Development of an ns-3 Based Network Simulator for Space Telemetry

by

Md Monjurul Islam Khan

A Thesis submitted to The Faculty of Graduate Studies of

The University of Manitoba

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg

April 2017

Abstract

A satellite communication module is developed to generate network variables, which are required to simulate the communication channel between a workstation on the Earth and the International Space Station (ISS). The main focus is to use this simulator to study the feasibility of transmitting data packet between a location on Earth and the ISS, an orbiting research laboratory, for different applications including teleoperation. Thus, a simulation platform is needed to mimic the actual communication scenario. Therefore, the performance of a remotely–controlled system could be examined using the developed satellite communication module, before final implementation in the field, which is time consuming and expensive. The Network Simulator 3 (ns-3) is employed to develop the satellite communication module. Values of network parameters, obtained from simulation using the developed module, are quantified when the ISS rotates around the Earth. To show the proof of concept, the simulator is tested for a haptic-enabled teleoperated system. At the master site, the simulation program of a 3–degree–of–freedom (DOF) haptic device is used, and its position and force components are transferred to the slave site on the ISS. A packet data, containing the information of the master site, is transmitted through a simulated communication channel. Results validate that, the simulator is capable of transferring the data packet, as the force/position signal, received at the slave site followed the characteristics of the signal transmitted from the master station on earth.

Acknowledgments

At first, from the core of my heart, I would like to acknowledge the blessings and guidance of Almighty ALLAH.

I am grateful to my thesis supervisor Professor Ekram Hossain, for his continuous guidance and support during my study period at University of Manitoba. I would also like to express my gratitude to Dr. Yaser Maddahi from Project neuroArm ath the University of Calgary, for his constant support throughout the project.

I appreciate, all the members from our research group, who motivate me in the time of failure or depression. I would like to thank, my old friend, Monowar Hasan, for facilitating my admission into the University of Manitoba. I appreciate the guidance, I got from Md Monirul Islam, regarding coding problem in C++, while working on my thesis. I can not express my gratitude in words to Amy Dario, for her kind assistance in each and every academic and administrative problems.

I appreciate the valuable advice from my thesis committee members, Dr. Noman Mohammed, and Dr. Robert D. McLeod. Their selfless advice and feedback have significantly enhanced the quality of my research.

Finally, I must express my profound gratitude to my mother, sister, and to my fiance, Israt Jahan for providing me with constant support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Dedication

This work is dedicated to my mother, a strong, dedicated and gentle soul who taught me to trust solely in ALLAH, believe in hard work and be kind to others. Without her I would not be here.

Table of Contents

List of Abbreviations **xi**

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Statement of the Problem . . . 3
- 1.3 Related Work . . . 4
- 1.4 Development Methodology . . . 6
- 1.5 Contributions of the Thesis . . . 8
- 1.6 Organization of the Thesis . . . 9

2 Development Preliminaries **10**
- 2.1 Selection of a Proper Simulation Platform . . . 10
- 2.2 Selection of the Transport Protocol . . . 13
  - 2.2.1 Limitations of TCP in Space Communication . . . 13
  - 2.2.2 Transport Protocol for Satellite Communication . . . 15
- 2.3 Explore Satellite Communication Strategy of ISS . . . 18
- 2.4 Determination of the Satellite Link Budget . . . 22
  - 2.4.1 Free Space Path Loss . . . 23
  - 2.4.2 Atmospheric Attenuation . . . 24
  - 2.4.3 Noise Temperature . . . 24
  - 2.4.4 Antenna Gain . . . 26
  - 2.4.5 Total Link Equation . . . 26
  - 2.4.6 Minimum Detectable Signal . . . 26
  - 2.4.7 Bandwidth and Packet Error Rate . . . 27
- 2.5 Implementation in ns–3 . . . 29
  - 2.5.1 Key Abstractions . . . 29
  - 2.5.2 Object Model . . . 31
  - 2.5.3 Type and Attribute System . . . 32
  - 2.5.4 Object Aggregation . . . 33
  - 2.5.5 Smart Pointers . . . 34
  - 2.5.6 Tracing System . . . 34
  - 2.5.7 Callback . . . 35

2.5.8 Satellite Network Device 35
2.5.9 Two Dimensional System Model 36
2.6 Implementation of Three Dimensional Orbital Propagation Model . . 36
2.6.1 Perturbing Forces and SGP Model 37
2.6.2 TLE and State Vectors 37
2.6.3 Coordinate System 38

3 Implementation in ns–3 43
3.1 Satellite Point to Point Module 43
3.2 Creating a New Module in ns–3 44
3.2.1 Module Layout 44
3.2.2 Create the Module Skeleton 45
3.2.3 Merge with the Existing System 46
3.3 Classes in the New Module 48
3.3.1 SatellietPointToPointNetDevice Class 49
3.3.2 SatellietPointToPointChannel Class 50
3.3.3 SatellitePhysicalClass 50
3.3.4 Use of SatellitePointToPoint Module 51
3.4 Incorporating Satellite Orbital Position 52
3.4.1 Implementing the 2D Model 52
3.4.2 Implementing the 3D Model 52
3.5 Transmitting Data Over the Satellite Channel 53

4 Simulation Results 55
4.1 Employing the Simulator Between Master-Slave Sites 55
4.2 Experimental Procedure 59
4.3 TCP Variants 59
4.4 Simulation Setup 60
4.5 Results 61
4.5.1 Analysis of Results for Path 1 61
4.5.2 Analysis of Results for Path 2 63
4.5.3 Analysis of Results for Path 3 63
4.6 Observations 63

5 Conclusion and Future Work 74
5.1 Concluding Remarks 74
5.2 Future Research Directions 75
References 77

A Appendix A 83

B Appendix B 86

C Appendix C **109**

List of Figures

1.1 In the conceptualized platform, the master site (on earth) sends information to the slave site (in the ISS) and the slave station in the ISS sends some information back to the experienced team on earth. . . . 4

2.1 Each circle represents a Tracking and Data Relay Satellite (TDRS). Satellites placed outside the orbit are either non-functional or in testing phase. The inset shows a typical TDRS. 20
2.2 Frequency bands used in International Space Station for communication with the Earth. S-band is used for transmitting command and other data and Ku-band is used for transmitting high payload data like video. 20
2.3 Two dimensional system model, where ground station is placed with a distance of Earth's radius and Tracking and Data Relay Satellites (TDRS) are placed 120° apart from each other and $35,787$ km away from ground station. 21
2.4 Longitude and latitude to Earth Centered Inertial (ECI) coordinate systems conversion. 40

3.1 Satellite communication module consists of network device and channel. 44

4.1 Positional and force components generated at the master site. The data packet contains 3 positional components ($x_m,\ y_m,\ z_m$) and 3 force components ($F_x,\ F_y,\ F_z$). 56
4.2 Positional components ($x_m,\ y_m,\ z_m$) at the master haptic hand–controller along $X_R,\ Y_R$ *and* Z_R. 57
4.3 Force components at the master haptic hand–controller, ($F_x,\ F_y,\ F_z$). 58
4.4 Changes in congestion window in path 1 which is ground station to TDRS-1 to ISS. 62
4.5 Change of RTT in path 1 which is ground station to TDRS-1 to ISS. 64
4.6 RTT histogram in path 1 which is ground station to TDRS-1 to ISS. 65
4.7 Changes in congestion window in path 2 which is ground station to TDRS-1 to TDRS-2 and then ISS. 66

4.8 Change of RTT in path 2 which is ground station to TDRS-1 to TDRS-2 and then ISS. 67
4.9 RTT histogram in path 2 which is ground station to TDRS-1 to TDRS-2 and then ISS. 68
4.10 Changes in congestion window in path 3 which is ground station to TDRS-1 to TDRS-3 and then ISS. 69
4.11 Change of RTT in path 3 which is ground station to TDRS-1 to TDRS-3 and then ISS. 70
4.12 RTT histogram in path 3 which is ground station to TDRS-1 to TDRS-3 and then ISS. 71

List of Tables

1.1 Work-flow of development of the space simulator-from concept to implementation 7

2.1 Available simulation platforms 12
2.2 List of TDRS 19
2.3 The nodes in ns–3 and their representation in 2D and 3D model. As TDRS-9 replaces TDRS-1 therefore TDRS-1 in 3D model will represent TDRS-9. This assumption is also applicable to other nodes 41

4.1 Simulation parameters used in sending HTS data from Earth to the ISS 60
4.2 Changes in round trip time 72
4.3 Throughput for three different paths employing different transport protocols. 72

List of Abbreviations

ns-3	Network Simulator 3
NASA	National Aeronautics and Space Administration
NORAD	North American Aerospace Defense Command
CCSDS	Consultative Committee for Space Data Systems
SCPS	Space Communications Protocol Specifications
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
RTT	Round Trip Time
TLE	Two Line Element
TDRS	Tracking and Data Relay Satellite
TDRSS	Tracking and Data Relay Satellite System
GEO	Geostationary Satellite
LEO	Low Earth Orbit Satellite
ISS	International Space Station
NISN	NASA's Integrated Services Network
MCC-H	Mission Control Center at Houston
WSGT	White Sand Ground Terminal
LoS	Line-of-Sight
BDP	Bandwidth Delay Product

Chapter 1

Introduction

Space technology has advanced drastically and space exploration has become a growing word. In 1969, The National Aeronautics and Space Administration (NASA) launched Apollo 11 spaceflight, the fifth manned mission of NASA's Apollo program, to land the first humans on the Moon. Their mission was to collect lunar materials for experimental investigations. Spirit and Opportunity mobile robots also completed their 3–month missions on Mars in 2004. These vehicles were part of NASAs Mars Exploration Rover project that continued their search for geological clues to confirm whether Mars formerly had environments wet enough to be hospitable to life [1, 2]. Presently, International Space Station (ISS), an orbiting research laboratory, is accepted as the main symbol of a continuous collaboration between different nations. Canada's contribution to the station, for instance, is the Mobile Servicing System (MSS) – a sophisticated robotic suite that assembled the station, and was developed for the Canadian Space Agency (CSA) by MacDonald Dettwiler Space and Advanced Robotics (MDA) [3]. The MSS is composed of three main components: the Space Station Remote Manipulator System (SSRMS) that is known as Canadarm2, the Mobile Remote Service Base System (MBS) and the Special Purpose Dexterous Manipula-

tor (SPDM) also known as Dextre or Canada hand. In addition to the MSS, there are several ISS-servicing robotic systems, including the Japanese Experiment Module Remote Manipulator System (JEMRMS) that was designed and fabricated built by the Japan Aerospace Exploration Agency (JAXA). The JEMRMS robotic setup was built to support experiments conducted on the Exposed Facility (EF) of the Japanese Experiment Module (JEM) [4]. This robot includes a 6–DOF, 10–meter long main arm, and a 6–DOF, 2–meter long small fine arm (SFA) designed to perform dexterous tasks [5]. The European Space Agency (ESA) also developed a 7–DOF robotic arm that is attached to the Russian segment of the ISS [6]. All the remote manipulators present in the ISS are controlled by the astronauts living there.

1.1 Motivation

Astronauts are living in the ISS for performing various researches, which should be conducted in space. The experiments are performed in space require human presence. Moreover, there exist many activities performed in the ISS that require high level of expertise, including maintenance of electromechanical systems, medical treatments, and technical troubleshooting etc. If there exist an assistive system such as a robotic arm which can be controlled from Earth, then the physical presence of human in ISS would not be required anymore. Analyzing the feasibility of remote controlling a system from Earth in the ISS is required. Till date not a single open source simulation platform supports satellite communication. Therefore, an extensive research is required to determine the feasibility of haptic teleoperation.

1.2 Statement of the Problem

A robotic arm controlled by a human from long distance is known as long–distance teleoperation in which the information between the master and slave sites is transferred through a communication channel. In the proposed platform, the master site is located on earth and the slave site is suited inside the ISS, as depicted in Fig. 1.1. In other words, some data such as position, velocity, and/or force are sent by the master hand–controller (on earth) to the manipulator located at the ISS, and some are sent back from the ISS to earth. Fig. 1.1 also illustrates how information flows between master and slave sites in a bilateral fashion. Therefore, a study is required to determine the feasibility of tele-presence between the Earth and the ISS.

The feasibility of long–distance space telemetry could be done by placing an actual robotic arm on the ISS, and control the manipulator from earth to verify its workability over the satellite communication. That option is not feasible, as placing a robotic arm inside the ISS requires time, infrastructure, and money. Therefore, the focus of this study is, to develop a simulator that simulates the network scenario between the Earth and the ISS. The developed simulator will enable us to study the feasibility of any communication between our planet and the ISS. Thus, the feasibility of remote operation could be investigated. Moreover, the simulator will also help universities and educational centres adopt the concept of *Space Communication* in an economical manner. The main goal of this work is, to develop the middle block in Fig. 1.1, *i.e.*, developing a space simulator that is capable of simulating the behaviour of the communication channel between the Earth and ISS.

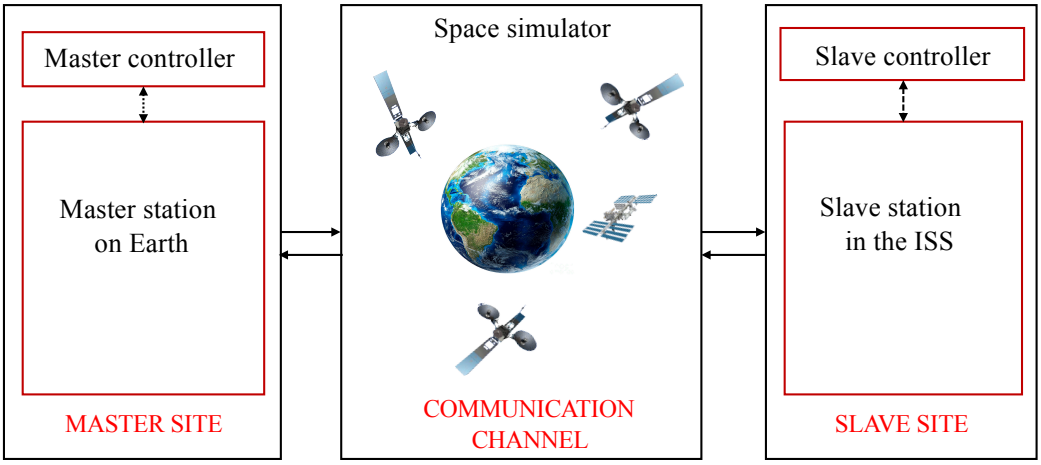


Figure 1.1: In the conceptualized platform, the master site (on earth) sends information to the slave site (in the ISS) and the slave station in the ISS sends some information back to the experienced team on earth.

1.3 Related Work

There have been several attempts to develop techniques, by which activities in ISS could be conducted remotely from our planet. To name a token, an expert team on the Earth assisted astronauts on ISS to accomplish a task [7]. Recently, the ESA conducted two experiments (Haptics–I and Haptics–II) to investigate disability of the performance of a remotely–controlled robotic arm, and observe how a human operator feels when they are in a robot's shoes. Haptics–I was an experiment designed to investigate the remote control of robots on the ground from the ISS. The experiment was a simple–looking lever that can be moved freely to play simple Pong–style computer games [8]. In Haptics–II project, the ESA and the Robotics Institute at the Delft University of Technology performed some peg–in–hole (with 1/6th of a millimetre clearance) experiments, and remotely controlled a wheeled mobile robot from orbit inside ISS [9]. Haptics–II experiment allowed astronauts to control the rovers arms and wheels to conduct the tasks by means of force feedback. In Haptics–II, ma-

nipulation of the peg was performed on the Earth that is simpler than manipulation in the ISS due to micro–gravity environment, where things seem to be weightless. As the ISS is always in free falling state so objects on the ISS seems to be weightless. In the term micro–gravity micro represents very small. Therefore, the experiment, which was done in Haptics–II is an opposite way of the experiment which we aim to do in this study, *i.e.,* the task is conducted in the ISS instead of earth [8].

A number of other experimental space manipulators have also been developed and successfully implemented in space. The German Aerospace Center (DLR) developed the Robot Technology Experiment to verify different teleoperation modes such as on–board teleoperation, teleoperation from the ground, and sensor–based offline programming [10]. The DLR developed the Robotics Component Verification on the ISS robotics experiment [11]. NASA developed Robonaut 1, a dexterous robot to assist astronauts during Extra Vehicular Activities (EVA) tasks with the ability to work with existing EVA tools and interfaces in high–fidelity ground–based test facilities [12]. A team from NASA and General Motors also developed a second generation of this system named the Robonaut 2, a dexterous robot with significant technical improvements compared to Robonaut 1 [13]. Robonaut 2 is recognized as the first humanoid robot in space [14]. All the experiments, done by the space agencies, used their own simulation software which is either closed source or commercial software. A team from University of Timisoara, proposed a design solutions for a haptic arm exoskeleton which could be used remotely in space [15].

The Open Source Satellite Simulator (OS3) [16] is a satellite simulator developed by the Technical University of Dortmund. The simulation platform used in developing OS3 is OMNet++ in conjunction with the INET framework, an open–source OMNeT++ model suite which provides component classes for wired, wireless, and mobile

networks. Developers of the OS3 claimed that it is modular and easy to use; however, the focus was on developing the satellite mobility, satellite constellations along with the weather data and channel models. To date, no satellite communication protocol was incorporated in OS3.

Digital Video Broadcasting (DVB) organization published a book on satellite return link specification which is known as DVB-S2/DVBRCS2. European Space Agency (ESA) uses DVB-S2/DVBRCS2 for satellite communication, as DVB-S2/DVBRCS2 is not implemented in publicly available simulators. A group of researcher developed a simulator to implement the DVB-S2/DVBRCS2 in combination with Network Simulator 3 (ns–3) and named their simulator as Satellite Network Simulator 3 (sns–3) [17]. The developers are still working on sns–3 and sns–3 is not publicly available yet. Moreover, the sole purpose of sns–3 is to implement DVBS2/DVBRCS2 in ns–3 along with satellite mobility. In [17], the investigators presented an overview of their proposed end–to–end architecture, which was the motivation of our work to develop an architecture in ns–3 from scratch.

1.4 Development Methodology

The satellite module development is done in several steps depicted in Table 1.1. In the first step, an extensive survey is performed on the available simulation platforms, to help the authors, to choose a proper simulation platform to develop the simulator. Afterwards, a study is conducted to identify the transport protocol used in the satellite communication, followed by investigating communication strategy of the ISS to earth. Note that, the ISS is a multinational satellite system; therefore, various modules of ISS, communicates with the earth, use different techniques. Next, a research on satellite communication link budget is completed to quantify the values of required

Table 1.1: Work-flow of development of the space simulator-from concept to implementation

Step1: Select a proper simulation platform
- Study existing simulation platforms
- Explore flexibility and extensibility of the studied platforms
- Choose a widely accepted platform in academia
Step2: Select the transport protocol
- Investigate transport protocols used in satellite communication
- Research existing protocols
- Select a transport protocol to incorporate in the simulator
Step3: Explore satellite communication strategy of the ISS
- Investigate communication strategy of the ISS
Step4: Determine the satellite link budget
- Research the satellite link budget equation
- Derive a link budget equation for the proposed application
- Collect the parameters needed to communicate with the ISS
Step5: Implement in ns–3
- Develop the satellite network module in the simulator
- Incorporate delay and link equation in the model
- Simulate two dimensional mobile setup
- Send mock data and analyze
Step6: Implement three dimensional orbital propagation model
- Investigate satellite orbital propagation
- Incorporate satellite trajectory information in the simulator
- Send haptic application data and analyze

parameters for the proposed simulator. In next step, the satellite network device and the channel are implemented in ns–3. The developed module is tested with existing TCP variants, followed by implementing a two dimensional satellite communication scenario. Finally, satellite orbital propagation methodology is studied and incorporated in the developed simulator. The transparency of data communication between the master and slave sites is then validated, when the information, obtained from the simulation of two teleoperated systems, are pushed into the simulator. Validations are conducted while the ISS is orbiting along it's actual trajectory, and the delays

and throughput are also analyzed.

1.5 Contributions of the Thesis

This research aims at developing a satellite communication module to reflect realistic network between the Earth and the ISS, therefore transmit the information between both locations. Thus, the performance of a teleoperated system could be tested using the developed module while there is a simulation of real communication channel, and then be implemented in real field. Therefore, the contributions of this research are:

- Study existing network simulation platforms, and propose a proper platform to develop the simulator.
- Research transport protocols, and determine the transport protocol used in the satellite communication.
- Inquire and find the communication strategy between the Earth and the ISS.
- Study link budget calculation in satellite communication, and determine the link budget used in communicating with the ISS.
- Develop a satellite module in ns–3, to incorporate satellite link budget equation and study the delay effect on the haptic communication.
- Test developed satellite module while a set of data packets, obtained from the simulation of a teleoperated platform, being pushed into the communication channel. Results will validate the possibility of sending and receiving the information between the Earth and the ISS.
- Review satellite orbital propagation, and incorporate actual ISS trajectory in the developed simulator.

It is believed that, the simulator developed in this study, will provide a platform for future studies as, to the best knowledge of the author, there is no open source packet level satellite simulator is currently available.

