdividing end-to-end application paths into separate TCP connections – uplink and downlink. We will clearly explore that an on board Proxy approach provides many advantages for GEO constellations, multiple satellite segments and multicast applications. For better understanding we will later on use the network simulation to show the performance factor.

1 INTRODUCTION

The approach of broadband access to the Internet continues to expand at a faster rate. Many people are started to realize that Internet is basic requirement of their lives so they want to extend this service for other areas of their lives. Satellite is one of the approaches for delivering broadband service. Many satellite systems either currently operating or in the process of development, they have only one goal i.e. how to support applications with greater data-rates. To achieve this goal it is necessary that they must support TCP-based applications. TCP has great importance due to its usage in world-wide-web and client-server applications. But when TCP is used with satellite links it creates many problems. The problems increasingly grow with longer delay, more frequent errors and larger bandwidth. Some solutions need various architectural modifications and others require modification to the protocol mechanisms.

In this paper we will introduce a concept of "splitting" to the satellite link, separating the up and down links with a proxy on board the satellite. Section1 explores the problems of TCP with satellite in details, section2 explores the key features of the on board

* Department of Computer Science, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan.
** Department of Telecommunication, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan.
*** Faculty of Applied Sciences, & Faculty of Management Sciences, International Islamic University, Islamabad, Pakistan.

forwarding agent and section3 describes the TCP splitting concept by using Simulation environment. After it section4 describes the results of simulation.

1.1 TCP AND SATELLITE

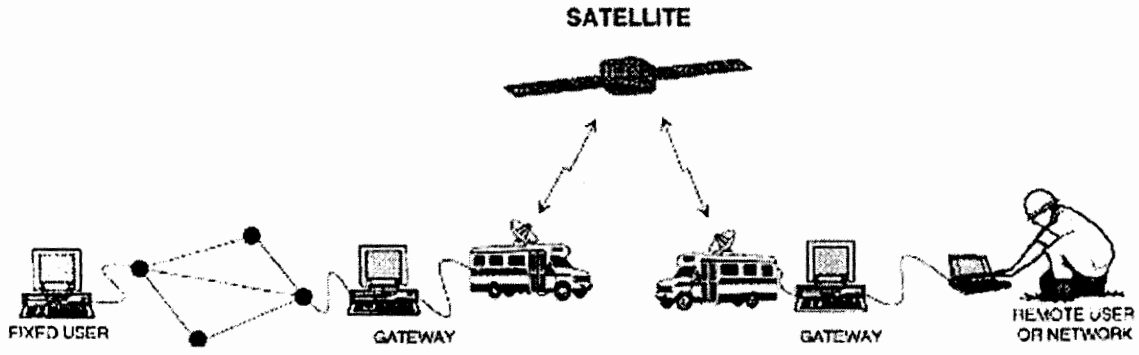
TCP is a connection-oriented, end-to-end reliable protocol. When TCP is using with any communication media including satellite system, provides a reliable stream of data. By exploring the functionality of TCP it will help in judging TCP performance satellite environments. Many features of satellite links can degrade TCP performance [1]. For example large delays change the duration of the "slow start" interval because the slow start mechanism basically depends upon the Round Trip Time (RTT) of the connection [2][3]. In details, the optimal size of the window is depending upon both the RTT and the maximum data rate of the connection. The optimal window size increases due to the large increase in RTT and data rate of the connection. Large RTT also leaves the impact on the rate at the windows grow during congestion avoidance. During slow start procedure, the window size increases by almost one segment for every RTT period. In error-free transmission it will give little impact. But in those cases having large amount of losses, decreases performance.

The loss characteristic of the link causes more problems. Congestion can be judged by amount of lost packets. Where as in wireless links including satellites when TCP connection has frequent losses from link-level errors, the performance of TCP degrade.

1.1.1 PROPOSED SOLUTIONS

Some solutions move towards the protocol modification keeping view the properties of satellite links [4]. These modifications just modify the error-control and flow-control mechanisms to acquire improved performance. Other solutions involve modifications to the architecture of the network. To acquire better performance the intermediaries are processing at TCP end points. According to one technique, subdivide the connection into terrestrial and space segments, or split the connection called splitting. We make changes in TCP flow-control process to introduce New Reno [5] and Peach [6] protocols. SACK [7] is the enhancement of the TCP protocol for error-control mechanism. [8] These enhancements are described according to satellite communication point of view. Finally UCLA introduces TCP Westwood – Fast Recovery algorithm. By using ACK
Inter-arrival times a TCP Westwood sender keeps effective rate of its connection. To control congestion Westwood sender estimate the rate by adjusting its window size and then finally control the losses. Westwood also estimates the bandwidth. The packet loss due to satellite link is not the real indication for congestion. By considering satellite properties we also introduce a new version of TCP called TCP Peach with Sudden Start and Rapid Recovery mechanisms. Sudden Start and Rapid Recovery mechanisms use "Dummy" packets to get knowledge about the characteristics of the network. These packets never create congestion because they have very low priority. These packets
cannot be recovered when they were lost, when acknowledged, the TCP sender increases the window. This causes the window to expand rapidly as compare to slow start and rapid recovery. To limit the dummy packets in TCP Peach, routers should treat IP Precedence

field properly. A number of architectural techniques are also considered to solve TCP's satellite problems. These techniques are generally called "Performance Enhancing Proxies" or PEPs [10]. PEP approach includes segmenting or "splitting" the TCP connection into satellite and non-satellite portions [11] [12]. On ground, splitting uses TCP gateways with satellite access for the transmission of data. These satellite gateways use their own connections that operate on the satellite segment. Between two satellite gateways splitting mechanism uses transport protocol. This transport protocol may be any version of TCP protocol or may be any other different protocol, e.g. a reliable UDP-based transport protocol. A PEP approach intelligently "Capture" a TCP connection and process data packets and acknowledgements. In "Spoofing" the gateway acknowledges the data, which is considered the acknowledgement from the TCP receiver [13] [14]. Spoofing approach divides the connection without the knowing about TCP sender and TCP receiver and sends acknowledgements with the help of gateways. By considering the splitting approach, this paper also gives an idea about architectural solution too. In this case, resources on board the satellite improve end-to-end TCP performance. To enhance TCP performance we should add transport-layer services on the satellite and this way releases some burden of ground nodes. In this case, terrestrial node has ability to use separate TCP connections for sending and receiving purposes from the satellite and satellite just forwards data between these two connections. So, this satellite acts as application-level gateway between these two TCP end points. This enhances the splitting concept by further subdividing the path between TCP end points into separate ground-to-space components. When allocates resources for transport on board, the cost vs. performance need careful consideration. In this given figure at first gateway the TCP protocol is changed into any TCP version and then at second gateway TCP version again is converted back into original TCP, is called Splitting.



2 THE OPERATION OF THE ON-BOARD TCP PROXY

In splitting mechanism we further subdivide the end-to-end path into separate segments between earth and space. This architecture contains two terrestrial satellite gateways. Each gateway uses separate connection with the satellite instead to use a connection with each other. The purpose of "Uplink" connection is to carry data from the ground to the satellite where as "Downlink" carry data from satellite to the ground. Realistically, each connection can act as both uplink and downlink. To analyze clearly here we only discuss those cases in which data packets flow from earth to space and back to earth again, and

acknowledgements flow in the reverse direction. In this way we further reduce the size of TCP's window. But the reduction window size gives impact on the expense of memory and processing on board the satellite. On-board splitting also violets the end-to-end semantics of TCP, if packets are corrupted then the gateway cache remain unrecoverable. To solve this problem, the satellite acknowledges those packets, which are received from the uplink connection only after the first successful data transmission on the downlink connection, that is, without waiting for the acknowledgement from the receiving end of the downlink connection. To check the corruption of the sending packets, satellite uses the TCP checksum or some other technique. Consequently, those packets are unrecoverable which are lost on the downlink as well as found corrupted in memory. In the semantics of TCP, this would represent an effective connection failure.

The primary purpose of satellite designers is the allocation of on-board processing and memory resources. So the usage of on-board proxy needs careful consideration. Here we will describe the advantages of using an on-board transport service beyond simple window reductions:

2.1 STANDARD TCP

A non-standard or non-TCP transport protocol is used in canonical splitting approach between satellite gateways. For on-board case, performance can be improved by using any TCP implementation and enhancement. Non-standard transports use a brilliant approach for flow-control. For congestion, TCP approach can be beneficial for satellite networks. Standard TCP mechanisms can give benefit to satellite system- introducing multiple-hops with packet switching and routing.

2.2 FLEXIBLE TERMINAL DESIGN

Standard TCP does not affect terminal design. By splitting the connection in space, disparate terminal types can also communicate even they are using different transport types. The main disadvantage, use the same TCP stack for all types of connections – wired, wireless and satellite segments. And this disadvantaged terminal might benefit from an advantaged gateway (used for earth-to-space transport).