1.6 Organization of the Thesis

The core contents of this thesis are organized in three chapters. A brief description of those chapters are given below.

- Chapter 2 depicts the simulator development preliminaries such as, selection of an appropriate simulation platform, discussion on the transport layer protocol, formulation of the link budget equation, discussion on ns-3 basics and orbital propagation techniques etc.
- A detailed description of satellite point to point module development is presented in Chapter 3.
- The results, generated from the simulation program, are discussed in Chapter 4 while sending the application data generated by a haptic application.
- Concluding remarks and several future research directions are outlined in Chapter 5.

Chapter 2

Development Preliminaries

Development of a satellite communication module requires extensive research on several topics enlisted in Table 1.1. Those core concepts are presented in this chapter.

2.1 Selection of a Proper Simulation Platform

In the field of computer networks, network simulators played an important role. Network simulators provide a set of libraries, for modeling arbitrary computer network by specifying both the communication channels as well as the behavior of the network node. For example, before implementing a new routing protocol in the real network, network simulators are used to investigate the characteristics of the proposed routing protocol in the simulated world first. With the help of simulator, the routing behavior can easily be studied for various network topologies as the network topology is simply a set of simulation parameters. There are two kinds of simulation paradigm exist. Those paradigms are time driven simulation (TDS) and discrete event–based simulation (DES). In time driven simulation, all the events are triggered in their scheduled time, while in the discrete event–based simulation paradigm, simulator maintains a

event queue sorted by the scheduled event execution time. The simulator executes all the events from the queue. If one event is scheduled at 10 s, and another one is scheduled at 20 s, then in time driven simulation after finishing first event at 11 s, the simulator will wait another 9 s before executing the second scheduled event. But in DES, simulator jumps from 11 s to 20 s for executing the second event without waiting 9 s. However, currently available network simulation platforms are developed based on the discrete event-based simulation paradigm [18]. The discrete-event simulator has to provide the following structures or properties:

1. A simulator object, which may manage the execution of events, by accessing an event queue where events are stored.

2. A scheduler, which inserts and removes events from that event queue.

3. A particular way of representing simulation time.

4. The events, which has to be simulated.

Identifying an appropriate simulation platform, comprises of consideration of the acceptability, of that platform in the academia, the flexibility and extensibility of that platform for new module, performance in terms of memory and processing speed of that platform etc. Furthermore, the simulation platform is preferred to be open source.

Available simulation platforms are enlisted in Table 2.1, from which it is easily deducible that OPNET, QualNet and NetSim are not appropriate for developing the simulator, as those are commercial software and closed source. From rest of the open source platforms, Network Simulator 2 (NS2) is obsolete, and J-Sim is not updated since 2005. In addition, the OMNeT++ is publicly distributed platform since 1997 [16]. OMNet++ follows the component based architecture [19]. The

Table 2.1: Available simulation platforms

Simulator name	Interface		Emulation	Open Source	Commercial	Language	Platform (OS)	Version
	GUI	CLI						
ns–3	N	Y	Y	Y	N	C++, Python	Windows, Linux, Mac, Free BSD	NS 3.26 (Oct 2016)
OMNET++	Y	N	Y	Y	N	C++	Windows, Linux, Mac OsX	V 5.1 (Dec 2016)
OPNET	Y	N	Y	N	Y	C, C++	Windows	V 17.5
J-Sim	Y	Y	Partial	N	Y	Java, TCL	Windows, Linux	V1.3 (Jan 2005)
QualNet	Y	Y	Y	Y	N	C++	Windows, Linux	–
NS2	N	Y	Y	Y	Y	C++, OTCL	Windows, Linux, Mac, Free BSD	NS2.35 (Obsolete)
NetSim	Y	N	Y	Y	Y	C, C++, Java	Windows	V9.1 (Sept, 2016)

components are developed using C++, then the developed components are assembled into larger components, to provide domain-specific functionality. Those domains are sensor networks, wireless ad–hoc networks, Internet protocols, photonic networks etc. As mentioned, in Section 1.3, OMNET++ is used to develop Open Source Satellite Simulator (OS3). But the performance of OMNET++, in terms of memory usage and scalability is lagging behind ns–3 [20].

On the other hand, Network Simulator 3 (ns–3) is highly modular, widespread in academia and integrates the architectural concepts from the Georgia Tech Network Simulator (GTNetS) [21], which is known for its good scalability characteristics. ns–3 is also developed using component based architecture [19], and ns–3 could be integrated with MatLab/Simulink program [22]. Additionally, ns–3 is developed on non-commercial *General Public License version* 2 *(GPLv2)*. Therefore, it is accessible at no cost. ns–3 also has Direct Code Execution (DCE) framework, which helps to execute existing implementations of user-space and kernel-space network protocols within ns–3. That means, ns–3 simulation program can use the real ping application from the operating system. It can also facilitate users, to use Linux networking stack in their simulations. The software core of ns–3 is built in C++ and Python [23]. Several design patterns are implemented in ns–3, such as object factory design pattern, template design pattern, component based object model, functor design pattern

and pointer to implementation (PIMPL) design pattern etc. [23]. ns–3 also supports object aggregation, which enables user to extend packet structure and design their own model [23] with less effort. The Internet nodes/computers are made more close to the real world implementation in ns–3, therefore nodes are capable of supporting the key interfaces, such as sockets, network devices, multiple interfaces and the use of IP addresses etc. [23].

In addition to those features, ns–3 can also support time driven simulation. Therefore, after finishing the rigorous analysis on the available simulation platforms, Network Simulator 3 (ns–3) has chosen for developing the satellite network simulator, as ns–3 is scalable, robust and flexible for adding modules. Moreover, more than 1000 peer reviewed publications have been published using ns–3.

2.2 Selection of the Transport Protocol

In network communication, the widely accepted reliable transport protocol is the Transport Control Protocol (TCP). Reliable data transfer is also required for haptic teleoperation. Thus, a comprehensive survey has done on the existing transport protocols, and their applicability in satellite communication.

2.2.1 Limitations of TCP in Space Communication

Early TCP implementations were designed for the point to point wired connection and used a go-back-n model, cumulative positive acknowledgments, and a retransmission timer expiration for re-sending lost data [24]. The main focus of these early stage TCP variants were not the minimizing network congestion, as the traffic congestion was not a serious problem at that time. But with the increase use of Internet various congestion avoidance algorithms have been developed. However, there are various

reasons, that impede reliable data transfer in satellite communication. Some of those reasons are:

High Transmission Error

In satellite communication, transmission error is much higher than the terrestrial networks. Thus, packet loss is common in satellite communication, which triggers the congestion control algorithm of TCP. But packet can be lost due to:

- Network congestion
- Corruption
- Link outage

So triggering the congestion control algorithm, in the event of packet loss, reduce the throughput.

Channel Asymmetry

The channel between the space station and the ground station is not same in terms of transmission power and bandwidth. That highly asymmetric channel is not available in the terrestrial network, therefore this asymmetry limits the TCP throughput, as low bandwidth return link carries the acknowledgements. Moreover, new transmission rate, in TCP, is dependant on the received acknowledgement.

Large Bandwidth Delay Product

Bandwidth delay product (BDP) is the maximum amount of data, that is present in the network at any given time, that means maximum amount of unacknowledged

data in the system. BDP calculated as follows,

$$BDP = Bandwidth(bits/second) \times RTT(second). \tag{2.1}$$

Due to large BDP in satellite communication, throughput becomes low.

Intermittent Connectivity and Variable Round Trip Time (RTT)

Intermittent connectivity and variable RTT leads to unstable flow and invokes congestion control and Slow Start algorithm, every now and then, which reduce the throughput unnecessarily.

2.2.2 Transport Protocol for Satellite Communication

In satellite communication, there exist at least three sources of loss: congestion, corruption and link outage. Traditional TCP behaves same to those losses by invoking the congestion control scheme, which limits the throughput in space communication. The Space Communication Protocol Standards (SCPS) originated in 1992, and proposed a transport protocol for the stressed environment like space. They proposed, Space Communication Protocol Standards-Transport Protocol (SCPS-TP) to identify the reason behind packet loss and respond accordingly [25]. SCPS-TP respond to congestion, corruption and link outage differently. SCPS-TP is different from traditional TCP, as it does not assume packet loss is caused by congestion, and uses parameters to figure out source of error per route basis.

Congestion-Induced Loss

In SCPS-TP, TCP Vegas is used as default congestion control mechanism. As TCP Vegas does not use receiver's receive window, as the upper bound of its congestion window and limits it self from overflowing the network by sending too much data. A modified Slow Start algorithm is also introduced in TCP Vegas, which is suitable for the stressed situation like satellite communication.

Corruption-Induced Loss

Corruption could occur during transmission, if that happens, SCPS-TP uses an open-loop, token bucket rate control mechanism [26]. Instead of invoking congestion control in response to packet loss due to corruption, SCPS-TP invoke rate control mechanism and stop overflowing the link capacity. In this token bucket rate control mechanism, the transmission is done at a specified rate. That specified rate is managed in a globally accessible routing structure at each end point. Therefore, the available capacity for a particular link is known by all SCPS-TP connection.

Link Outage Loss

In satellite communication, if a SCPS-TP host receives a link outage signal then it enter in the persist mode, and sends periodic probe packets to the SCPS-TP host, who sent the outage signal. In persist mode, it does not repeatedly time-out, retransmit, or back-off the retransmission timer.

Coping with Asymmetric Channels

To reduce the impact of asymmetric channel, SCPS-TP has to reduce the uses of the return link or the acknowledging link. Therefore, SCPS-TP uses a different header compression technique from traditional TCP, to reduce the overhead significantly on the acknowledgment channel, and achieve higher acknowledgment rates. SCPS-TP also delayed the acknowledgement sending time, by following an algorithm.

Relieving Bandwidth Constraints

In bandwidth-constrained environment, SCPS-TP uses Header Compression and Selective Negative Acknowledgment (SNACK) to improve performance.

SCPS-TP Header Compression

At the transport layer, SCPS-TP uses a end-to-end loss-tolerant TCP header compression scheme, which can tolerate connectivity changing and losses. As the compression scheme is working end-to-end, it can easily tolerate the changing connectivity since transmitter and receiver remains same so the position of compressor and decompressor remains same. This technique does not add any additional benefit if the satellite link is single hop.

SCPS-TP SNACK

SCPS-TP SNACK option uses the concept of TCP Selective Acknowledgement (SACK) option and TCP Negative Acknowledgment (NACK) option [27]. SNACK is selective negative acknowledgment, but it could specify multiple gaps in the received sequence in a bit efficient manner. Various TCP extensions for space, including

SNACK was proposed for increasing the throughput [28].

Implementing SCPS-TP in ns–3, is not the primary focus of this work. Thus, in the preliminary stage of simulator development, SCPS-TP is not incorporated. Moreover, ns–3 already incorporated TCP Vegas in its latest release. Therefore, instead of incorporating SCPS-TP in ns–3, TCP Vegas is used to test the developed satellite module, as TCP Vegas is used in SCPS-TP for congestion control.

2.3 Explore Satellite Communication Strategy of ISS

The International Space Station (ISS) consists of different modules from different country. Thus, ISS does not have a single communication strategy, different countries use their own communication system to send and receive data from Earth. The Russian modules communicate with the ground in two different ways. It communicates with the ground directly, with the help of the *Lira* antenna mounted to *Zvezda* (Star) module. *Zvezda* (Star) is a Russian module, docked with ISS on July 26, 2000 [29]. The *Lira* antenna also relays data to the ground using *Luch* data relay satellite system [29]. *Luch* data relay satellite system was temporarily unavailable [29, 30]; but, Russian Space Agency launched two new *Luch* satellites *Luch-5A* and *Luch-5B* in 2011 and 2012 respectively, to revive the system. The US Orbital Segment (USOS) communicate with Earth through the United States Tracking and Data Relay Satellite System (TDRSS), in geostationary orbit. ISS uses two different radio bands with the help of antennas mounted in the *Z1* truss structure. Those two radio bands are, S band $(2 - 4)$ GHz and Ku band $(12 - 18)$ GHz, shown in Fig. 2.2. By using TDRSS, ISS has almost continuous real-time uninterrupted communications with NASA's Mission Control Center (MCC-H) in Houston [29, 31].

Ground terminals in the MCC-H, relay signals to and from Tracking and Data

Table 2.2: List of TDRS

Name	Launched	Status
First Generation		
TDRS-A (TDRS-1)	April 04, 1983	Retired Fall 2009
TDRS-B (TDRS-2)	–	Destroyed in 1986
TDRS-C (TDRS-3)	September 29, 1988	–
TDRS-D (TDRS-4)	March 13, 1989	Retired 2011
TDRS-E (TDRS-5)	August 02, 1991	–
TDRS-F (TDRS-6)	January 13, 1993	–
TDRS-G (TDRS-7)	July 13, 1995	–
Second Generation		
TDRS-H (TDRS-8)	June 30, 2000	–
TDRS-I (TDRS-9)	March 8, 2002	–
TDRS-J (TDRS-10)	December 4, 2002	–
Third Generation		
TDRS-K (TDRS-11)	January 30, 2013	–
TDRS-L (TDRS-12)	January 23, 2014	–
TDRS-M (TDRS-13)	will launch 2017	–

Relay Satellites (TDRS). For download and upload voice, video, command, systems and research cargo data between ISS and MCC-H each TDRS has a Space-to-Ground Link (SGL) antenna. The MCC-H uses two functionally-identical off-site ground terminals at the White Sands Complex (WSC), located in Las Cruces, New Mexico for sending data. These two terminals are known as the White Sands Ground Terminal (WSGT) and the Second TDRSS Ground Terminal (STGT).

Tracking and data relay satellite system consists of some geostationary satellites listed in Table 2.2. An S-band antenna attached to the ISS always looking for exchanging data with the SGL antenna and a closest TDRS. For transmitting the data, S-band antenna has to be pointed in the right direction of the nearest TDRS. For that reason, the S-band antennas in the ISS, are attached to motorized gimbals. Therefore, the antennas can move and point into the correct position, to transmit the data. The other nations modules, such as European Columbus laboratory and Japanese

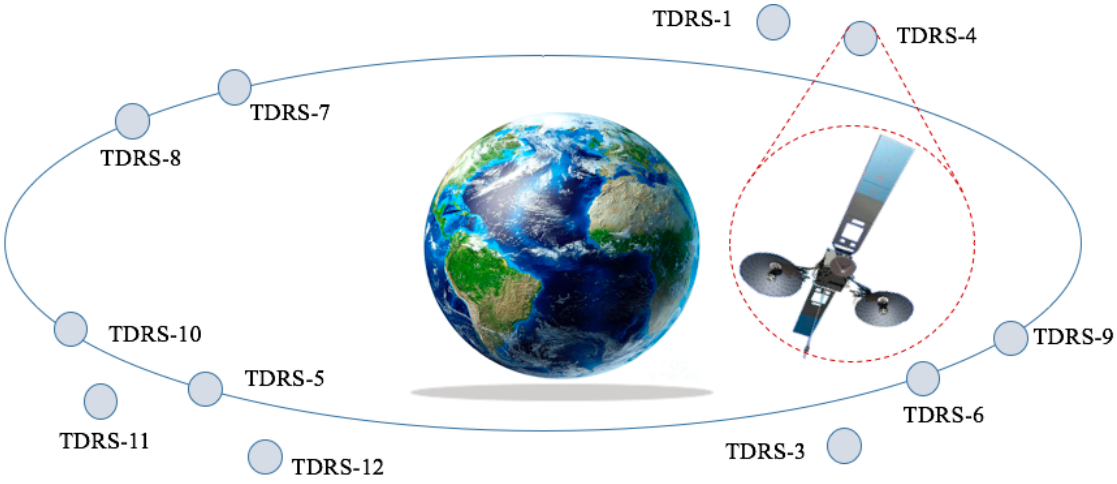


Figure 2.1: Each circle represents a Tracking and Data Relay Satellite (TDRS). Satellites placed outside the orbit are either non-functional or in testing phase. The inset shows a typical TDRS.

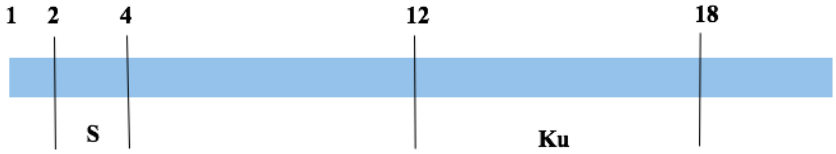


Figure 2.2: Frequency bands used in International Space Station for communication with the Earth. S-band is used for transmitting command and other data and Ku-band is used for transmitting high payload data like video.

Kib modules use the S and Ku band for tranmission. The European Data Relay System and Japanese system, uses similar technology like TDRSS [31]. In this work, the communication strategy used by NASA is adopted.

The investigation on ISS communication strategy showed that ISS is communicating with the MCC-H via the United States Tracking and Data Relay Satellite System (TDRSS) and White Sand Ground Terminal (WSGT). The working strategy of TDRS system is classified. Till date, NASA launched twelve satellites in the space as part of the TDRS system. As enlisted in Table 2.2, first generation satellites are about to expire and the third generation satellites are not operational yet. Thus,

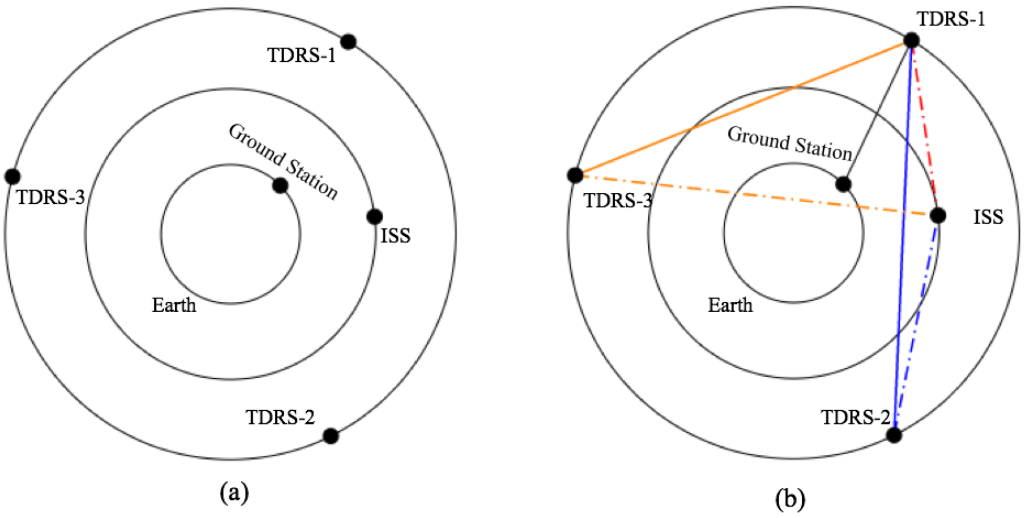


Figure 2.3: Two dimensional system model, where ground station is placed with a distance of Earth's radius and Tracking and Data Relay Satellites (TDRS) are placed 120° apart from each other and 35, 787 km away from ground station.

in the initial phase of the simulator development a two dimensional (2D) simulation model is developed. The 2D model, shown in Fig. 2.3, consists of five nodes, where three geosynchronous satellites (TDRS) are positioned 120° apart from each other, ground station placed with the radius of the Earth (R_E) and ISS rotating in a circular path. In 2D model, all the nodes are placed with the actual distance from the center of the coordinate system. Moreover, in 2D model three TDRSs are used, as theoretically three TDRSs could cover 95% of the Earth [32]. Furthermore, the MCC-H is considered as the ground station, though MCC-H is the control center, situated in Houston, Texas and data is sent to ISS through WSC in White Sands, New Mexico. But MCC-H and WSC is connected via NASA's Integrated Services Network (NISN), which is classified. Therefore, the assumption is made that, data from MCC-H to WSC reaches instantly. Thus, the communication with ISS, from ground station is done in three different ways, based on the ISS position in the space. Ground station always send data to TDRS-1. Then TDRS-1 either send data to ISS

directly or relay data to the TDRS positioned in close proximity of ISS. Therefore, three scenarios are as follows,

1. ISS is close to TDRS-1, therefore TDRS-1 sends data to ISS directly.
2. ISS in close proximity with TDRS-2, therefore TDRS-1 sends data to TDRS-2 afterwards TDRS-2 sends data to ISS.
3. ISS is close to TDRS-3 thus, TDRS-1 relay data to TDRS-3, then TDRS-3 deliver data to ISS.

As ground station always send data to TDRS-1, and TDRS-1 relay data based on the position of the ISS, therefore all the connections between nodes are point-to-point. In this work, other satellites, that orbiting in the space are not considered, therefore calculating the interference caused by other satellites is not required.

2.4 Determination of the Satellite Link Budget

The satellite link is similar to the terrestrial microwave radio relay link, but it has an advantage of not requiring as many re-transmitters, as required in the terrestrial link. The signals are transmitted over a satellite communication link, requires Line-of-Sight (LoS) communication. The transmitted power attenuates while propagates through the space and some noises added with the signal. Furthermore, transmitted signal and received signal amplified with the antenna gains. The received energy-per-bit to noise-density $\left(\frac{E_b}{N_0}\right)$ is defined as follows [33]:

$$\frac{E_b}{N_0} = \frac{P_t L_l G_r G_t L_p L_a}{k T_s R}, \tag{2.2}$$

where $\frac{E_b}{N_0}$ represents the ratio of received energy-per-bit to noise-density, P_t denotes the transmitted power, L_l depicts the transmitter-to-antenna line loss, G_t and G_r represents the antenna gain of transmitter and receiver respectively, L_p depicts the path loss, L_a is the atmospheric loss, k represents the Boltzmann constant and T_s denotes the system noise temperature (K) and R represents the data rate of the channel in bps.

The link equation can be represented in terms of decibels (dB).

$$\frac{E_b}{N_0} = P_{t\,dB} + L_{l\,dB} + G_{r\,dB} + G_{t\,dB} + L_{p\,dB} + L_{a\,dB} - 10log(k) - 10log(T_s) = 10log(R). \tag{2.3}$$

The parameters used in Eq. (2.3) is calculated or collected before the simulation starts except the free space path loss value as it depends on the distance between source and destination.

2.4.1 Free Space Path Loss

The free space path loss is calculated as follows [33]:

$$L_p = \left(\frac{\lambda}{4\pi D}\right)^2, \tag{2.4}$$

where λ is the wavelength of the signal and D is the distance between sender and receiver. The relationship between frequency and wavelength is characterized as follows,

$$\lambda = \frac{c}{f}, \tag{2.5}$$

where c represents the speed of light and f is the frequency in Hz. Therefore, from

Eq. (2.4) and Eq. (2.5) the following equation is obtained,

$$L_p = \left(\frac{c}{4\pi D f}\right)^2, \tag{2.6}$$

Converting Eq. (2.6) to dB gives:

$$\begin{aligned} L_{p\,dB} &= 20log(3 \times 10^8) - 20log(4\pi) - 20log(D) - 20log(f) \\ &= 147.5523 - 20log(D) - 20log(f). \end{aligned} \tag{2.7}$$

2.4.2 Atmospheric Attenuation

As the wave propagates through the atmosphere, it attenuates for gases, vapor, rain, cloud and fog *etc.* The attenuation can vary based on the rain drop sizes [34]. The attenuation can also be varied based on the humidity. [35]. Moreover, the atmospheric attenuation is low on the frequencies lower than 10 GHz [35, 36]. The atmospheric attenuation is a vast research topic, and it needs complex calculation and real time weather information, temperature, wind velocity, humidity *etc.* As the goal of this work is to determine the feasibility of long–distance space telemetry in ISS. Thus, atmospheric attenuation due to gases, rain, vapor and fog is considered as a constant loss irrespective to the frequency, and assumed as $L_{a\,dB} = -10$ dB.

2.4.3 Noise Temperature

Calculating the noise temperature, is challenging for a dynamic system, where ISS is moving with $8kms^{-2}$ speed. Noise temperature depends on the altitude and elevation from the ground, as well as the position of the sender and receiver. Calculating all the components, which contributes to the noise temperature, is a complex task and not aligned with the primary focus of this work. Therefore, some simplified assump-

tions have been made. For the simplification, three different noise temperatures are considered. First noise temperature is the noise from the sky (T_{sky}), depending on the elevation, T_{sky} varies from 2 K to 90 K [36]. As mentioned earlier, to cancel losses from weather and atmosphere the simulator used the $2-4$ GHz frequency band, thus, T_{sky} is assumed to be 40 K [37,38]. The other two noise sources are the galactic noise (T_G) and the noise from the Earth (T_E. The galactic noise T_G is the background cosmic noise and is assumed to be 2.73 K for all links [37,39,40]. The noise generated by Earth (T_E) is assumed as 60 K, as its value is typically between 10 K and 100 K [39]. The antenna temperature (T_A) is calculated as follows,

$$T_A = T_{sky} + T_G + T_E. \tag{2.8}$$

An amplifier has a noise figure (NF), which is typically varies between 2 dB to 3 dB [40]. The noise figure (NF) is a ratio between output system temperature and input system temperature or room temperature (T_0). In the link budget calculation, 3 dB is chosen for noise figure (NF). The noise figure is then converted to a noise temperature, of an amplifier as follows,

$$T_{Amp} = T_0(NF - 1). \tag{2.9}$$

The receiver temperature (T_{Rx}) is assumed as 293.15 K or (20°) C, and the loss (L_{Rx}) between the receiving antenna and the receiver input is 0.5 dB. The system noise temperature (T_S) is computed as,

$$T_S = \frac{T_A}{L_{Rx}} + T_{Rx}\Big(1 - \frac{1}{L_{Rx}}\Big) + T_{Amp}. \tag{2.10}$$

2.4.4 Antenna Gain

The antenna gain depends on the diameter d of the antenna and the wave length (λ). Antenna gain is calculated as [32],

$$G = \eta\Big(\frac{\pi d}{\lambda}\Big)^2, \tag{2.11}$$

where η denoted the antenna efficiency which considered as 60% or 0.6 and λ is calculated from Eq. (2.5) as frequency of the signal is 2.3 GHz. Diameter of the antenna in the ground station varied from 5 m $\sim$ 20 m and in the satellite varied from 1 m$\sim$ 2 m.

2.4.5 Total Link Equation

The line loss ($L_{l\,dB}$) in transmitter is -0.5 dB for all frequency ranges [33]. Some additional losses need to be added in the calculation. Those are implementation loss of -2 dB and polarisation loss of -3 dB [33, 39]. This two losses are specified as additional loss $L_{add} = -5$ dB. The total link equation for the simulation is calculated from Eq. (2.3),

$$\begin{aligned}\frac{E_b}{N_0} &= P_{t\,dB} + L_{l\,dB} + G_{r\,dB} + G_{t\,dB} + L_{s\,dB} \\ &\quad + L_{a\,dB} - 10log(k) - 10log(T_s) + L_{add} - 10log(R).\end{aligned} \tag{2.12}$$

The distance between various nodes D will be calculated during the simulation.

2.4.6 Minimum Detectable Signal

The minimum detectable signal or discernible signal (MDS) is defined as the smallest RF input signal, which could be observed in presence of background noise [38].

Therefore, to distinguish RF from the noise, it has to be at least ten times stronger than the noise floor. Noise floor (N_F) is reckoned as,

$$N_F = 10\log(kT_s1000) + NF + 10\log B, \tag{2.13}$$

where NF denotes the Noise Figure used in Eq. 2.9 as 3 dB, T_s is the system temperature calculated in Eq. (2.10) and B represents the bandwidth of the signal. The noise floor is calculated before the simulation start, and it is -73.431 dBm. Therefore, the received power to noise ratio has to be ten times greater than -73.431 dBm. Thus, computed MDS is -63.431 dBm. If the received SNR falls below this MDS value, then the packet would be dropped.

2.4.7 Bandwidth and Packet Error Rate

The frequency band used in practice to communicate with ISS is S-band ($2 \sim 4$ GHz) as mentioned in section 2.3. From the Eq. 2.6, it is deducible that with the increment of the frequency, path loss increases. But with the higher frequency more bandwidth is achieved. From the Shanon-Hartley theorem, channel capacity is calculated [41]:

$$C = B\log_2(1 + SNR), \tag{2.14}$$

where C represents the channel capacity in bits per seconds (bps), B denotes the channel bandwidth in Hertz (Hz) and SNR depicts the Signal-to-Noise ratio. From Eq.2.14, it is obvious that, doubling the signal bandwidth, doubles the channel capacity, if SNR remains same. But with the increment of the frequency noise increases, therefore SNR decreases.

In wireless communication, signal modulation is used for encoding digital (bi-

nary) data onto analogue signals. Data corruption is related with the modulation techniques. There are different types of modulation techniques are used nowadays and each of them has advantages and disadvantages. For encoding digital data all the modulation techniques modify the amplitude, frequency or phase of an analogue carrier signal. Based on the modulation scheme, Bit Error Rate or BER of a signal changes. In this work, Binary Phase Shift Keying (BPSK) is chosen as the modulation scheme. In BPSK, it uses two phases of the carrier signal which are separated by 180°. The BER for BPSK is as follows,

$$BER = \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right). \tag{2.15}$$

where erfc() is the complementary error function and $\frac{E_b}{N_0}$ is the energy-per-bit to noise-power-density-ratio.

In this study, the ratio of $\frac{E_b}{N_0}$ is computed whenever a receiver receives a packet in the simulation time, and from the value of $\frac{E_b}{N_0}$, the packet error rate is calculated using the following equation,

$$P_e = 1 - (1 - BER)^L, \tag{2.16}$$

where P_e denotes the packet error rate (PER). PER is the number of incorrectly received bits divided by the total number of received bits in a packet and L represents the number of bits in a packet. Eq. 2.16 is approximated for small bit error probabilities by

$$P_e = BER \times L. \tag{2.17}$$

The P_e is computed for each packet, afterwords a uniform random number is gen-

erated, if the generated random number turned out less than P_e then the packet is dropped in the receiver side. This process is known as a Bernoulli trial (or binomial trial), in the theory of probability and statistics.

2.5 Implementation in ns–3

Before discussing the developed satellite network device and channel, some core concepts of ns–3 should be described first. ns–3 is a collection of software libraries that work together. Program can be written in C++ or Python that links with these libraries. Developing any module in ns–3 requires a clear understanding of these following concepts.

2.5.1 Key Abstractions

Some key abstraction has to be discuss which are commonly used in networking, but have a specific meaning in ns–3.

Node

In ns–3, a basic computing device is denoted as node which is represented in C++ by the class `Node`. `Node` is basically a class providing various functionality like adding network device, install application, protocol stacks etc. `Node` is the simulated version of a bare computer without having network devices and Internet protocols etc.

Application

The `Application` class in ns–3 mimics the real world computer software, that can be installed in ns–3 `Node` class, and do user defined tasks in the simulated world of ns–3.

Channel

The channel is the medium of sending data from one computer to another computer in the real world. It could be wireless or wired connection. In ns–3, `Channel` class perform the same task of transmitting data between nodes. The `Channel` class provides some basic characteristics which could be extended as per specialized class.

Net Device

Net device is the short form of network device. In ns–3, `NetDevice` is a class which represents the network device installed in the node/computer. `Node` class then send or receive data in the simulation from other nodes via the `Channel` class. Multiple network devices can be installed in the node like real world.

Topology Helpers

Topology helpers are classes, which provides functions to create specific simulation scenarios, without writing detailed instructions to reduce coding complexity. With the help of those classes, programmer could create complex network topologies with minimal coding. For example, programmer could create 100 nodes, install particular `NetDevice` and `Channel` then set their values collectively, with the help of topology helper class. Without helper class, programmer has to tweak each and every nodes manually. In **Appendix A**, a code snippet from the developed topology helper is provided.

Events and Simulator

The properties of a discrete event simulator is described in subsection 2.1 in page 10. To achieve those properties, a `Simulator` class is used in ns–3 to schedule events. Where `Event` is another class. After scheduling events in the event queue, program has to execute them by entering the simulator main loop by calling `Simulator::Run`. When the main loop starts running, it sequentially executes all scheduled events in the event queue from oldest to most recent until there are either no more events left or `Simulator::Stop` has been called.

2.5.2 Object Model

As mentioned earlier, ns–3 is a C++ library which consists of traditional C++ objects, with some extra functionality and features.To provide those extra features, ns–3 used several design patterns, such as classic object-oriented design (polymorphic interfaces and implementations), separation of interface and implementation, the non-virtual public interface design pattern, an object aggregation facility, and reference counting for memory management. ns–3 created three special base classes which can be extended to obtain special features by user program. Those base classes are:

- Class `Object`
- Class `ObjectBase`
- Class `SimpleRefCount`

By extending these base classes, a user class attains the following properties:

- ns–3 type and attribute system
- an object aggregation system

- a smart-pointer reference counting system (class `Ptr`).

Classes that extend class `ObjectBase` inherit first two properties, but not the third one. Again, classes which derived from class `SimpleRefCount`, get only the smart-pointer based reference counting system. Only `Object` class provides all three properties to the child classes.

2.5.3 Type and Attribute System

`Object` class in ns–3 provides, some features for organizing the system and set or retrieve values are stored in the objects.

- It provides the "Metadata" system which links the class name with a lot of meta-information about that object, including:
  - Information about the base class of the subclass,
  - Accessible constructors of that subclass,
  - The set of values of the subclass which is known as attributes in ns–3,
  - Information about those attributes like whether each attribute can be set, or is read-only,
  - Limit the values for each attribute.

`TypeId` Class

The `TypeId` class is an extended form of run time type information (RTTI) for ns–3 `Object` class. Normally C++ language provides a simple kind of RTTI in order to support `dynamic_cast` and `typeid` operators. But in ns–3 classes, that derived from class `Object`, inherits a metadata class named `TypeId`, similar to C++ `typeid`

operator. In ns–3, `TypeId` class records meta-information about the class which is extended from `Object` class. Therefore, newly created class can be used in the object aggregation and with component manager system.

Attribute System

The main purpose of the attribute system is to provide a organized way during a simulation to access internal members of an object. Therefore, during a simulation program, programmer can access any particular attribute or internal variable conveniently. Thus, programmer can tweak any parameter, and observe the simulation behavior. Attribute system in ns–3, manipulated with the help of `TypeId` class and `Config` class. Using the `TypeId` class within the `Object` class enables programmer to set, change, control and retrieve a value of an attribute. With the help of the `Config` class a particular attribute can be set from anywhere in the code.

2.5.4 Object Aggregation

Object aggregation is an well established design pattern, which is incorporated in ns–3 as object aggregation system for the use of inheritance and polymorphism to extend protocol models. For example, specialized versions of TCP, such as `RenoTcpAgent` derive from class `TcpAgent`. In ns–3, `Node` class do not posses any Internet stack; therefore, an Internet stack has to be incorporated with the `Node` class by aggregating `Ipv4L3Protocol` object. Following code snippet taken from ns-3 code base, shows the procedure,

```
Ptr<Node> node = CreateObject<Node>();
Ptr<Ipv4L3Protocol> ipv4 = CreateObject<Ipv4L3Protocol> ();
ipv4->SetNode (node);
node->AggregateObject (ipv4);
```

2.5.5 Smart Pointers

Memory management in a C++ program is very crucial and complex process. It has to be done correctly and in a proper way, otherwise program may crash. In C++, if a object is created using `new` command, then that object has to be deleted by explicitly calling `delete` command. Most often, programmers forgot to do that, which leads to memory fragmentation in heap. In a complex system, like ns–3, memory management is important, therefore a reference counting design is implemented. Where `Ptr` class provides a mechanism of reference counting, which provides automatic object deletion. This mechanism, deletes object when it goes out of the scope or no longer needed. For that reason, it is known as Smart Pointers. Reference counted objects are allocated using `Create` or `CreateObject` method. Following code snippet shows `CreateObject` method's use,

```
Ptr<WifiNetDevice> device = CreateObject<WifiNetDevice> ();
```

2.5.6 Tracing System

In ns–3, tracing subsystem is one of the core subsystem to understand. As the main focus of doing simulation is to generate data and analyze those data later on. Tracing subsystem allows user to get data from the core classes of ns–3 without modifying the

core code. There are hundreds of classes in ns–3 and those classes contain thousands of variables. Therefore, printing particular attributes value, without modifying the core code, is tricky. That tricky part is done with the help of `Callback`, `Attribute` and `Config` subsystems.

2.5.7 Callback

`Callback` system is another core subsystem of ns–3. The main goal of the `Callback` system in ns–3 is to call a method from anywhere of the system without any specific inter-module dependency. That means, if a piece of code wants to call a method, it needs to understand that method's signature, or at least, where that method definition is located. For that reason, in programming language, a common paradigm is to call a method. Therefore, when one method wants to be called by a piece of code, that method uses callback system. The concept of callback is not straight forward, as one method wants to be called by a code. Callback is heavily used by Tracing subsystem as one method wants to be called, when the value of an attribute changes. Without `Callback`, that task can not be done.

2.5.8 Satellite Network Device

Before implementing the simulator, an extensive study has been done on the existing module in ns–3. As ns–3 does not have any satellite module, which can serve the satellite communication, a new module has to be implemented and incorporated with the existing modules in ns–3. Implementing the new module is done with maintaining all the standards imposed by ns–3, and providing proper tracing system, declare attributes properly thus callbacks methods can hook those as per requirements. The proposed module structure is showed in Fig. 3.1. The module is capable of aggregat-

ing `Mobility` class and `PropagationLoss` class with the `SatelliteChannel` class, therefore delay and path loss can be calculated. In the planning phase, the naming of channel and device was kept short which changed in the implementation phase. For example, in the Fig.3.1 satellite network device denoted as, `SatNetDevice` which renamed in the implementation phase as, `SatellitePointToPointNetworkDevice`.

2.5.9 Two Dimensional System Model

The developed `SatNetDevice` is installed in all the nodes in the simulation. Then those nodes are connected with each other using point-to-point `SatChannel`. Therefore, ground station is connected with TDRS-1 only. TDRS-1 is connected with TDRS-2, TDRS-3 and ISS using different `SatNetDevice` objects. TDRS-2 and TDRS-3 connected with ISS separately. After that, we created three different TCP socket in ISS for three different scenarios described in subsection 2.3 in page 18. In 2D model, ISS is moving following a circular path and send/receive data.

2.6 Implementation of Three Dimensional Orbital Propagation Model

After simulating the 2D model in ns–3, the orbital propagation model is incorporated in the simulator. Orbital propagation is the process of predicting a satellites position accurately in a particular time. To generate that information, it is important to know all the existing coordinate systems and transformation among them. Though the fundamental concepts about moving objects under gravitational attraction is known since the time of Sir Isaac Newton but the practical determination of an object's position in orbit is difficult.

2.6.1 Perturbing Forces and SGP Model

As per Newton's Law of Universal Gravitation, in case of two point masses, where one is heavier than other, then the smaller object is following an elliptical orbit around the heavy object. In the satellite orbit prediction, all the perturbing forces along with the primary force of gravitational attraction between Earth and satellite has to be calculated. These additional forces act on satellite and change its trajectory from ellipse. Perturbing forces are anything that bring aberration to the satellite orbit. For example, Earth geospherical structure, atmospheric drag, gravitational attraction from Sun, Moon, other planets and stars are accumulate, as perturbing forces or perturbation. Moreover, calculation starting point with respect to time, is also an important consideration in orbital propagation as the Earth is rotating around the Sun and Sun is also orbiting along with the solar system. Therefore, the North American Aerospace Defense Command (NORAD) in 1970 developed a model known as SGP (Simplified General Perturbation) for predicting any orbiting objects trajectory. The SGP model has improved and has several versions. In this work, SGP4 model is used to predict satellite trajectory.

2.6.2 TLE and State Vectors

Keeping data in a particular format for the orbiting objects, is known as orbital element sets. NORAD uses two forms of orbital element set. Those are state vectors and Keplerian orbital element set, which is also known as NORAD two-line element set (TLE). A state vector is a collection of states or values which is used with the state transformation rules to predict satellites position in past or future in terms of state vectors. Again, TLE is a data format stored in two lines, each line is consists of 80−columns of `ASCII` text, to store the data regarding orbital elements of an Earth-

orbiting object for a given point in time. The TLE data representation is used with the simplified perturbations models (SGP, SGP4 etc.) for generating state vectors. SGP4 model takes input in the form of TLE and generate state vectors.

2.6.3 Coordinate System

In absence of perturbing forces, two set of values would suffice for predicting satellites orbit: the position and the velocity. Without velocity satellite cannot rotate around the Earth. Position and velocity is closely coupled with time and the coordinate system. In SGP4 model, the time is represented in terms of Julian Day. Before describing the Julian day it is necessary to know about the solar day and the sidereal day.

Sidereal Day

Sidereal day is the time that the Earth takes to rotate 360° about its axis.

Solar Day

Solar day is noon to noon time difference from a particular position of the Earth. If the Earth was rotating in a fixed place with respect to sun then sidereal day and solar day would be the same thing. But Earth is rotating on its axis, as well as around the Sun in a elliptical path. Therefore, solar day and sidereal day is not same and sidereal day is 23 hour 56 minutes but solar day is 24 hours in length.

Julian Day

The Julian day is an integer assigned to a whole solar day starting at noon on January 1, 4713 BC, proleptic Julian calendar which is November 24, 4714 BC, in the proleptic

Gregorian calendar. That day is considered as the 0^{th} Julian day [42–44]. In that particular day, three multi-year cycles were started. Those cycles are *Indiction*, *Solar*, and *Lunar* cycles. With the help of Julian day, any day or time in the history of mankind can be represented. To illustrate that, the Julian day number for the day starting at $12:00$ Universal Time (UT) on January $1, 2000$, is $2,451,545$ [45].