2.3 SHADOWING

For providing mobile services shadowing creates critical problems for GEO satellites due to higher latitudes. This causes serious and critical problems for TCP performance. To solve shadowing problem use independent error-control mechanisms between earth to space connection, enhances robustness of the transport. To acquire successful transmission it is necessary that both up and down links remain unshadowed. But on-board transport increases the percentage of success. And on-board transports recover many losses very quickly, caused by shadowing.

2.4 MULTICAST

Uplink can use TCP but a multicast downlink has great advantage because satellite has broadcast ability for serving multiple terminals and other satellites.

3 SIMULATION ENVIRONMENT

We have explored on-board splitting using the newly developed ns-2 (Network Simulator-2). Developing simulation tool we always consider satellite features, by considering mobility of satellite and shadowing. We have also added several enhancements to the satellite channel model.

3.1 LAND MOBILE SATELLITE CHANNEL MODEL

In this paper, we use a physical –statistical land mobile satellite channel model as described in [16]. This model contains the mobile terminal, which is surrounded by geometrical projections of buildings. In the model, height and width statistical distributions describe these projections [17] [18], state of the channel is described by the existence or absence of the direct ray. The existence of ray means light-of-sight (LOS) where as shadowed is shown by the absence of ray.

3.2 ON-BOARD FORWARDING AGENT MODEL

We have already discussed splitting TCP connections using on-board satellite resources. To clearly examine this architecture, we enhanced ns-2 with a new TCP forwarding agent. When attached to a satellite, this agent receives data and acknowledgements through the uplink connection and then sends and retransmits packets on the downlink connection. Because sender will quickly overflow the cache that is why the forwarding agent controls the window size on the uplink connection to one-third of the total space in the cache, reducing this further as the cache fills. At the same time, the satellite limits the window of the downlink connection to one-third of the total space in the cache, limiting the amount of unacknowledged data sent on the downlink connection. Due to this, the standard TCP implementations of ns-2 were changed to support an advertised window that changes for connection. We can support the satellite to store the "in-flight" packets on uplink and downlink by limiting the window size to the one-third of the total space of cache but we have to leave one window of additional space for backpressure.

4 SIMULATION SCENARIOS

Simulation results have four categories:

1. Single-Hop Unshadowed LEO: Two terminals with a single satellite.
2. Single-Hop Unshadowed double LEO: Two terminals connect via two LEO satellites. The LEOs communicate using an ISL.
3. Single-Hop Source Shadowed LEO: Same to first case but in this case TCP

sender suffers from shadowing.

4. Single-Hop Sink Shadowed LEO: Same to first case but in this case TCP receiver suffers from shadowing.

In these cases, the single hop scenarios include two terminals, the TCP sender in New York and the TCP receiver in San Francisco. The LEO satellite resides at -96 W degrees longitude. In the double LEO cases the TCP sender in Rome and the TCP receiver is kept in Los Angeles. In this case, one LEO resides at -95 W and one at zero degrees. The shadowed cases use the Land Mobile Satellite Channel. The speed of mobile terminal is 2m/s. In all cases, both uplink and downlink connections suffer with bit errors having Bit Error Rates (BER) of 10^-6, 10^-7, or 10^-8. These bit errors are not for only data but also for acknowledgements. In each scenario, we have considered the concept of "good put" for both TCP New Reno and TCP Westwood with New Reno enhancements. For the unshadowed cases, we have measured through put by using 100 FTP sessions lasting 31 seconds each. Shadowed cases show great variation, use 1000 FTP sessions lasting 31 seconds each. TCP connection uses 1500-byte packets and the TCP window size is not limited due to the buffer space of the source and sink nodes.

4.1 SINGLE –HOP UNSHADOWED LEO

Figure 1 shows results that would not have shadowing effect. Losses occur due to uniformly distributed BER. Although having high error rate of 10^-6 TCP achieves reasonable performance due to low capacity of the link. For the "split" cases, TCP performance increases when the size of the forwarding cache increases. The performance and the size of the cache are related to the optimal window size for the link. Both links of this case has 1 M bit/s capacity and use 1500-byte packets.