ECI Coordinate System

For calculating state vectors, the Earth-Centered Inertial (ECI) coordinate system is used, which is a Cartesian coordinate system, centered at the center of the Earth and in this system, Earth is fixed relative to the stars. That means, Earth is not rotating or accelerating. The z axis is pointing North through the Earth's rotational axis, the x axis pointed towards the direction of the vernal equinox and the y axis completes the right-handed orthogonal system. Vernal equinox is imaginary connecting line of the center of the Earth to the center of the Sun at the beginning of the spring when the Sun crosses the Earth's equator moving North. The position of Earth has to be calculated from 0^{th} Julian day to predict the position of a satellite accurately. Calculating time from 0^{th} Julian day is hectic, therefore a time stamp is defined with the Earth's Mean Equator and Equinox position, at $12:00$ Terrestrial Time on 1^{st} January, 2000 in the ECI frame. It is also known as $J2000$ or $EME2000$. All the calculation in SGP4 is done with respect to $J2000$.

As position of an observer or ground station is represented in terms of latitude, longitude and altitude above the Earth's surface and satellite's position is typically represented using TLE. NORAD SGP4 model takes the standard TLE data set and time, then it produces the state vectors of the satellite in J2000 ECI coordinate system.

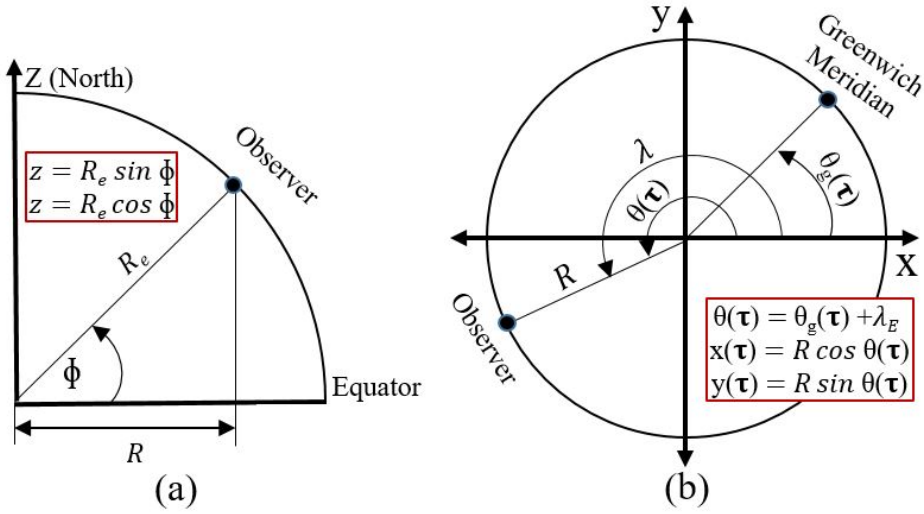


Figure 2.4: Longitude and latitude to Earth Centered Inertial (ECI) coordinate systems conversion.

The position of a observer on the Earth is dependant on the time as Earth is rotating. Calculating z axis is depicted in the Fig. 2.4, where the Earth is considered spherical for the ease of illustration. In Fig. 2.4, a side sliced view of the Earth with North up is shown. Where ϕ observers latitude, R_e is Earth's equatorial radius. Thus, the z coordinate of the observer, located on the mean sea level, is computed as follows,

$$z = R_e sin\phi. \tag{2.18}$$

If the observer is located above mean sea level by h, then the R_e would be replaced with $R_e + h$ in Eq. 2.18. Now for calculating x and y coordinate of the observer, the value of R is required from Fig. 2.4 which calculated as follows,

$$R = R_e cos\phi. \tag{2.19}$$

Computation of the x and y coordinates requires the consideration of time as Earth rotates in the $x - y$ plane (i.e., about the z axis). Therefore, unlike the z coordinate

Table 2.3: The nodes in ns–3 and their representation in 2D and 3D model. As TDRS-9 replaces TDRS-1 therefore TDRS-1 in 3D model will represent TDRS-9. This assumption is also applicable to other nodes

Representation in simulation	2D Model	3D Model
n0	Ground station	WSGT located at latitude 32.5007°N and longitude 106.6086°W
n1	TDRS-1	TDRS-9
n2	TDRS-2	TDRS-8
n3	TDRS-3	TDRS-10
n4	ISS	ISS

the x and y coordinates of a point on the Earth's surface will vary with time. As discussed earlier, local sidereal time is related with the rotation of the Earth directly but time is calculated in terms of Universal Coordinated Time (UTC), which is mean solar time. For calculating the local sidereal time with respect to UTC, observer's east longitude, λ_E is added to the Greenwich Sidereal Time (GST) also known Greenwich Mean Sidereal Time (GMST), $\theta_g(\tau)$. The GMST $\theta_g(0^h)$ at 0^h of any particular date is obtained from the US Naval Observatory's Astronomical Almanac [46]. After obtaining $\theta_g(0^h)$ of any particular day, the position of the observer at any particular time $\Delta\tau$ is calculated as follows,

$$\theta_g(\Delta\tau) = \theta_g(0^h) + \omega_e \Delta\tau, \tag{2.20}$$

where $\Delta\tau$ is the UTC time of interest and $\omega_e = 7.29211510 \times 10^{-5}$ radians/second is the Earth's rotation rate. The above calculation is done considering Earth is spherical, but in practical Earth is geospherical in shape. Therefore, some more calculations is done and incorporated in the SGP4 model library developed by Vallado [47]. In their `C++` library, they incorporated all the orbital propagation aspects to predict

exact satellite location. The library take TLE as input and generate state vectors as output. Thus, the library is used along with the publicly published TLE by NASA, to generate actual trajectory of ISS in the simulation. After generating the actual trajectory data of ISS, TDRS-8, TDRS-9, TDRS-10 using the TLE data from NASA, those data is incorporated in the 2D model and enhanced the model to 3D. The representation of nodes, in ns–3 of 2D and 3D model, is depicted in table 2.3. As the development evolved from 2D to 3D model, TDRS-9 in 3D is represented as TDRS-1.

Chapter 3

Implementation in ns–3

3.1 Satellite Point to Point Module

The ns–3 is developed using C++ programming language. There exist various modules in ns–3 to construct a simulation program as per user's requirement. But ns–3 does not have any satellite communication module to use. Therefore, in this work, a satellite communication module is developed to simulate the haptic communication scenario over the space. The development of a satellite module is inspired by the existing `point-to-point` module in ns–3 and the diagram provided in the paper [17] on Satellite Network Simulator–3, which is not public yet. The developed `satellite-point-to-point` module's architecture is depicted in Fig. 3.1.

The ns–2 did not have any standard structure to follow, therefore researchers develop their module in their own way in ns–2 and upload it in the ns–2 repository. Thus, ns–2 becomes unmanageable and ns–3 comes to the scene. The ns–3 follows a strict organization of the files in a module. The next section, describes the module creation in ns–3.

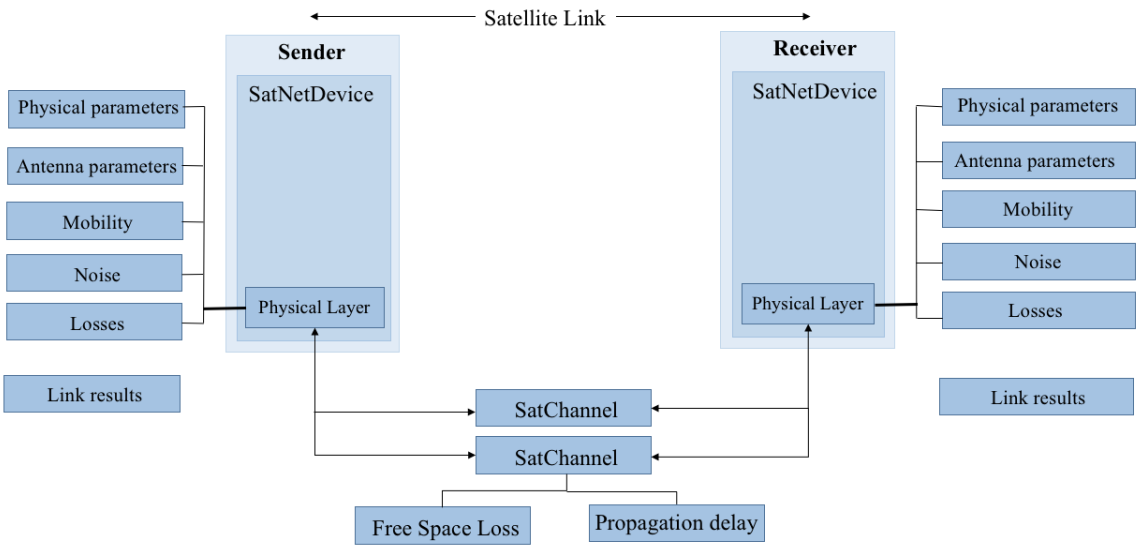


Figure 3.1: Satellite communication module consists of network device and channel.

3.2 Creating a New Module in ns–3

In ns–3, a group of related classes, examples, and tests are combined together to form a module. Those modules are reusable. Therefore, when a new researcher creates his own module that module can be used with the existing ns–3 modules. For this work, a module named `satellite-point-to-point`, is developed using the following steps:

3.2.1 Module Layout

In ns–3, all the modules are stored in the `src` directory. Different researchers develop their module in their own way in ns–2, but in ns–3 they have to follow a guideline for developing their module. If researcher did not follow the guidelines and did not organized his/her code according to guideline, his/her code would not be merged with the existing ns–3 repository. Therefore, the manageability of ns–3 is ensured. A typical ns–3 module consists of the following directories and files,

```
src/
 module-name/
   bindings/
   doc/
   examples/
     wscript
   helper/
   model/
   test/
     examples-to-run.py
   wscript
```

Note that, it is not necessary to have all the folders present in the exact manner in a particular module. But it is expected, that researcher would provide proper examples and test programs, before submitting his/her module to be accepted by the ns–3 community.

3.2.2 Create the Module Skeleton

In ns–3, there is a python script named `create-module.py`, which is provided in the `src` directory for creating the skeleton of a new module. The developed `satellite-point-to-point` module is created with the help of `create-module.py` script in the `src` directory. For creating the module, following command was used,

```
./create-module.py satellite-point-to-point
```

That command, creates `satellite-point-to-point` module in the `src` directory with the following structure,

```
src/
  satellite-point-point/
    doc/
      satellite-point-point.rst
    examples/
      satellite-point-point-example.cc
      wscript
    helper/
      satellite-point-point-helper.cc
      satellite-point-point-helper.h
    model/
      satellite-point-point.cc
      satellite-point-point.h
    test/
      satellite-point-point-test-suite.cc
    wscript
```

3.2.3 Merge with the Existing System

After creating the module, the module has to be linked with the existing system. As ns–3 uses `Waf` system to compile and run, therefore newly created module has to be linked with the existing `Waf` system. Thus, existing `Waf` system could compile and run newly created module. The linking is done, by editing the `wscript` file created in the `satellite-point-to-point` directory. In that `wscript` file, all the required module has to be defined, therefore `Waf` system could add the existing required modules

for newly created module. The developed module depends on the `network`, `mpi`, `mobility` and `propagation` modules. Therefore, those dependencies are added in the `wscript` file. Thus, after modifying the `wscript` file, it becomes:

```
def build(bld):
    module = bld.create_ns3_module('satellite-point-to-point', ['network',
        'mpi', 'mobility', 'propagation'])
    module.includes = '.'
    module.source = [
        'model/satellite-point-to-point-net-device.cc',
        'model/satellite-point-to-point-channel.cc',
        'model/satellite-point-to-point-remote-channel.cc',
        'model/sppp-header.cc',
        'model/satellite-physical.cc',
        'helper/satellite-point-to-point-helper.cc',
        ]

    module_test =
        bld.create_ns3_module_test_library('satellite-point-to-point')
    module_test.source = [
        'test/satellite-point-to-point-test.cc',
        ]

    headers = bld(features='ns3header')
    headers.module = 'satellite-point-to-point'
    headers.source = [
        'model/satellite-point-to-point-net-device.h',
        'model/satellite-point-to-point-channel.h',
```

```
        'model/satellite-point-to-point-remote-channel.h',
        'model/sppp-header.h',
        'model/satellite-physical.h',
        'helper/satellite-point-to-point-helper.h',
        ]

    if (bld.env['ENABLE_EXAMPLES']):
        bld.recurse('examples')

    bld.ns3_python_bindings()
```

This `wscript` file will be called by the `Waf` system during building ns–3, and the modules that used in the `satellite-point-point` module such as `network`, `mpi`, `mobility`, `propagation` etc. are also linked with the newly developed module. The newly developed module is usable from any application in ns–3 by incorporating the header file `satellite-point-to-point-module.h`.

3.3 Classes in the New Module

The newly developed `satellite-point-to-point` module is the preliminary model for satellite communication. This model mimics a very simple point to point wireless link `SatellitePointToPointChannel`, connecting exactly two `SatellietPointToPointNetDevice` devices, via `SatellitePhysical` layer class. In this module, data is encapsulated in the Point-to-Point Protocol (PPP RFC 1661), and assumed to be established and authenticated all the times. In the developed module, structure of `SPPP-Header` is analogous to `PPP-Header` class in `point-to-point` module.

3.3.1 SatellietPointToPointNetDevice Class

The `SatellitePointToPointNetDevice` provides following Attributes:

- ReceiverGain: The double value of the receiver antenna gain.
- TransmitterGain: The double value of the transmitter antenna gain.
- TransmitPowerDbm:The double value for the transmit power of the transmit antenna.
- Address: The ns3::Mac48Address of the device (if desired).
- DataRate: The data rate of the device (ns3::DataRate).
- TxQueue: The transmit queue used by the satellite network device (ns3::Queue).
- Rx: A trace source for received packets.
- Drop: A trace source to notify the dropped packets.

The `SatellitePointToPointNetDevice` mimics a real world transmitter section which puts bits in a wireless channel. The `DataRate` attribute specifies the data generation rate in terms of number of bits per second that the device will simulate sending over the satellite channel. Actually, no bits are sent, but an event is scheduled in later time, based on the number of bits in each packet and the specified `DataRate`. In the developed model, receiver can receive data in any data rate. Therefore, there is no receiver data rate attribute in present in the model. By setting the transmitter data rate, both devices connected via a given *SatellitePointToPointChannel* create a symmetric channel. Moreover, user can also create a asymmetric channel by setting

the `DataRate` in both network devices. In `SatellitePointToPointNetDevice`, there is a random variable instantiated from the `UniformRandomVariable` class.

3.3.2 SatellietPointToPointChannel Class

The satellite point to point net devices are connected via `SatellitePhysical` class and an `SatellitePointToPointChannel` as depicted in the Fig. 3.1. This satellite channel, models an wireless medium transmitting data bits at the data rate specified by the source network device. The `SatellitePointToPointChannel` calculates the delay between source and receiver by calculating their distances from their position in the Cartesian coordinate system which is obtained from their `Mobility` model. Then the delay is calculated by dividing the distance with the light speed. Therefore, `satellite-point-to-point` module is depended on `mobility` module. After adding the delay, `SatellitePointToPointChannel` deliver the data packet to the `SatellitePhysical` class of the receiver side with the received power calculated using the Eq. (2.12).

3.3.3 SatellitePhysicalClass

`SatellitePhysical` class, at the receiver side, deliver the packet with the received power. The `SatellitePointToPointNetDevice` then decide whether it will discard the packet or not, based on the packet error rate calculated by the Eq. (2.16) depicted in Chapter 2 and a uniform random variable generated by `UniformRandomVariable` class. If the generated number is higher than the packet error rate, then the packet is dropped.

3.3.4 Use of SatellitePointToPoint Module

There is a `SatellitePointToPointHelper` class, which is used to create and configure the satellite net devices and channel. In ns–3, all the helper class works in a same way. In this work, `SatellitePointToPointHelper` class takes either a `NodeContainer` object containing two `Node` objects or two `Node` objrcts as parameter to install satellite communication module between those objects. Those node objects, actually represents two bare computer without having any communication module installed. Then helper class, creates two network devices in those two nodes and then connect them using satellite point to point channel. The `Install()` method of helper class is given in **Appendix A** to show how installing network module in ns–3 is done. After completing the installation, the helper class returns a `NetDeviceContainer` object containing those newly created net devices along with a satellite channel connecting them. The following code snippet shows, how helper class simplify programmers life. The `Install()` method from helper class do all the work to connect two nodes. Using the helper class programmer can also set device attribute collectively. Data rate of the created device is initialized in the following code as well.

```
NodeContainer satNodes;
nodes.Create (2);
SatellitePointToPointHelper satHelper;
satHelper.SetDeviceAttribute ("DataRate", StringValue ("256Kbps"));
NetDeviceContainer devices;
devices = satHelper.Install (satNodes);
```

3.4 Incorporating Satellite Orbital Position

In ns–3, the mobility modules source code placed in *src/mobility* directory. A *MobilityModel* object track the position of a `Node` object with respect to Cartesian coordinate system. To be noted that, *MobilityModel* object has to be aggregated with the *Node* object and could be retrieved by `GetObject<MobilityModel> ()` method. in ns–3, different motion behavior is implemented, but those only use Cartesian coordinate systems, more specifically `MobilityModel` only support 3D vectors (x, y, z).

3.4.1 Implementing the 2D Model

In the preliminary stage of the development, the model described in Subsection 2.5.9 is implemented in the 2D simulated plane, where the value of z axis kept 0. The nodes are placed in 2D plane with the actual distance from each other. For example, Ground Station placed at a distance of Earth radius from the center of the coordinate system. All the TDRSs, placed 120° apart from each other and ISS changed its position per second basis. The 2D model was the part of the incremental development phase. The 2D simulation, validated that placing a TDRS at $35,786\ Km$ apart from Ground Station generates the real delay in the simulation.

3.4.2 Implementing the 3D Model

As mentioned in Subsection 2.6.2, satellite orbits are represented either in TLE or in state vectors. State vectors contains satellite's Cartesian coordinate and some other information. Implementing a new mobility module which will support the state vectors in ns–3 is not the primary focus of this work. Therefore, the Cartesian coordinate values from the state vectors are extracted and used to simulate the 3D orbital propagation. As mentioned in Section 2.6, the satellite orbital prediction is

done by the library, developed by Vallado [47]. The generation of the state vectors for TDRS-8, TDRS-9, TDRS-10 and ISS is done in one single program depicted in **Appendix B**. The position predicted continuously 92 minutes and stored in separate files.

Those files then used in the simulation to set `Node` objects mobility in Cartesian coordinate. Therefore, actual position of all the satellites are used in the developed simulator. The position changes of the satellites are scheduled in each second.

3.5 Transmitting Data Over the Satellite Channel

The data generated by a haptic application is stored in a comma separated file (csv). Therefore, the double data is stored in string format. Thus, while retrieving data from file using `std::string` returns each variables in different length. Moreover, sending data over the TCP socket has a problem, which is TCP treats data as a byte stream. It does not care about the data it self but only care about the byte sequence. Therefore, sending and retrieving data using TCP is a bit tricky for this work, as data has to be sent as per the sample size. Again the data should be retrieved from the receiving end in a proper order. Therefore, in this work, data is presented using the well-known C++ custom data type `union`. Using the `union`, data is stored in a `double` array and sent through the socket as a `byte` array, and in the receiving end, received as a `byte` stream but interpret as a double array. The following code snippet shows the `union` declaration in the developed simulator,

```
union{
   double d[120];
   uint8_t byteStream[sizeof(double)*120];
}data;
```

The conversion between `std::string` to a `std::vector<>` is depicted in **Appendix C**.

Chapter 4

Simulation Results

The application that generates the data used in haptic teleoperation is discussed in this chapter. The generated data is encapsulated in TCP packets and then sent over the `satellite-point-to-point` module while ISS is orbiting around the Earth. The three scenarios depicted in Section 2.3 is employed while sending data to ISS from Earth. Afterwards, the behavior of the congestion window, throughput and Round Trip Time (RTT) is analyzed.

4.1 Employing the Simulator Between Master-Slave Sites

The developed simulator was employed in a virtual platform in order to investigate how the platform operated when a series of packet data is transferred between Earth and the ISS. The platform was a *Haptic Tele-operated System (HTS)*. In this case study, the information of position/orientation and /or force at the master site on Earth was sent to the slave site on the ISS. The data generated in the HTS system was used in the ns-3 simulation and sent from ground station to the ISS encapsulated in TCP packets.

The HTS is a master–slave setup comprising a haptic hand–controller that is located at the master site on Earth and a robotic manipulator installed on the ISS. Accurate control of both master hand–controller and slave robot is essential, and hence the kinematic and dynamic modelling of the teleoperated system is very important. In this case study, the information of a 3 degree–of–freedom (DOF) haptic apparatus was sent to the slave site on the ISS. The position vector at the slave end-effector is shown by ($P_s = (x_s, y_s, z_s)$, and ($P_m = x_m, y_m, z_m$) represents the position of the master end–effector. P_s is the scaled version of the master haptic end–effector (P_m), and the scaling factor is determined based on different parameters such as the workspace required at the slave site and the accuracy needed to accomplish the given task. Therefore, the first requirement to have the robot running at the ISS, is to sent the packet data from Earth.