Figure 2 shows results for the unshadowed single-hop case. For the "split" case, the forwarding agent used a "Saturated" cache size, that is, a cache having enough capacity for the performance to stabilize. From this, we analyzed that splitting mechanism increases the performance especially with increasing bandwidth and error rates. In some cases, splitting achieves remarkable increase. On the whole, splitting gives less benefit to Westwood but still Westwood perform better than New Reno.

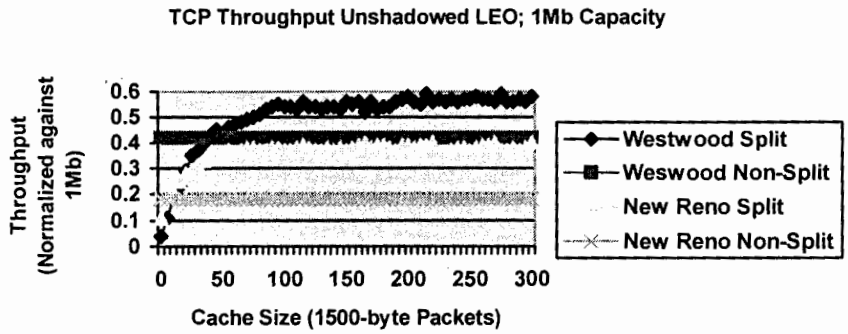


Figure 1 (Throughput vs. Cache Size for Unshadowed)

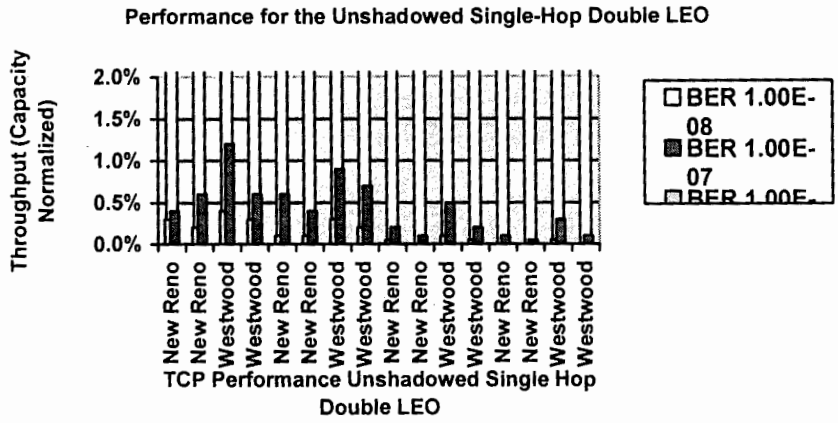


Figure 2 (Performance for Unshadowed Scenario)

4.2 *SINGLE –HOP UNSHADOWED DOUBLE LEO*

Figure 3 shows TCP through put for the single – hop double GEO scenario. In this case, when through put decreases due to increased delay and performance only levels with much larger cache sizes. Splitting mechanism also increases performance for saturated cache sizes.

Figure 4 is diagramed to show the results of single – hop double GEO cases. In non-split case, due to longer delays, TCP Westwood improves performance and New Reno performs outstandingly when there is more single – hop single GEO case. But there should be large bandwidth.

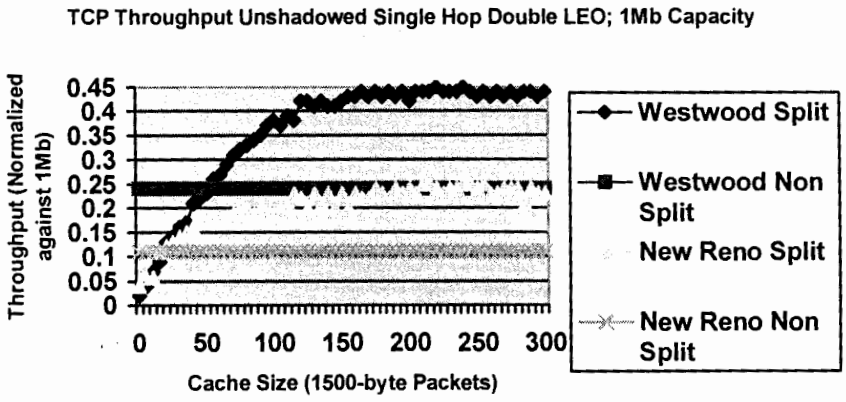


Figure 3 (Throughput vs. Cache Size for Unshadowed Single-Hop Double LEO)