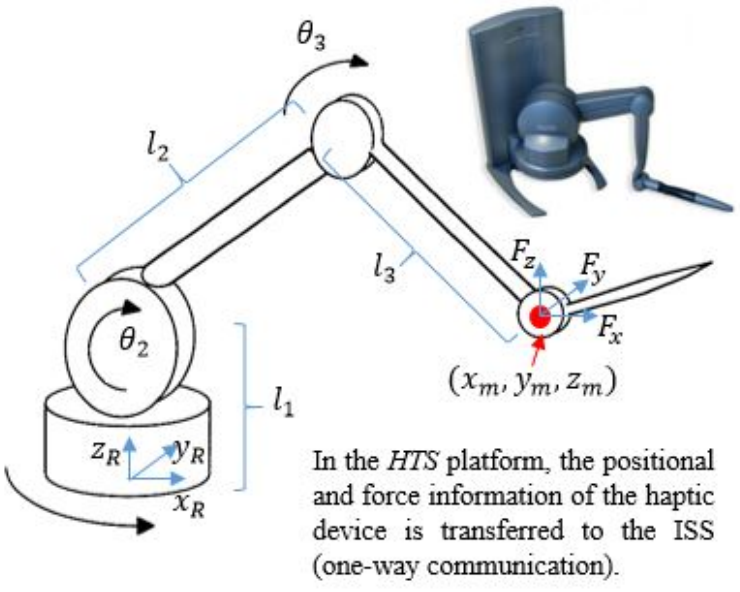


Figure 4.1: Positional and force components generated at the master site. The data packet contains 3 positional components ($x_m,\ y_m,\ z_m$) and 3 force components ($F_x,\ F_y,\ F_z$).

The position components of the haptic device, illustrated in Fig. 4.1, is calculated in real–time using,

$$\begin{cases} x_m = [l_2\cos(\theta_2) + l_3\cos(\theta_2+\theta_3)]\cos(\theta 1), \\ y_m = [l_2\cos(\theta_2) + l_3\cos(\theta_2+\theta_3)]\sin(\theta 1), \\ z_m = l_2\sin(\theta_2) + l_3\sin(\theta_2+\theta_3), \end{cases} \tag{4.1}$$

where $\theta_1, \theta_2, \theta_3$ represent the angular displacements of the first three joints of the haptic apparatus. l_1, l_2 and l_3 denote the lengths of the links. We used the simulation of a Geomatic Touch^{TM} haptic device with $l_1 = 132$ mm, $l_2 = 132$ mm and $l_3 = 132$ mm.

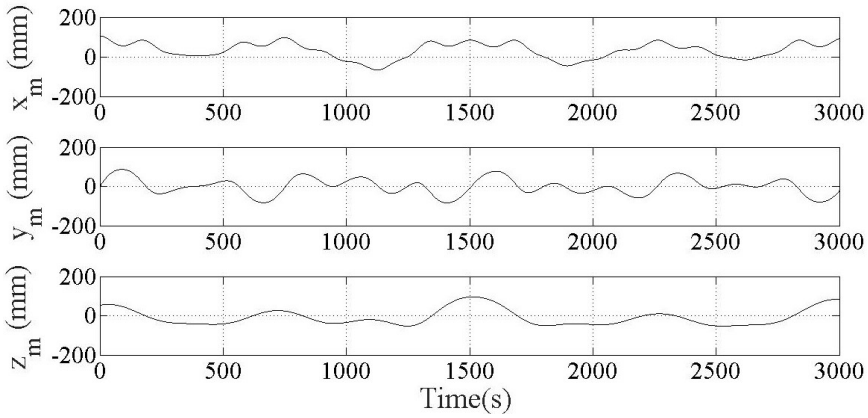


Figure 4.2: Positional components ($x_m,\ y_m,\ z_m$) at the master haptic hand–controller along $X_R,\ Y_R$ *and* Z_R.

In addition to the position data of the haptic end–effector, a set of force was also produced based on the location of hand–controller in its reference coordinate system, *i.e.*, (X_R, y_R, Z_R) in Fig. 4.1. The force vector was generated based on the concept of the forbidden–region virtual fixtures [48] according to the following equation:

$$\begin{cases} f(x) = k_x x_m, & \text{if } x_m > 0 \\ f(y) = k_y y_m, & \text{if } y_m > 0. \\ f(z) = k_z z_m, & \text{if } z_m > 0 \end{cases} \tag{4.2}$$

According to Eq. (4.2), when any positional component has a negative value, no force is generated along the corresponding axis. The simulated Geometric Touch^{TM} haptic hand–controller has a maximum exert able force and torque of 1.8 lbf and 7.9 N, respectively. Using Eqs. (4.1) and (4.2), the packet data, sent to the slave site on the ISS, contains the information of position and force of the haptic on Earth.

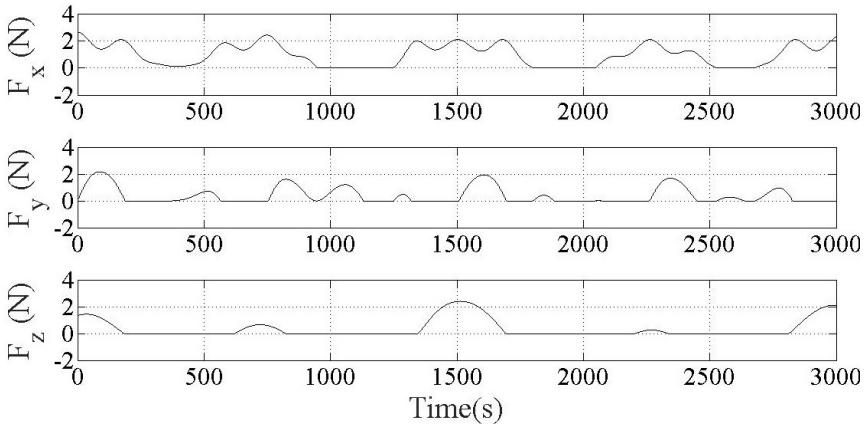


Figure 4.3: Force components at the master haptic hand–controller, ($F_x,\ F_y,\ F_z$).

On application front, using the HTS, a robotic arm in the ISS could remotely be controlled by the expert team, located on Earth, by moving the haptic hand–controller a given task. In conventional technique, an operator normally uses visual information to conduct the task; however, using haptics, when kinematic and/or dynamic as well as visual and audio information of one site (master or slave) is transferred to the other site, a sense of telepresence is provided to the operator to help them perceive the remote environment on the ISS [49] (as presented in Eq. 4.2). The addition of

haptic feedback has been shown to be more valuable than visual feedback [50–53].

4.2 Experimental Procedure

In this experiment, the data packet is sent from the ground station using the satellite net device and received in the ISS. For testing the HTS platform, the three actuators of the haptic device were given a trigonometric angular displacement to follow. Figs. 4.2 and 4.3 illustrate the positional ($x_m,\ y_m,\ z_m$) and force ($F_x,\ F_y,\ F_z$) components of the haptic end–effector over 1-hour of experiment. The sample is taken at the rate of 320 Hz therefore 320 rows generated per second. Each row contains six variables: three positional and three force components. Thus, the size of TCP data packet is considered as 1000 bytes therefore 20 rows of data is accommodate in one packet as six `double` takes 48 bytes of storage in the memory. The receiving side would receive the same data as TCP is used as transport protocol, but the delay would be added to the input signal. Therefore, the main findings of that work is analyzing the changes occur in RTT and congestion window in different position of the ISS around the Earth.

4.3 TCP Variants

As mentioned in Chapter 2, SCPS-TP uses TCP Vegas for congestion controlling with some modification. Therefore, in this work TCP Vegas is used as the transport control protocol to send data to ISS generated by HTS program. Another variant of TCP known TCP HighSpeed which is also used nowadays in the large delay network or Long Fat Networks (LFN) [54]. The simulator also tested against the User Datagram Protocol (UDP) to observe the throughput of the system.

Table 4.1: Simulation parameters used in sending HTS data from Earth to the ISS

Name	Symbol	Value
Transmit power	P_t	30 dBm
Frequency	f	2.3 GHz
Data rate	R	$256Kbps$
Wave length	λ	$0.0767m$
Ground station antenna gain	$G_{[Diameter=18m]}$	51.19 dB
TDRS antenna gain	$G_{[Diameter=4.9m]}$	39.89 dB
ISS antenna gain	$G_{[Diameter=2m]}$	32.109 dB
Antenna efficiency	η	60%
Boltzman constant	k	$-228.6dBHzk^{-1}$
Additional losses	l_{add}	-5 dB
Line loss	l_l	-0.5 dB
Noise figure	NF	3 dB
Noise floor	N_F	-73.431 dBm
Minimum detectable signal	MDS	-62.5 dBm
Room temperature	T_0	290 K
Amplifier noise temperature	T_{Amp}	288.626 K
Receiver temperature	T_{Rx}	293.15 K
Sky noise temperature	T_{sky}	40 K
Galactic noise temperature	T_G	2.73 K
Earth noise temperature	T_E	60 K
Antenna temperature	T_A	102..7 K
System noise temperature	T_S	412.037 K
Atmospheric attenuation	L_a	10 dB

4.4 Simulation Setup

The data generated in the HTS platform at 320 Hz rate and each sample consists of six parameters stored as `double` data type. Therefore, each sample gives 48 bytes of data. In the simulation, 20 rows of data or 960 bytes, put in a TCP packet of length 1000 bytes and transmit to the receiver side or in this case ISS. The maximum delay in all cases is 1 s and data bandwidth is 256 Kbps, therefore Bandwidth Delay Product (BDP) becomes 32 Kbps. Thus, TCP can send 32 packets without getting acknowledgement. After sending packets, it will wait for the acknowledgement. The

parameters used in the link budget equation in the simulation is given in Table 4.1. The simulation parameters are not varied as the main focus of this work is to develop a platform which generate the actual ISS trajectory and obtain the time delay from Earth to ISS.

For UDP, the sender program in the simulation generates data at the rate of 128 Kbps, therefore UDP would not overflow the network as the maximum data rate of the channel is 256 Kbps.

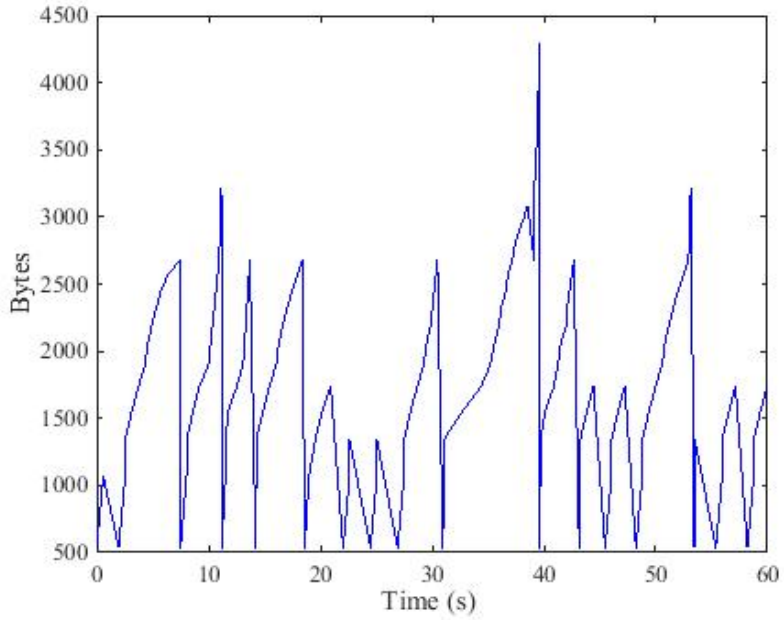
4.5 Results

The HTS data is sent over the three path scenario depicted in chapter 2 using TCP Vegas, TCP HighSpeed and UDP. The results are discussed in path wise basis. Therefore, the performance of two TCP variants would be easier to distinguish.

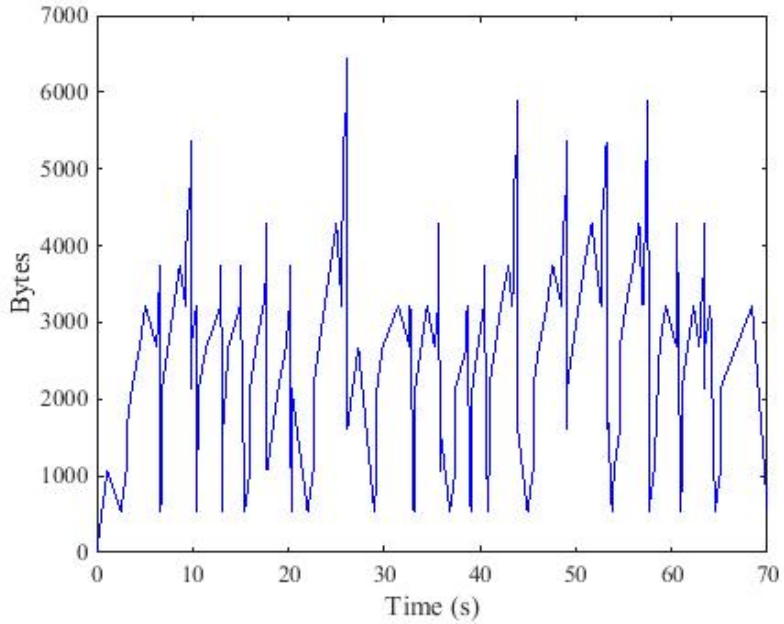
4.5.1 Analysis of Results for Path 1

As discussed in Chapter 2, path 1 is denoted the simulation setup when ISS is communicated with the ground station via only TDRS-1. From the Fig. 4.4, it becomes obvious that TCP HighSpeed tries to increase the congestion more faster than TCP Vegas. In Fig. 4.5, the RTT of two TCP variation is plotted where the mean $\pm$ SD for RTT in TCP Vegas is 670 ± 24.2599 ms and in TCP HighSpeed is 652 ± 19.8848 ms. Fig. **??** shows the RTT histograms for both TCP variants in path 1. In the throughput measurement, TCP HighSpeed provided higher throughput than the TCP Vegas. Throughput of these two variants are depicted in Table 4.3. From Table 4.3, it is observed that, UDP provides the maximum throughput of 100.36 Kbps which is obvious, as there is no retransmission of packets occurred.

The Round Trip Time (RTT) for path 1 is minimum 542 ms and maximum 743



(a) Changes in congestion window for TCP Vegas



(b) Changes in congestion window for TCP HighSpeed

Figure 4.4: Changes in congestion window in path 1 which is ground station to TDRS-1 to ISS.

ms for both TCP variants. But the mean ± SD for TCP Vegas is larger than TCP HighSpeed which is given in Table 4.2.

4.5.2 Analysis of Results for Path 2

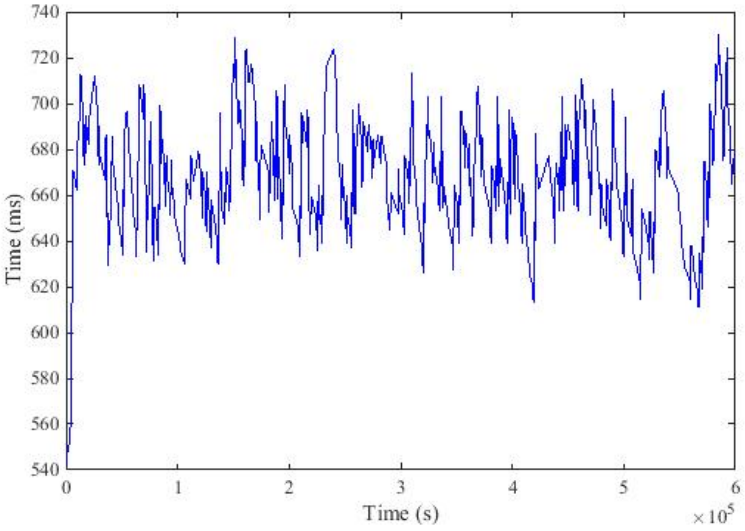
Path 2 is the time when ISS is more close to TDRS-2 than TDRS-1, therefore TDRS-1 relays data to TDRS-2 and TDRS-2 deliver the packet to ISS. Therefore, delay increases when ISS is communicated via TDRS-2. Fig. 4.7 shows the saw tooth behavior of the TCP congestion window in both of the TCP variants. In terms of RTT, TCP HighSpeed give lower value than TCP Vegas which is 702.2514 ± 25.9822. The RTT variates in TCP Vegas from 634 ms to 988 ms which is visible in Fig.4.8. The distribution of the RTT is also visible in Fig. 4.9.

4.5.3 Analysis of Results for Path 3

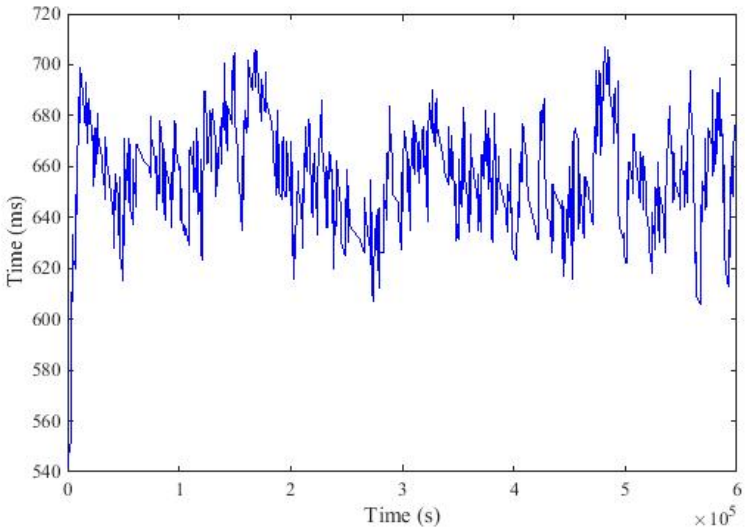
In path 3, ISS receives data from TDRS-3. Therefore, the delay changes between Earth and ISS which is shown in Table 4.2. The maximum RTT is observed in the path 3 which is 1107 ms and the throughput in path 3 drops drastically to 5.47488 Kbps for Tcp HighSpeed. The mean RTT for TCP Vegas is 767.5596 ms and for TCP HighSpeed is 751.4011 ms. The RTT histogram for TCP Vegas also depicts the variations in RTT in Fig. 4.12. The behavior of the congestion window seems regular, which means TCP variants works with the developed platform without any aberration.

4.6 Observations

After observing the changes in congestion window, RTT and throughput values in all three different communication settings based on the position of ISS, an observation

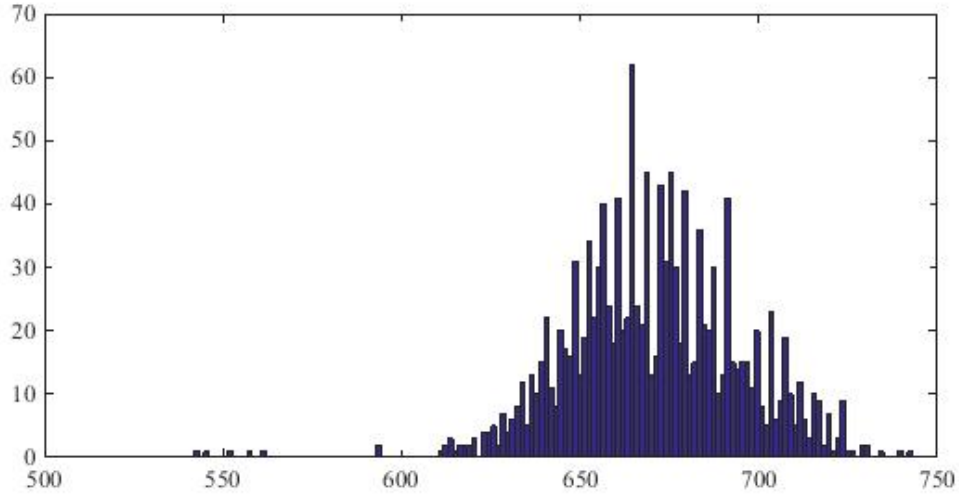


(a) Change of RTT for TCP Vegas

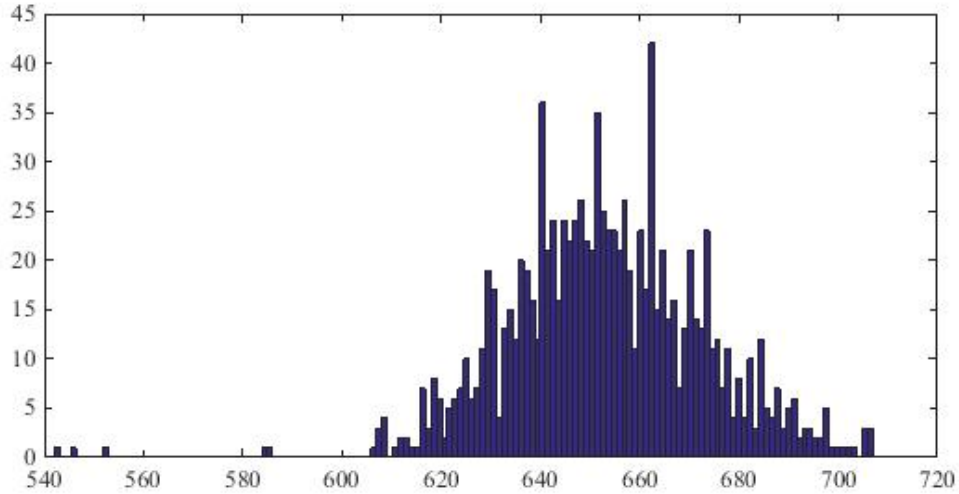


(b) Change of RTT for TCP HighSpeed

Figure 4.5: Change of RTT in path 1 which is ground station to TDRS-1 to ISS.