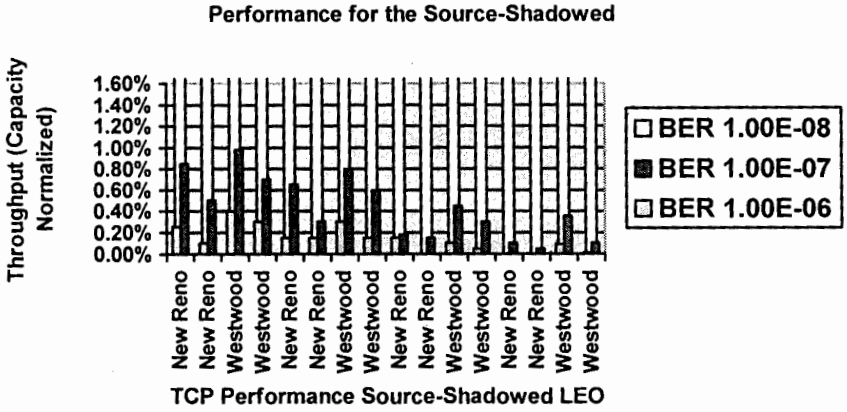


Figure 4 (Performance for the Unshadowed Single-Hop Double LEO)

4.3 SINGLE – HOP SOURCE AND SINK SHADOWED LEO

Figure 5 shows the results of source shadowed terminal scenario – when the TCP senders move and suffer from shadowing.

Figure 6 shows that shadowed are caused due to the movement of TCP receivers. These results clearly show that sink shadowing causes more problems than source shadowing. Because when packets are lost due to sink shadowing then for their retransmissions they consume transmission resources and satellite buffer size. Therefore, the relative improvement of splitting appears consistent between source and sink-shadow TCP connections.

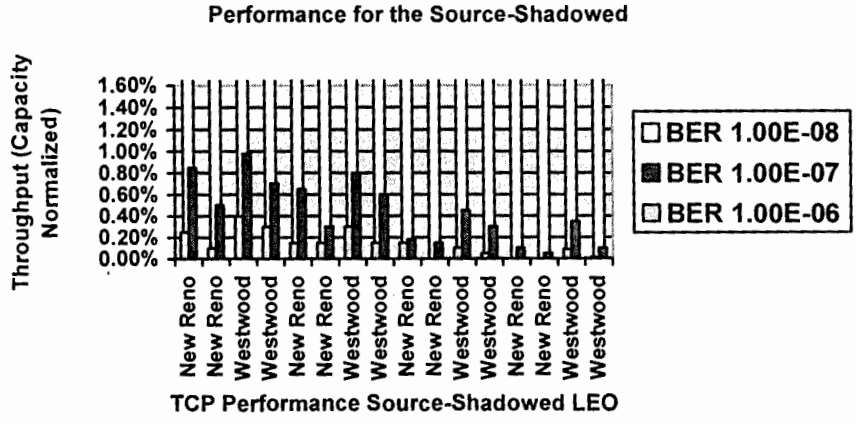


Figure 5 (Performance for Source-Shadowed)

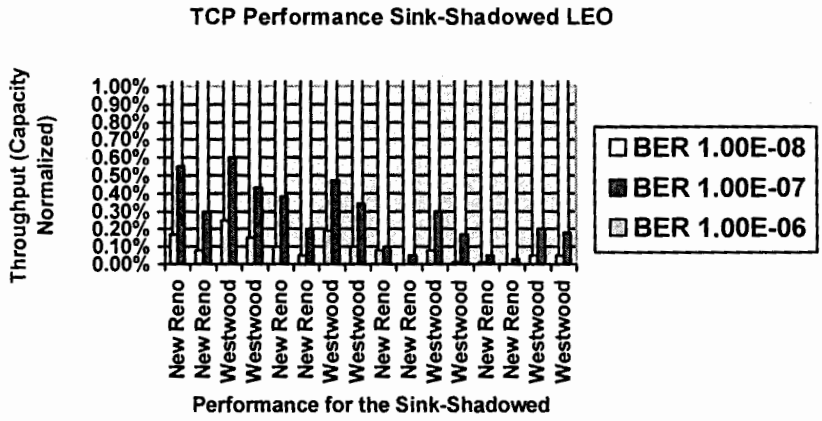


Figure 6 (Performance for Sink-Shadowed)

5 CONCLUSION

Finally we have examined the idea of adding transport services to the satellite in order to improve the end-to-end performance of TCP over satellite links. We found that the "splitting" of TCP connections into separate uplink and downlink components yields significant performance improvements. Moreover, as the connection suffers from additional problems from mobility and multiple satellites, the on-board cache makes a greater impact.