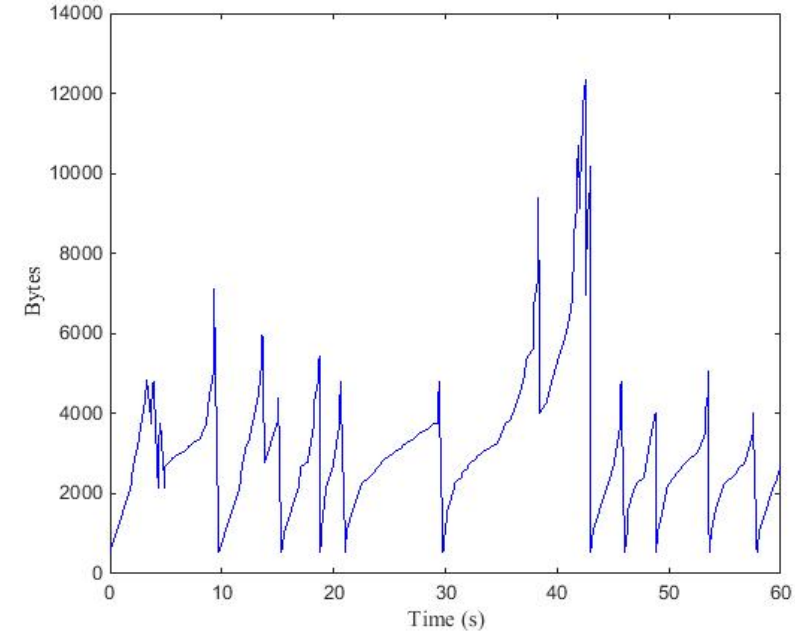


(a) RTT histogram for TCP Vegas

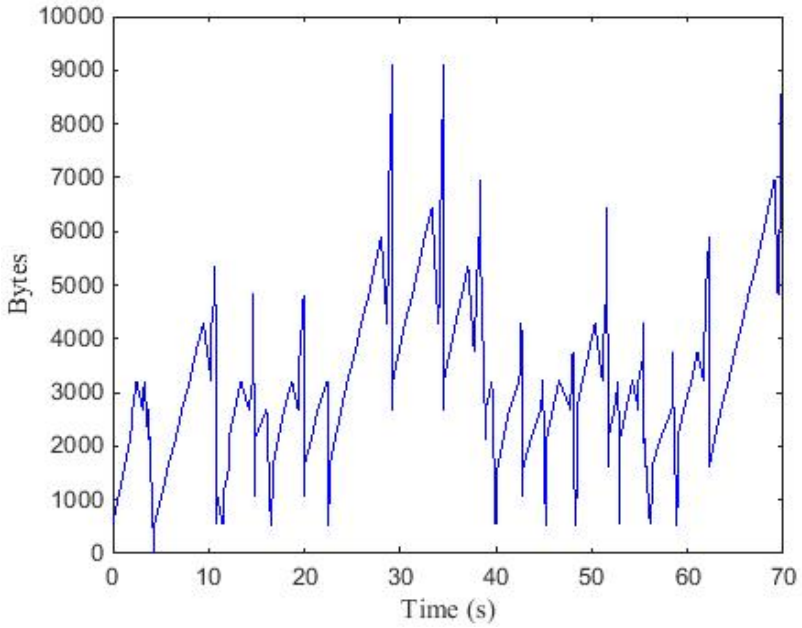


(b) RTT histogram for TCP HighSpeed

Figure 4.6: RTT histogram in path 1 which is ground station to TDRS-1 to ISS.

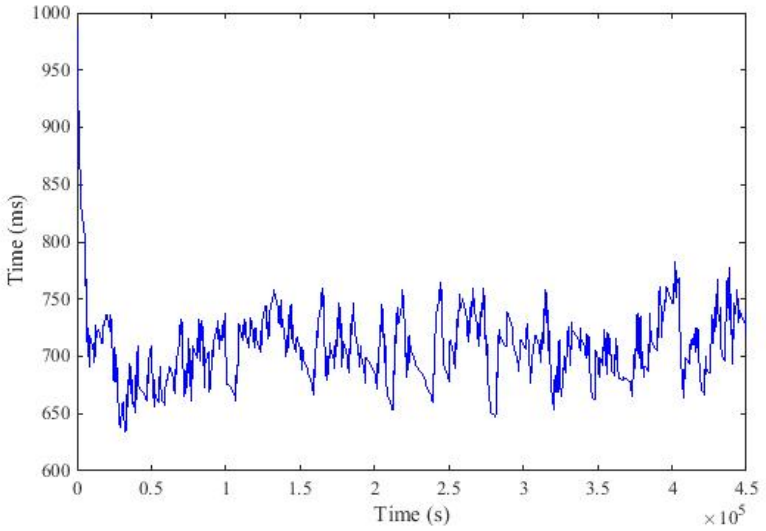


(a) Changes in congestion window for TCP Vegas

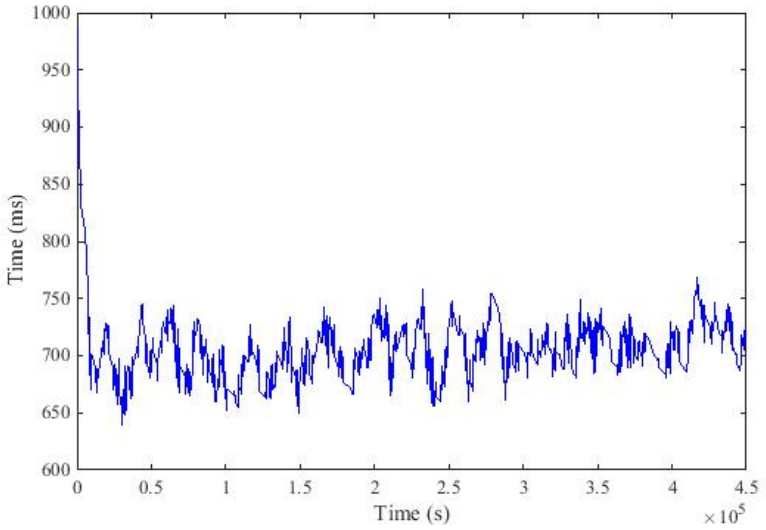


(b) Changes in congestion window for TCP HighSpeed

Figure 4.7: Changes in congestion window in path 2 which is ground station to TDRS-1 to TDRS-2 and then ISS.

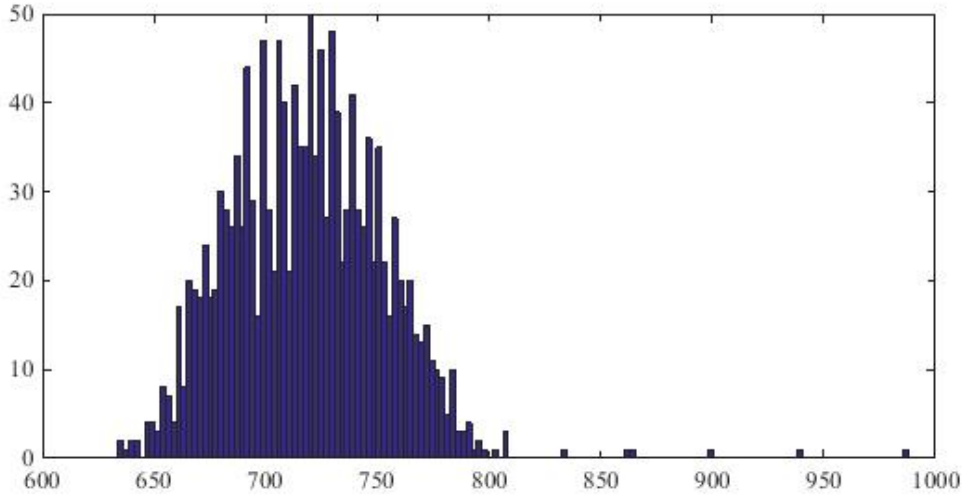


(a) Change of RTT for TCP Vegas

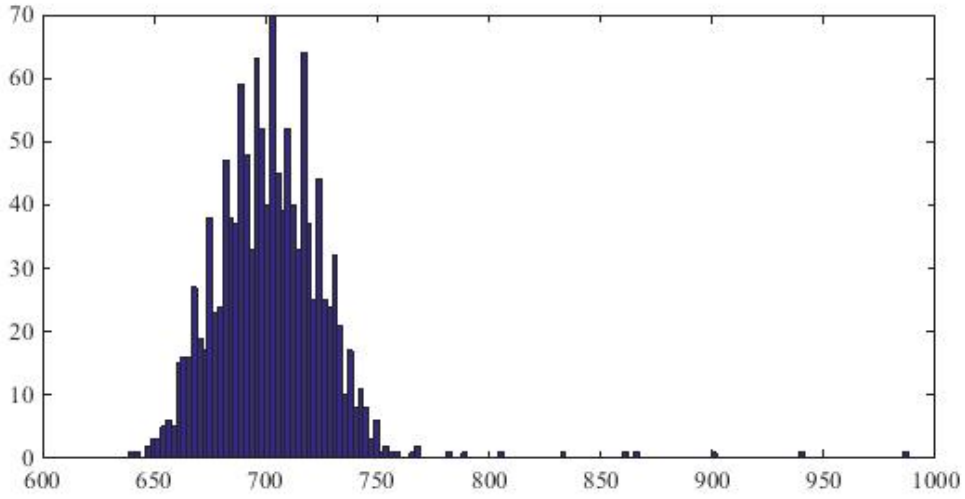


(b) Change of RTT for TCP HighSpeed

Figure 4.8: Change of RTT in path 2 which is ground station to TDRS-1 to TDRS-2 and then ISS.

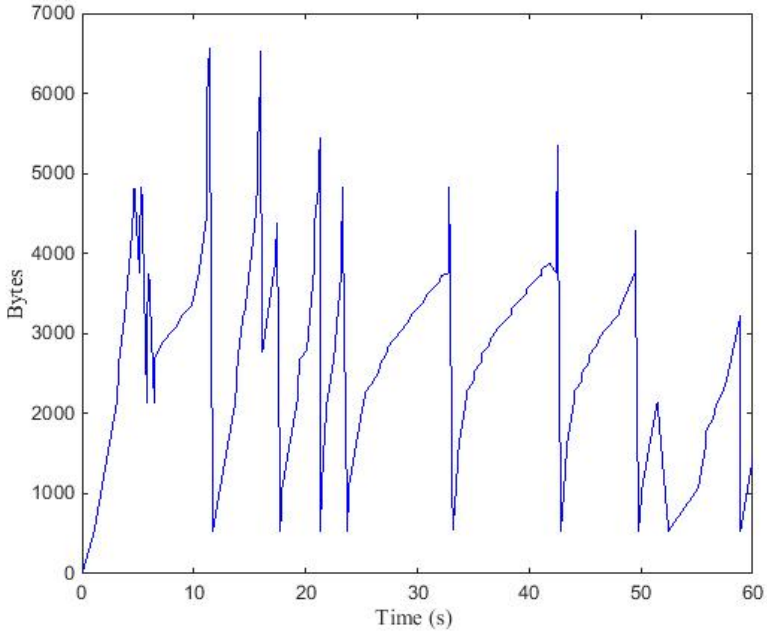


(a) RTT histogram for TCP Vegas

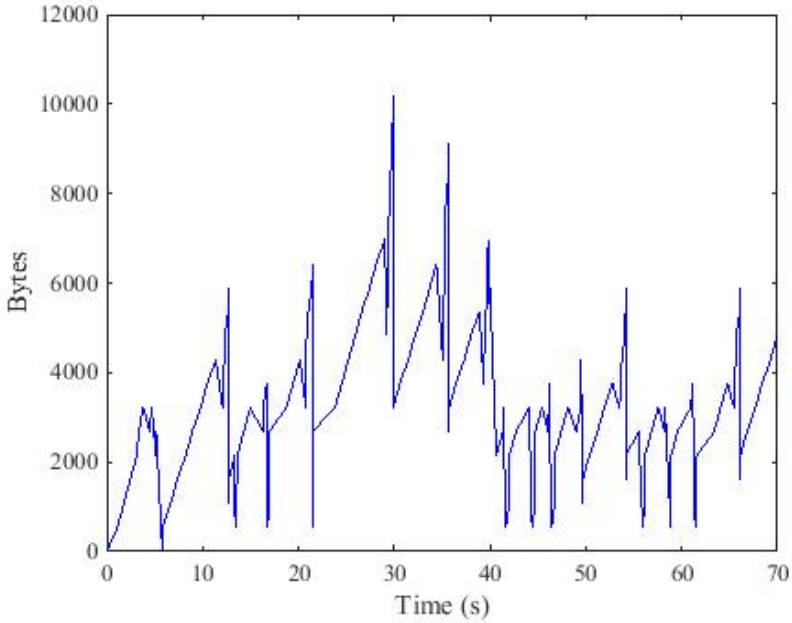


(b) RTT histogram for TCP HighSpeed

Figure 4.9: RTT histogram in path 2 which is ground station to TDRS-1 to TDRS-2 and then ISS.

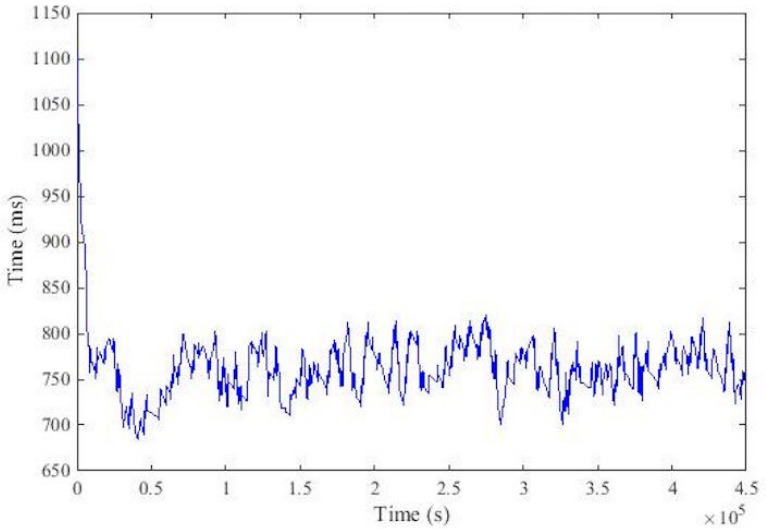


(a) Changes in congestion window for TCP Vegas

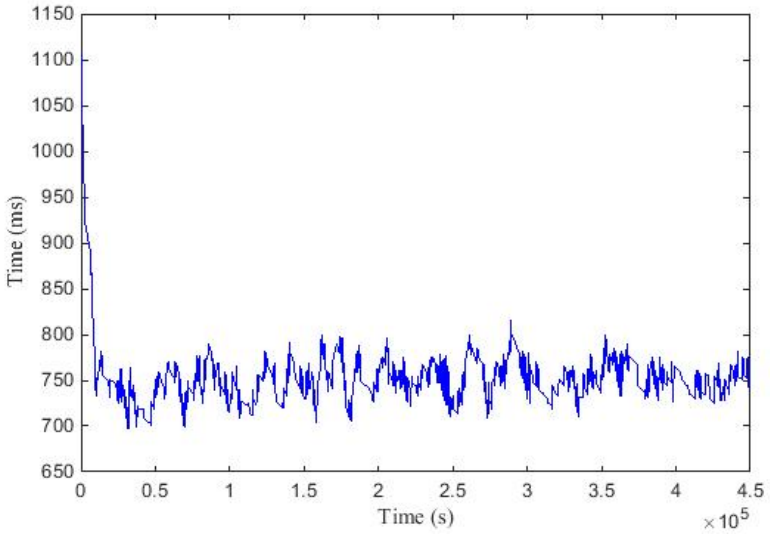


(b) Changes in congestion window for TCP HighSpeed

Figure 4.10: Changes in congestion window in path 3 which is ground station to TDRS-1 to TDRS-3 and then ISS.

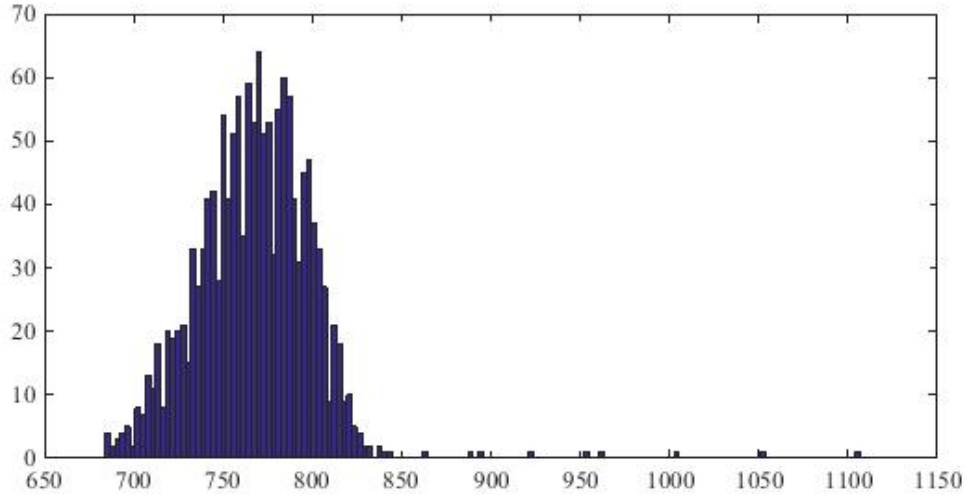


(a) Change of RTT for TCP Vegas

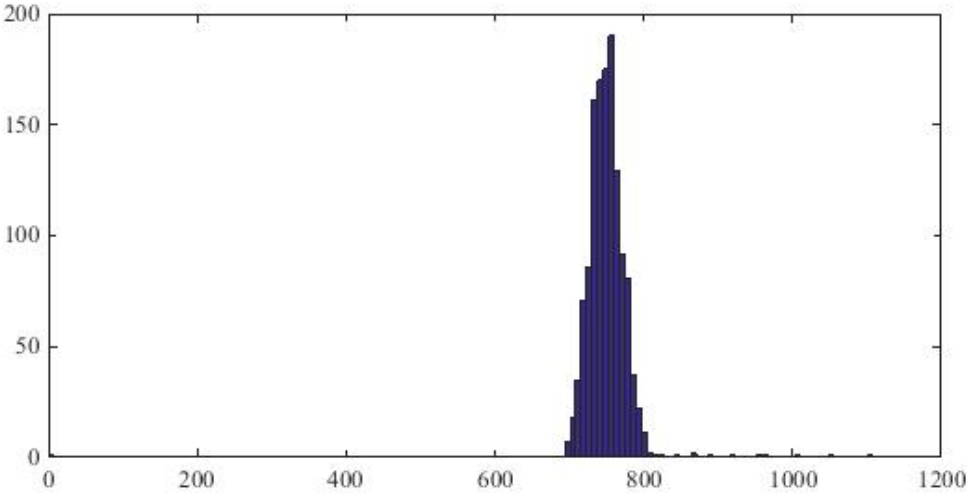


(b) Change of RTT for TCP HighSpeed

Figure 4.11: Change of RTT in path 3 which is ground station to TDRS-1 to TDRS-3 and then ISS.



(a) RTT histogram for TCP Vegas.



(b) RTT histogram for TCP HighSpeed

Figure 4.12: RTT histogram in path 3 which is ground station to TDRS-1 to TDRS-3 and then ISS.

Table 4.2: Changes in round trip time

Path	TCP Variant	Minimum	Maximum	SD	Mean
Path 1	TCP Vegas	542	743	24.2599	670.6091
	TCP High Speed	542	743	19.8848	652.9981
Path 2	TCP Vegas	634	988	34.6565	718.5422
	TCP High Speed	639	988	25.9822	702.2514
Path 3	TCP Vegas	684	1107	33.1397	767.5596
	TCP High Speed	696	1107	27.581	751.4011

Table 4.3: Throughput for three different paths employing different transport protocols.

Path	TCP Variant	Throughput
Path 1	TCP Vegas	7.84645 Kbps
	TCP High Speed	7.98732 Kbps
	UDP	100.36 Kbps
Path 2	TCP Vegas	16.8964 Kbps
	TCP High Speed	16.86 Kbps
	UDP	119.962 Kbps
Path 3	TCP Vegas	5.48374 Kbps
	TCP High Speed	5.47488 Kbps
	UDP	103.304 Kbps

can be made that, the RTT between Earth and ISS is vary from 542 ms to 1107 ms which gives lowest mean $\pm$ SD 652.9981 ± 19.8848 ms and highest mean $\pm$ SD is 767.5596 ± 33.1397 ms. Therefore, the maximum delay between Earth and ISS is 1 s. Thus, a haptic teleoperation which can tolerate up to 1 s delay can be implemented if there is no error occurred.

Again, it is easily deducible from Table 4.2 and Table 4.3 that, TCP HighSpeed performs better in terms of mean RTT. But the throughput of TCP highspeed slightly falls behind TCP Vegas when number of hop increases.

In this simulation, unmodified TCP Vegas is adopted. Therefore, the result gives the lower bound in terms of throughput, as SCPS-TP uses some more modification along with TCP Vegas, described in Chapter 2.

Chapter 5

Conclusion and Future Work

5.1 Concluding Remarks

In this work, by developing a satellite communication module in ns-3, the communication channel between the Earth and the International Space Station (ISS) is simulated, to determine the feasibility of the haptic communication over the space. Towards the development of the network simulator, the transport protocols for space have been investigated, followed by calculating the link budgets of the satellite channel, developing a new module in ns-3, and incorporating actual trajectory of the ISS. As the ISS is following it's actual orbits around the Earth, and ground station is placed in the exact location on the Cartesian coordinate system as per it's longitude and latitude, the delay measurement is fairly accurate, compared to the actual time delay. The satellite communication module has been implemented from scratch, and the delay effect and packet loss on the HTS were examined. The results have showed that, UDP provides maximum throughput and TCP HighSpeed offers the minimum RTT. Moreover, with the increment of the hops, TCP HighSpeed lagged slightly behind the TCP Vegas.

5.2 Future Research Directions

The developed module provides a base, for the future study of satellite communication in ns–3. Some of the possible future work would be as follows,

- Implementing the SCPS-TP in ns-3 and testing against the developed satellite module.
- Investigating the atmospheric attenuation in more details, therefore the Ku band could be used as the transmission frequency. The Ku band is more susceptible to atmospheric gas, clouds, fogs and humidity [36], therefore an exact atmospheric attenuation should be calculated before implementing Ku band in satellite communication.
- Incorporating the live weather data fetching feature using C++ Application Programming Interface (API) provided by *Worldwide Weather* website. The weather data would be fetched based on the position of the ISS above the Earth. Live weather data then used in the atmospheric attenuation calculation.
- Implementing a `SatelliteMobility` model in ns-3 which would support state vectors. As discussed in Chapter 3, ns-3 only support Cartesian coordinate system, therefore the other elements of state vectors mentioned in Subsection 2.6.2 remains unused.
- Incorporating the multi-hop TCP. Satellite channels are error prone, and ISS is communicating with Earth using Tracking and Data Relay Satellite System (TDRSS), where each node or hop is situated thousands of kilometers away from each other. In this work, the haptic communication has been done with an end to end TCP connection. The data packet and the acknowledgement of that

packet travels hop-to-hop basis, therefore if the packet is lost in between any hop the transmitter has to retransmit the packet after time out occurred. Thus, the throughput of the system reduces. If the custody based communication or the multi-hop TCP is introduced in the future work, then the throughput of the system possibly increases. The multi-hop TCP means when ground station transmit data to TDRS-1, TDRS-1 will take the full responsibility for delivering that packet to the next hop and send acknowledge to the ground station back. For that reason, it is known as multi-hop TCP or custody based communication.

- Including the interference model in the channel. In this work, the satellite point-to-point communication model has been implemented and it has been assumed, that there is no other interfering channels. Therefore, in future work, the channel should be extended, to take into account interference in the communication channel.

Bibliography

[1] J. Nelson, "Mars exploration rover," 2014.

[2] J. Nelson, "Mars exploration rover -opportunity," 2014.

[3] "Case study: Mda - canadian space arm," 2002.

[4] N. Sato and S. Doi, "Jem remote manipulator system(jemrms) human-in-the-loop test," in *International Symposium on Space Technology and Science, 22 nd, Morioka, Japan*, pp. 1195–1199, 2000.

[5] P. Laryssa, E. Lindsay, O. Layi, O. Marius, K. Nara, L. Aris, and T. Ed, "International space station robotics: a comparative study of era, jemrms and mss," in *Proc. 7th ESA Workshop Advanced Space Technologies Robotics Automation*, 2002.

[6] F. Didot, M. Oort, J. Kouwen, and P. Verzijden, "The era system: Control architecture and performance results," in *Proc. 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Montral, Canada*, Citeseer, 2001.

[7] A. Flores-Abad, O. Ma, K. Pham, and S. Ulrich, "A review of space robotics technologies for on-orbit servicing," *Progress in Aerospace Sciences*, vol. 68, pp. 1–26, 2014.

[8] M. Bualat, W. Carey, T. Fong, K. Nergaard, C. Provencher, A. Schiele, P. Schoonejans, and E. Smith, "Preparing for crew-control of surface robots from orbit," in *IAA Space Exploration Conference*, 2014.

[9] E. Ackerman, "Astronaut aboard the iss controls a robot on earth using haptic feedback," *IEEE Spectrum*, 2015.

[10] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl, "Rotex-the first remotely controlled robot in space," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 2604–2611, IEEE, 1994.

[11] G. Hirzinger, K. Landzettel, D. Reintsema, C. Preusche, A. Albu-Schaeffer, B. Rebele, and M. Turk, "Rokviss-robotics component verification on iss," in *Proc. 8th Int. Symp. Artif. Intell. Robot. Autom. Space (iSAIRAS)(Munich 2005) p. Session2B*, 2005.

[12] W. Bluethmann, R. Ambrose, M. Diftler, S. Askew, E. Huber, M. Goza, F. Rehnmark, C. Lovchik, and D. Magruder, "Robonaut: A robot designed to work with humans in space," *Autonomous robots*, vol. 14, no. 2, pp. 179–197, 2003.

[13] M. A. Diftler, J. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. Permenter, *et al.*, "Robonaut 2-the first humanoid robot in space," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2178–2183, IEEE, 2011.

[14] M. Diftler, T. Ahlstrom, R. Ambrose, N. Radford, C. Joyce, N. De La Pena, A. Parsons, and A. Noblitt, "Robonaut 2initial activities on-board the iss," in *Aerospace Conference, 2012 IEEE*, pp. 1–12, IEEE, 2012.

[15] E.-C. Lovasz, D. T. Mărgineanu, V. Ciupe, I. Maniu, C. M. Gruescu, E. S. Zăbavă, and S. D. Stan, "Design and control solutions for haptic elbow exoskeleton module used in space telerobotics," *Mechanism and Machine Theory*, vol. 107, pp. 384–398, 2017.

[16] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[17] J. Puttonen, S. Rantanen, F. Laakso, J. Kurjenniemi, K. Aho, and G. Acar, "Satellite model for network simulator 3," in *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pp. 86–91, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[18] G. S. Fishman, "Principles of discrete event simulation.[book review]," 1978.

[19] G. T. Heineman and W. T. Councill, *Component-based software engineering*. Springer, 2001.

[20] E. Weingartner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *Communications, 2009. ICC'09. IEEE International Conference on*, pp. 1–5, IEEE, 2009.

[21] G. F. Riley, "Simulation of large scale networks ii: large-scale network simulations with gtnets," in *Proceedings of the 35th conference on Winter simulation: driving innovation*, pp. 676–684, Winter Simulation Conference, 2003.

[22] D. Kachan, "Integration of ns-3 with matlab/simulink," 2010.

[23] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, 2008.

[24] W. R. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," 1997.

[25] A. S. D. L. Protocol, "Ccsds 732.0-b-2 blue book," *Consultative Committee for Space Data System*, 2006.

[26] C. Partridge, *Gigabit networking*. Addison-Wesley Professional, 1994.

[27] R. Fox, "Tcp big window and nak options," 1989.

[28] R. C. Durst, G. J. Miller, and E. J. Travis, "Tcp extensions for space communications," *Wireless Networks*, vol. 3, no. 5, pp. 389–403, 1997.

[29] G. H. Kitmacher, *Reference guide to the international space station.* 2006.

[30] B. Harvey, *The rebirth of the Russian space program: 50 years after Sputnik, new frontiers.* Springer Science & Business Media, 2007.

[31] J. E. Catchpole, *The international space station: building for the future.* Springer Science & Business Media, 2008.

[32] A. A. Atayero, M. K. Luka, and A. A. Alatishe, "Satellite link design: A tutorial," *International Journal of Electrical & Computer Sciences*, vol. 11, no. 4, pp. 1–6, 2011.

[33] W. J. Larson and J. R. Wertz, "Space mission analysis and design," tech. rep., Microcosm, Inc., Torrance, CA (US), 1992.

[34] E. A. Mueller and A. L. Sims, "Relationships between reflectivity, attenuation, and rainfall rate derived from drop size spectra," tech. rep., DTIC Document, 1969.

[35] R. Crane, "Propagation phenomena affecting satellite communication systems operating in the centimeter and millimeter wavelength bands," *Proceedings of the IEEE*, vol. 59, no. 2, pp. 173–188, 1971.

[36] R. Sector, "International telecommunication union (itu)," *General Aspects of Digital Transmission Systems–Terminal Equipments. Pulse Code Modulation of Voice Frequencies. ITU-T Recommendation G*, vol. 711, 2013.

[37] X. Chen, "Study of system noise temperature from 50 mhz to 15 ghz with application to eleven antennas," 2007.

[38] H. W. Ott and H. W. Ott, *Noise reduction techniques in electronic systems*, vol. 442. Wiley New York, 1988.

[39] G. Maral and M. Bousquet, *Satellite Communication Systems: Systems, Techniques and Technology.* Wiley, 5 ed., 2009.

[40] L. J. Ippolito Jr, *Satellite communications systems engineering: atmospheric effects, satellite link design and system performance*, vol. 6. John Wiley & Sons, 2008.

[41] B. L. Z. Ding and B. Lathi, "Modern digital and analog communication systems," 2009.

[42] N. Dershowitz and E. M. Reingold, *Calendrical calculations*. Cambridge University Press, 2008.

[43] M. Soffel, S. A. Klioner, G. Petit, P. Wolf, S. Kopeikin, P. Bretagnon, V. Brumberg, N. Capitaine, T. Damour, T. Fukushima, *et al.*, "The iau 2000 resolutions for astrometry, celestial mechanics, and metrology in the relativistic framework: explanatory supplement," *The Astronomical Journal*, vol. 126, no. 6, p. 2687, 2003.

[44] P. K. Seidelmann, *Explanatory supplement to the astronomical almanac*. University Science Books, 2005.

[45] R. A. Nelson, D. D. McCarthy, S. Malys, J. Levine, B. Guinot, H. F. Fliegel, R. L. Beard, and T. Bartholomew, "The leap second: its history and possible future," *Metrologia*, vol. 38, no. 6, p. 509, 2001.

[46] A. Almanac, "Us naval observatory," *Washington DC*, 1975.

[47] D. Vallado and P. Crawford, "Sgp4 orbit determination," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, p. 6770, 2008.

[48] Y. Maddahi, K. Zareinia, and N. Sepehri, "An augmented virtual fixture to improve task performance in robot-assisted live-line maintenance," *Computers & Electrical Engineering*, vol. 43, pp. 292–305, 2015.

[49] D. A. Lawrence, "Stability and transparency in bilateral teleoperation," *IEEE transactions on robotics and automation*, vol. 9, no. 5, pp. 624–637, 1993.

[50] O.-Y. Ming, D. V. Beard, and F. P. Brooks, "Force display performs better than visual display in a simple 6-d docking task," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pp. 1462–1466, IEEE, 1989.

[51] M. C. Yip, M. Tavakoli, and R. D. Howe, "Performance analysis of a haptic telemanipulation task under time delay," *Advanced Robotics*, vol. 25, no. 5, pp. 651–673, 2011.

[52] A. Hamam, M. Eid, and A. El Saddik, "Effect of kinesthetic and tactile haptic feedback on the quality of experience of edutainment applications," *Multimedia tools and applications*, vol. 67, no. 2, pp. 455–472, 2013.

[53] H. I. Son, L. L. Chuang, J. Kim, and H. H. Bülthoff, "Haptic feedback cues can improve human perceptual awareness in multi-robots teleoperation," in *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*, pp. 1323–1328, IEEE, 2011.

[54] S. Floyd, "Highspeed tcp for large congestion windows," 2003.

Appendix A

`Install()` Method in Helper Class

In this method helper object take two arguments and use `ObjectFactory` class instances to create particular object of that class. It also assign Mac address and queue object in those net device instances.

```
NetDeviceContainer
    SatellitePointToPointHelper::Install(Ptr<Node> a, Ptr<Node> b) {
        NetDeviceContainer container;
        Ptr<SatellitePointToPointNetDevice> devA =
            m_deviceFactory.Create<SatellitePointToPointNetDevice> ();
        devA->SetAddress(Mac48Address::Allocate());
        a->AddDevice(devA);
        Ptr<Queue> queueA = m_queueFactory.Create<Queue> ();
        devA->SetQueue(queueA);
        Ptr<SatellitePointToPointNetDevice> devB =
            m_deviceFactory.Create<SatellitePointToPointNetDevice> ();
        devB->SetAddress(Mac48Address::Allocate());
        b->AddDevice(devB);
        Ptr<Queue> queueB = m_queueFactory.Create<Queue> ();
```

```
devB->SetQueue(queueB);
// If MPI is enabled, we need to see if both nodes have the same
    system id
// (rank), and the rank is the same as this instance. If both are
    true,
//use a normal p2p channel, otherwise use a remote channel
bool useNormalChannel = true;
Ptr<SatellitePointToPointChannel> channel = 0;
if (MpiInterface::IsEnabled()) {
    uint32_t n1SystemId = a->GetSystemId();
    uint32_t n2SystemId = b->GetSystemId();
    uint32_t currSystemId = MpiInterface::GetSystemId();
    if (n1SystemId != currSystemId || n2SystemId != currSystemId) {
        useNormalChannel = false;
    }
}
if (useNormalChannel) {
    channel = m_channelFactory.Create<SatellitePointToPointChannel>
        ();
} else {
    channel =
        m_remoteChannelFactory.Create<SatellitePointToPointRemoteChannel>
        ();
    Ptr<MpiReceiver> mpiRecA = CreateObject<MpiReceiver> ();
    Ptr<MpiReceiver> mpiRecB = CreateObject<MpiReceiver> ();
    mpiRecA->SetReceiveCallback(MakeCallback(\&SatellitePointToPointNetDevice::Re
        devA));
```

```
        mpiRecB->SetReceiveCallback(MakeCallback(\&SatellitePointToPointNetDevice::Re
            devB));
        devA->AggregateObject(mpiRecA);
        devB->AggregateObject(mpiRecB);
    }
    AddPropagationLoss("ns3::FriisPropagationLossModel", "Frequency",
        DoubleValue(2.3e9));
    SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
    channel->SetPropagationDelayModel(m_propagationDelay.Create<PropagationDelayModel
    channel->SetPropagationLossModel(m_propagationLoss.Create<FriisPropagationLossMod
    Ptr<SatellitePhysical> phyA =
        m_physicalFactory.Create<SatellitePhysical>();
    phyA->SetChannel(channel);
    phyA->SetNetDevice(devA);
    devA->SetPhysical(phyA);
    Ptr<SatellitePhysical> phyB =
        m_physicalFactory.Create<SatellitePhysical>();
    phyB->SetChannel(channel);
    phyB->SetNetDevice(devB);
    devB->SetPhysical(phyB);
    devA->Attach(channel);
    devB->Attach(channel);
    container.Add(devA);
    container.Add(devB);
    return container;
}
```