6 REFERENCES

[1] C. Partridge, T. J. Shepard, "TCP/IP Performance over Satellite Links", IEEE Network, September-October 1997, pp. 44-49.
[2] V. Jacobson, "Congestion Avoidance and Control", Proc. of SIGCOMM '88, 1988, ACM.
[3] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, Jan. 1997.
[4] M. Allman, D. Glover, L. Sanchez, "Enhancing TCP over Satellite Channels using Standard Mechanism", RFC 2488, January 1999.
[5] S. Floyd, T. Henderson, "The New Reno Modification to TCP's Fast Recovery Algorithm," RFC 2582, April 1999.
[6] I. F. Akyildiz, G.Morabito, S. Palazzo, "TCP-Peach: a new congestion control scheme for satellite IP networks", IEEE/ACM Transactions on Networking Volume 9, Issue 3 (2001), Pages 307-321.
[7] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996
[8] M. Allman (editor), "Ongoing TCP Research Related to Satellites", RFC 2760, Feb. 2000.
[9] S. Mascolo, C. Casetti, M. Gerla, S. S. Lee, and M. Sanadidi, "TCP Westwood:

Congestion control with faster recovery", UCLA CS-Technical Report #200017, 2000.
[10] J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, "Performance Enhancing Proxies intended to mitigate link-related degradations", RFC 3135, June 2001.
[11] T. Henderson and R. Katz, "Transport protocols for Internet-compatible satellite networks," IEEE Journal on Selected Areas in Communications, Vol. 17, No. 2, pp. 345-359, February 1999.
[12] M. Karialiopoulos, R. Tafazolli, B. G. Evans, "TCP Performance on Split Connection GEO Satellite Link", 19th AIAA International Communications Satellite Systems Conference, AIAA19, session 16, vol. 2, April 17-20, 2001, Tolouse, France.
[13] J. Ishac and M. Allman, "On the performance of TCP spoofing in satellite networks", Proceedings of Milcom 2001, Tysons Corner, McLean, VA, USA, October 28-31, 2001.
[14] A. Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," Proc. 15th Intern. Conference on Distributed Computing Systems (ICDCS), May 1995.
[15] "Network Simulator (NS-2)", http://www.isi.edu/nsnam/ns/
[16] P. Loreti, M. Luglio, R. Kapoor, J. Stepanek, M. Gerla, F. Vatalaro, M. A. Vazquez-Castro, "LEO Satellite Systems Performance with TCP-IP applications", Proceedings of Milcom 2001, Tysons Corner, McLean, VA, USA, October 28-31, 2001, session U24.
[17] P. Loreti, M. Luglio, R. Kapoor, J. Stepanek, M. Gerla, F. Vatalaro, M. A. Vazquez-Castro, Throughput and Delay Performance of Mobile Internet Applications Using LEO Satellite Access, International Symposium on 3G Infrastructure and Services, Athens, GR, 2-3 July, 2001, pp.68-73.
[18] M. Gerla, R. Kapoor, J.Stepanek, P. Loreti, M. Luglio, F. Vatalaro, M. A. Vazquez-Castro, Satellite Systems Performance with TCP-IP Applications, Tyrrhenian International Workshop on Digital Communications, IWDC 2001, September 17-20, 2001, Taormina, Italy, pp. 76-90.

MESSAGE DISPLAY

Message number 2

2

neomail-trash Move

All headers

Date: 1/14/2005 07:05:30 -0800
From: Helda@oitj.info
To: tauseef@iiu.edu.pk, maleeha_saeed@hotmail.com
CC: stauseef@yahoo.com
Subject: Confirmation of Paper

Dear Dr. Rehman!

Hi,

I have rechecked our record and the paper with following specifications has been accepted for publication, but as yet we have not recieved the ammended copy. The specification (as per our record) based on the original submission are:

Title: A Splitting Approach to the TCP Routing for Satellite Communication
Authors: Maleeha Saeed, S.Tauseef ur Rehman and Khalid Rashid

By the same token, we are still waiting for your final copy. As per the last mail the only changes required were related to formatting.

Regards
Gwen Helda
Associate Editor
OITJ

NeoMail version 1.27

Camera ready copy has been submitted. The concerns were related to formatting only. Subsequent mail will be provided.

S. Rehman