Appendix B

Program to Generate State Vectors from TLE

In the following code snippet, the generation of state vectors for all the satellites are given. The position is calculated in every second and stored in a csv file for later use.

```
double siteLat, siteLon, siteAlt, siteLatRad, siteLonRad;
    //ENTER SITE DETAILS HERE : I put white sands location in decimal
        which is
    //White Sands Ground Terminal (WSGT) 32.5007N 106.6086W
    siteLat = 32.512027777777774; //+North (Austin ->30.25N)
    siteLon = -105.3886388888888; //+East (Austin ->-97.75)
    siteAlt = 0.15; //km
    siteLatRad = siteLat * pi / 180.0;
    siteLonRad = siteLon * pi / 180.0;



    double latlongh[3]; //lat, long in rad, h in km above ellipsoid
    double tdrs8_latlongh[3]; //lat, long in rad, h in km above ellipsoid
    double tdrs9_latlongh[3]; //lat, long in rad, h in km above ellipsoid
    double tdrs10_latlongh[3]; //lat, long in rad, h in km above ellipsoid
```

```
    //</editor-fold>
/**
 * TDRS 8
1 26388U 00034A 17013.59608280 -.00000247 00000-0 00000+0 0 9997
2 26388  7.2325 57.1835 0009991 212.2588 147.6815 1.00274321 60663

TDRS 9
1 27389U 02011A 17013.90048035 -.00000116 00000-0 00000-0 0 9993
2 27389  4.9338 81.0900 0022850 231.3684 113.2557 1.00271631 55937

TDRS 10
1 27566U 02055A 17013.33419306 .00000063 00000-0 00000-0 0 9998
2 27566  4.7666 59.1125 0013477 216.3876 143.4830 1.00266459 51726
 *
 * @param argc
 * @param argv
 * @return
 */
    //ENTER TWO-LINE ELEMENT HERE
    //<editor-fold defaultstate="collapsed" desc="TLE">
    //ISS TLE-last2 digit of year and day of the year in julean format..
        16 from 2016 and 344 is the juLean day
    char longstr1[] = "1 25544U 98067A 17070.76826441 .00004009 00000-0
        67626-4 0 9997";
    char longstr2[] = "2 25544 51.6426 163.9162 0006947 279.4609 222.8623
        15.54188847 46609";
```

```
char tdrs8_longstr1[] = "1 26388U 00034A 17069.44246042 -.00000254
    00000-0 00000+0 0 9994";
char tdrs8_longstr2[] = "2 26388 7.3273 56.6785 0007650 242.5635
    117.4975 1.00276825 61229";

char tdrs9_longstr1[] = "1 27389U 02011A 17070.11211106 -.00000127
    00000-0 00000+0 0 9991";
char tdrs9_longstr2[] = "2 27389 5.0482 80.3244 0022530 243.4506
    233.4023 1.00271188 56496";

char tdrs10_longstr1[] = "1 27566U 02055A 17069.18155452 .00000051
    00000-0 00000+0 0 9997";
char tdrs10_longstr2[] = "2 27566 4.8634 58.8470 0012012 233.6093
    126.5185 1.00271761 52276";

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc="R and V vector">
double ro[3]; // R -position vector for ISS
double vo[3]; // V- velocity vector for ISS

double tdrs8_ro[3]; // R -position vector for tdrs8
double tdrs8_vo[3]; // V- velocity vector for tdrs8

double tdrs9_ro[3]; // R -position vector for tdrs9
double tdrs9_vo[3]; // V- velocity vector for tdrs9
```

```
double tdrs10_ro[3]; // R -position vector for tdrs10
double tdrs10_vo[3]; // V- velocity vector for tdrs10

double ground_ro[3]; // R -position vector
double prevground_ro[3]; // R -position vector
double ground_vo[3]; // V- velocity vector

double recef[3]; //R- vector for ISS in Earth Centered Earth Fixed
    frame or TEME frame
double prevrecef[3]; //R- vector for ISS in Earth Centered Earth Fixed
    frame or TEME frame
double vecef[3]; //V- vector for ISS in Earth Centered Earth Fixed
    frame or TEME frame

double tdrs8_recef[3]; //R- vector for TDRS8 in Earth Centered Earth
    Fixed frame or TEME frame
double prevtdrs8_recef[3]; //R- vector for TDRS8 in Earth Centered
    Earth Fixed frame or TEME frame
double tdrs8_vecef[3]; //V- vector for TDRS8 in Earth Centered Earth
    Fixed frame or TEME frame

double tdrs9_recef[3]; //R- vector for TDRS9 in Earth Centered Earth
    Fixed frame or TEME frame
double prevtdrs9_recef[3]; //R- vector for TDRS9 in Earth Centered
    Earth Fixed frame or TEME frame
double tdrs9_vecef[3]; //V- vector for TDRS9 in Earth Centered Earth
    Fixed frame or TEME frame
```

```
double tdrs10_recef[3]; //R- vector for TDRS10 in Earth Centered Earth
   Fixed frame or TEME frame
double prevtdrs10_recef[3]; //R- vector for TDRS10 in Earth Centered
   Earth Fixed frame or TEME frame
double tdrs10_vecef[3]; //V- vector for TDRS10 in Earth Centered Earth
   Fixed frame or TEME frame

/*
 * razel         Range, azimuth, and elevation matrix
 * razelrates    Range rate, azimuth rate, and elevation rate matrix
 */

double razel[3]; // Range, Azimuth and Elevation Vector for ISS
double razelrates[3]; // Range, Azimuth and Elevation Changing rate
   Vector for ISS

double tdrs8_razel[3]; // Range, Azimuth and Elevation Vector for tdrs8
double tdrs8_razelrates[3]; // Range, Azimuth and Elevation Changing
   rate Vector for tdrs8

double tdrs9_razel[3]; // Range, Azimuth and Elevation Vector for tdrs9
double tdrs9_razelrates[3]; // Range, Azimuth and Elevation Changing
   rate Vector for tdrs9

double tdrs10_razel[3]; // Range, Azimuth and Elevation Vector for
   tdrs10
```

```
double tdrs10_razelrates[3]; // Range, Azimuth and Elevation Changing
    rate Vector for tdrs10
//</editor-fold>

//<editor-fold defaultstate="collapsed" desc="Variables used in sgp4
    model">
char typerun, typeinput, opsmode;

gravconsttype whichconst; // wgs72 or wgs84 World Geodatic System
    constants

double sec, secC, juleanDate, juleandateCurrent, tsince, startmfe,
    stopmfe, deltamin;
double tumin, mu, radiusearthkm, xke, j2, j3, j4, j3oj2;


double tdrs8_juleanDate, tdrs9_juleanDate, tdrs10_juleanDate;
//</editor-fold>


//<editor-fold defaultstate="collapsed" desc="TIME related variable
    initialization">

//time variables from scenario epoch time
int year, mon, day, hr, min;
//current time variables
int yearC, monC, dayC, hrC, minC;
```

```
typedef char str3[4];
str3 monstr[13];
strcpy(monstr[1], "Jan");
strcpy(monstr[2], "Feb");
strcpy(monstr[3], "Mar");
strcpy(monstr[4], "Apr");
strcpy(monstr[5], "May");
strcpy(monstr[6], "Jun");
strcpy(monstr[7], "Jul");
strcpy(monstr[8], "Aug");
strcpy(monstr[9], "Sep");
strcpy(monstr[10], "Oct");
strcpy(monstr[11], "Nov");
strcpy(monstr[12], "Dec");

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc="Structure Declaration
    for satellites">
elsetrec satrec; // ISS related values or structure that holds various
    data related to ISS
elsetrec tdrs8_satrec; // TDRS8 related values or structure that holds
    various data related to TDRS8
elsetrec tdrs9_satrec; // TDRS9 related values or structure that holds
    various data related to TDRS9
elsetrec tdrs10_satrec; // TDRS10 related values or structure that
```

```
    holds various data related to TDRS10
//</editor-fold>

float elevation;
float azimuth; //-180 to 0 to 180



//SET REAL TIME CLOCK (Set values manually using custom excel function
    until I find a way to do it automatically)
//    set_time(1440763200);

//SET VARIABLES
opsmode = 'i';
typerun = 'c';
typeinput = 'e';
whichconst = wgs72;

// initialize wgs constant based on the which constant value, we are
    sending reference variables to initialize
getgravconst(whichconst, tumin, mu, radiusearthkm, xke, j2, j3, j4,
    j3oj2);

//<editor-fold defaultstate="collapsed" desc="SET VARIABLES">

//INITIALIZE SATELLITE TRACKING

//printf("Initializing ISS orbit from TLE...\n");
```

```
twoline2rv(longstr1, longstr2, typerun, typeinput, opsmode,
    whichconst, startmfe, stopmfe, deltamin, satrec);
//printf("twoline2rv function complete for ISS...\n");

//printf("Initializing TDRS8 orbit from it's TLE...\n");
twoline2rv(tdrs8_longstr1, tdrs8_longstr2, typerun, typeinput,
    opsmode, whichconst, startmfe, stopmfe, deltamin, tdrs8_satrec);
//printf("twoline2rv for TDRS8 function complete...\n");

//printf("Initializing TDRS9 orbit from it's TLE...\n");
twoline2rv(tdrs9_longstr1, tdrs9_longstr2, typerun, typeinput,
    opsmode, whichconst, startmfe, stopmfe, deltamin, tdrs9_satrec);
//printf("twoline2rv for TDRS9 function complete...\n");


//printf("Initializing TDRS10 orbit from it's TLE...\n");
twoline2rv(tdrs10_longstr1, tdrs10_longstr2, typerun, typeinput,
    opsmode, whichconst, startmfe, stopmfe, deltamin, tdrs10_satrec);
//printf("twoline2rv for TDRS10 function complete...\n");

//Initialize State Vectors Ro and Vo
//Call propogator to get initial state vector value
sgp4(whichconst, satrec, 0.0, ro, vo);
//printf("SGP4 at t = 0 to get initial state vector complete...\n");

//Call propogator to get initial state vector value
sgp4(whichconst, tdrs8_satrec, 0.0, tdrs8_ro, tdrs8_vo);
```

```
//printf("SGP4 at t = 0 to get initial state vector complete...\n");

//Call propogator to get initial state vector value
sgp4(whichconst, tdrs9_satrec, 0.0, tdrs9_ro, tdrs9_vo);
//printf("SGP4 at t = 0 to get initial state vector complete...\n");

//Call propogator to get initial state vector value
sgp4(whichconst, tdrs10_satrec, 0.0, tdrs10_ro, tdrs10_vo);
//printf("SGP4 at t = 0 to get initial state vector complete...\n");



juleanDate = satrec.jdsatepoch;
tdrs8_juleanDate = tdrs8_satrec.jdsatepoch;
tdrs9_juleanDate = tdrs9_satrec.jdsatepoch;
tdrs10_juleanDate = tdrs10_satrec.jdsatepoch;

 int i = 0;
std::time_t end = std::time(NULL) + (95 * 60.0);
while (std::time(NULL) <= end) {



    //RUN SGP4 AND COORDINATE TRANSFORMATION COMPUTATIONS
    juleandateCurrent = getJulianFromUnix(time(NULL));

    tsince = (juleandateCurrent - juleanDate) * 24.0 * 60.0;
    sgp4(whichconst, satrec, tsince, ro, vo);
    teme2ecef(ro, vo, juleandateCurrent, recef, vecef);
```

```
ijk2ll(recef, latlongh);
rv2azel(ro, vo, siteLatRad, siteLonRad, siteAlt, juleandateCurrent,
    razel, razelrates);


site(siteLatRad, siteLonRad, siteAlt, ground_ro, ground_vo);


//CHECK FOR ERRORS
if (satrec.error > 0) {
    printf("# *** error: t:= %f *** code = %3d\n", satrec.t,
        satrec.error);
} else {
    azimuth = razel[1]*180 / pi; // Azimuth * (180/pi)

    elevation = razel[2]*180 / pi; // Elevation * (180/pi)


    if (i == 0) {


        for (int j = 0; j < 3; j++) {
            prevrecef[j] = recef[j];
            prevground_ro[j] = ground_ro[j];
        }
        myfile << i << " " << recef[0] << " " << recef[1] << " "
               << recef[2] << " " << vecef[0] << " " << vecef[1] <<
```

```
                " "
            << vecef[2] << " " << latlongh[0] * 180 / pi << " "
            << latlongh[1] * 180 / pi << " " << latlongh[2] << "
                "
            << razel[0] << " " << razel[1] * 180 / pi << " "
            << razel[2] * 180 / pi << "\n";
        issFile << i << "," << recef[0] << "," << recef[1] << ","
            << recef[2] << "," << vecef[0] << "," << vecef[1] <<
                ","
            << vecef[2] << "," << latlongh[0] * 180 / pi << ","
            << latlongh[1] * 180 / pi << "," << latlongh[2] <<
                ","
            << razel[0] << "," << razel[1] * 180 / pi << ","
            << razel[2] * 180 / pi << "\n";
        issWithJd << satrec.t << "," << recef[0] << "," << recef[1]
            << ","
            << recef[2] << "," << vecef[0] << "," << vecef[1] <<
                ","
            << vecef[2] << "," << latlongh[0] * 180 / pi << ","
            << latlongh[1] * 180 / pi << "," << latlongh[2] <<
                ","
            << razel[0] << "," << razel[1] * 180 / pi << ","
            << razel[2] * 180 / pi << "\n";


        geoLoc << i << " " << ground_ro[0] << " " << ground_ro[1] <<
            " " << ground_ro[2] << "\n";
```

```
        } else {
            if (isChanged(recef, prevrecef)) {
                myfile << i << " " << recef[0] << " " << recef[1] << " "
                       << recef[2] << " " << vecef[0] << " " << vecef[1]
                          << " "
                       << vecef[2] << " " << latlongh[0] * 180 / pi << "
                          "
                       << latlongh[1] * 180 / pi << " " << latlongh[2]
                          << " "
                       << razel[0] << " " << razel[1] * 180 / pi << " "
                       << razel[2] * 180 / pi << "\n";
                issFile << i << "," << recef[0] << "," << recef[1] << ","
                   << recef[2] << "," << vecef[0] << "," << vecef[1] <<
                      ","
                   << vecef[2] << "," << latlongh[0] * 180 / pi << ","
                   << latlongh[1] * 180 / pi << "," << latlongh[2] <<
                      ","
                   << razel[0] << "," << razel[1] * 180 / pi << ","
                   << razel[2] * 180 / pi << "\n";
            issWithJd << satrec.t << "," << recef[0] << "," << recef[1]
               << ","
                   << recef[2] << "," << vecef[0] << "," << vecef[1] <<
                      ","
                   << vecef[2] << "," << latlongh[0] * 180 / pi << ","
                   << latlongh[1] * 180 / pi << "," << latlongh[2] <<
                      ","
                   << razel[0] << "," << razel[1] * 180 / pi << ","
```

```
                << razel[2] * 180 / pi << "\n";
            for (int j = 0; j < 3; j++) {
                prevrecef[j] = recef[j];
                prevground_ro[j] = ground_ro[j];
            }
        } else {
            printf("ISS loc not changed \n");
        }

        if (isChanged(ground_ro, prevground_ro)) {
            geoLoc << i << " " << ground_ro[0] << " " <<
                ground_ro[1] << " " << ground_ro[2] << "\n";
        }
    }

}

//tdrs-8 SGP4 AND COORDINATE TRANSFORMATION COMPUTATIONS

tsince = (juleandateCurrent - tdrs8_juleanDate) * 24.0 * 60.0;
sgp4(whichconst, tdrs8_satrec, tsince, tdrs8_ro, tdrs8_vo);
teme2ecef(tdrs8_ro, tdrs8_vo, juleandateCurrent, tdrs8_recef,
    tdrs8_vecef);
ijk2ll(tdrs8_recef, tdrs8_latlongh);
rv2azel(tdrs8_ro, tdrs8_vo, siteLatRad, siteLonRad, siteAlt,
    juleandateCurrent, tdrs8_razel, tdrs8_razelrates);
//CHECK FOR ERRORS
```

```
if (satrec.error > 0) {
    printf("# *** error: t:= %f *** code = %3d\n", tdrs8_satrec.t,
        tdrs8_satrec.error);
} else {



    if (i == 0) {
        for (int j = 0; j < 3; j++) {
            prevtdrs8_recef[j] = tdrs8_recef[j];
        }
        tdrs8 << i << " " << tdrs8_recef[0] << " " << tdrs8_recef[1]
            << " "
                << tdrs8_recef[2] << " " << tdrs8_vecef[0] << " " <<
                    tdrs8_vecef[1] << " "
                << tdrs8_vecef[2] << " " << tdrs8_latlongh[0] * 180
                    / pi << " "
                << tdrs8_latlongh[1] * 180 / pi << " " <<
                    tdrs8_latlongh[2] << " "
                << tdrs8_razel[0] << " " << tdrs8_razel[1] * 180 /
                    pi << " "
                << tdrs8_razel[2] * 180 / pi << "\n";
        tdrs8C << i << " " << tdrs8_recef[0] << " " <<
            tdrs8_recef[1] << " "
                << tdrs8_recef[2] << " " << tdrs8_vecef[0] << " " <<
                    tdrs8_vecef[1] << " "
                << tdrs8_vecef[2] << " " << tdrs8_latlongh[0] * 180
                    / pi << " "
```

```
                << tdrs8_latlongh[1] * 180 / pi << " " <<
                    tdrs8_latlongh[2] << " "
                << tdrs8_razel[0] << " " << tdrs8_razel[1] * 180 /
                    pi << " "
                << tdrs8_razel[2] * 180 / pi << "\n";



        } else {
            if (isChanged(tdrs8_recef, prevtdrs8_recef)) {
                tdrs8 << i << " " << tdrs8_recef[0] << " " <<
                    tdrs8_recef[1] << " "
                        << tdrs8_recef[2] << " " << tdrs8_vecef[0] << " "
                            << tdrs8_vecef[1] << " "
                        << tdrs8_vecef[2] << " " << tdrs8_latlongh[0] *
                            180 / pi << " "
                        << tdrs8_latlongh[1] * 180 / pi << " " <<
                            tdrs8_latlongh[2] << " "
                        << tdrs8_razel[0] << " " << tdrs8_razel[1] * 180
                            / pi << " "
                        << tdrs8_razel[2] * 180 / pi << "\n";
                tdrs8C << i << " " << tdrs8_recef[0] << " " <<
                    tdrs8_recef[1] << " "
                        << tdrs8_recef[2] << " " << tdrs8_vecef[0] << " "
                            << tdrs8_vecef[1] << " "
                        << tdrs8_vecef[2] << " " << tdrs8_latlongh[0] *
                            180 / pi << " "
                        << tdrs8_latlongh[1] * 180 / pi << " " <<
```

```
                    tdrs8_latlongh[2] << " "
                  << tdrs8_razel[0] << " " << tdrs8_razel[1] * 180
                    / pi << " "
                  << tdrs8_razel[2] * 180 / pi << "\n";
            for (int j = 0; j < 3; j++) {
                prevtdrs8_recef[j] = tdrs8_recef[j];
            }
        } else {
            printf("tdrs8 loc not changed \n");
        }

    }


}

//tdrs-9 SGP4 AND COORDINATE TRANSFORMATION COMPUTATIONS

tsince = (juleandateCurrent - tdrs9_juleanDate) * 24.0 * 60.0;
sgp4(whichconst, tdrs9_satrec, tsince, tdrs9_ro, tdrs9_vo);
teme2ecef(tdrs9_ro, tdrs9_vo, juleandateCurrent, tdrs9_recef,
    tdrs9_vecef);
ijk2ll(tdrs9_recef, tdrs9_latlongh);
rv2azel(tdrs9_ro, tdrs9_vo, siteLatRad, siteLonRad, siteAlt,
    juleandateCurrent, tdrs9_razel, tdrs9_razelrates);
//CHECK FOR ERRORS
if (satrec.error > 0) {
```

```
            printf("# *** error: t:= %f *** code = %3d\n", tdrs9_satrec.t,
                tdrs9_satrec.error);
        } else {

            if (i == 0) {

                for (int j = 0; j < 3; j++) {
                    prevtdrs9_recef[j] = tdrs9_recef[j];
                }
                tdrs9 << i << " " << tdrs9_recef[0] << " " << tdrs9_recef[1]
                    << " "
                        << tdrs9_recef[2] << " " << tdrs9_vecef[0] << " " <<
                            tdrs9_vecef[1] << " "
                        << tdrs9_vecef[2] << " " << tdrs9_latlongh[0] * 180
                            / pi << " "
                        << tdrs9_latlongh[1] * 180 / pi << " " <<
                            tdrs9_latlongh[2] << " "
                        << tdrs9_razel[0] << " " << tdrs9_razel[1] * 180 /
                            pi << " "
                        << tdrs9_razel[2] * 180 / pi << "\n";
                tdrs9C << i << " " << tdrs9_recef[0] << " " <<
                    tdrs9_recef[1] << " "
                        << tdrs9_recef[2] << " " << tdrs9_vecef[0] << " " <<
                            tdrs9_vecef[1] << " "
                        << tdrs9_vecef[2] << " " << tdrs9_latlongh[0] * 180
                            / pi << " "
                        << tdrs9_latlongh[1] * 180 / pi << " " <<
```

```
                    tdrs9_latlongh[2] << " "
                << tdrs9_razel[0] << " " << tdrs9_razel[1] * 180 /
                    pi << " "
                << tdrs9_razel[2] * 180 / pi << "\n";



        } else {
            if (isChanged(tdrs9_recef, prevtdrs9_recef)) {
                tdrs9 << i << " " << tdrs9_recef[0] << " " <<
                    tdrs9_recef[1] << " "
                        << tdrs9_recef[2] << " " << tdrs9_vecef[0] << " "
                            << tdrs9_vecef[1] << " "
                        << tdrs9_vecef[2] << " " << tdrs9_latlongh[0] *
                            180 / pi << " "
                        << tdrs9_latlongh[1] * 180 / pi << " " <<
                            tdrs9_latlongh[2] << " "
                        << tdrs9_razel[0] << " " << tdrs9_razel[1] * 180
                            / pi << " "
                        << tdrs9_razel[2] * 180 / pi << "\n";
                tdrs9C << i << " " << tdrs9_recef[0] << " " <<
                    tdrs9_recef[1] << " "
                        << tdrs9_recef[2] << " " << tdrs9_vecef[0] << " "
                            << tdrs9_vecef[1] << " "
                        << tdrs9_vecef[2] << " " << tdrs9_latlongh[0] *
                            180 / pi << " "
                        << tdrs9_latlongh[1] * 180 / pi << " " <<
                            tdrs9_latlongh[2] << " "
```

```
                    << tdrs9_razel[0] << " " << tdrs9_razel[1] * 180
                        / pi << " "
                    << tdrs9_razel[2] * 180 / pi << "\n";


            for (int j = 0; j < 3; j++) {
                prevtdrs9_recef[j] = tdrs9_recef[j];
            }
        } else {
            printf("tdrs8 loc not changed \n");
        }

    }



}

//tdrs-10 SGP4 AND COORDINATE TRANSFORMATION COMPUTATIONS

tsince = (juleandateCurrent - tdrs10_juleanDate) * 24.0 * 60.0;
sgp4(whichconst, tdrs10_satrec, tsince, tdrs10_ro, tdrs10_vo);
teme2ecef(tdrs10_ro, tdrs10_vo, juleandateCurrent, tdrs10_recef,
    tdrs10_vecef);
ijk2ll(tdrs10_recef, tdrs10_latlongh);
rv2azel(tdrs10_ro, tdrs10_vo, siteLatRad, siteLonRad, siteAlt,
    juleandateCurrent, tdrs10_razel, tdrs10_razelrates);
```

```
//CHECK FOR ERRORS
if (satrec.error > 0) {
    printf("# *** error: t:= %f *** code = %3d\n", tdrs10_satrec.t,
        tdrs10_satrec.error);
} else {



    if (i == 0) {
        for (int j = 0; j < 3; j++) {
            prevtdrs10_recef[j] = tdrs10_recef[j];
        }


        tdrs10 << i << " " << tdrs10_recef[0] << " " <<
            tdrs10_recef[1] << " "
                << tdrs10_recef[2] << " " << tdrs10_vecef[0] << " "
                    << tdrs10_vecef[1] << " "
                << tdrs10_vecef[2] << " " << tdrs10_latlongh[0] *
                    180 / pi << " "
                << tdrs10_latlongh[1] * 180 / pi << " " <<
                    tdrs10_latlongh[2] << " "
                << tdrs10_razel[0] << " " << tdrs10_razel[1] * 180 /
                    pi << " "
                << tdrs10_razel[2] * 180 / pi << "\n";
        tdrs10C << i << " " << tdrs10_recef[0] << " " <<
            tdrs10_recef[1] << " "
                << tdrs10_recef[2] << " " << tdrs10_vecef[0] << " "
                    << tdrs10_vecef[1] << " "
```

```
                << tdrs10_vecef[2] << " " << tdrs10_latlongh[0] *
                    180 / pi << " "
                << tdrs10_latlongh[1] * 180 / pi << " " <<
                    tdrs10_latlongh[2] << " "
                << tdrs10_razel[0] << " " << tdrs10_razel[1] * 180 /
                    pi << " "
                << tdrs10_razel[2] * 180 / pi << "\n";



        } else {
            if (isChanged(tdrs10_recef, prevtdrs10_recef)) {
                tdrs10 << i << " " << tdrs10_recef[0] << " " <<
                    tdrs10_recef[1] << " "
                        << tdrs10_recef[2] << " " << tdrs10_vecef[0] << "
                            " << tdrs10_vecef[1] << " "
                        << tdrs10_vecef[2] << " " << tdrs10_latlongh[0] *
                            180 / pi << " "
                        << tdrs10_latlongh[1] * 180 / pi << " " <<
                            tdrs10_latlongh[2] << " "
                        << tdrs10_razel[0] << " " << tdrs10_razel[1] *
                            180 / pi << " "
                        << tdrs10_razel[2] * 180 / pi << "\n";
                tdrs10C << i << " " << tdrs10_recef[0] << " " <<
                    tdrs10_recef[1] << " "
                        << tdrs10_recef[2] << " " << tdrs10_vecef[0] << "
                            " << tdrs10_vecef[1] << " "
                        << tdrs10_vecef[2] << " " << tdrs10_latlongh[0] *
```

```
                        180 / pi << " "
                    << tdrs10_latlongh[1] * 180 / pi << " " <<
                        tdrs10_latlongh[2] << " "
                    << tdrs10_razel[0] << " " << tdrs10_razel[1] *
                        180 / pi << " "
                    << tdrs10_razel[2] * 180 / pi << "\n";
                for (int j = 0; j < 3; j++) {
                    prevtdrs10_recef[j] = tdrs10_recef[j];
                }
            } else {
                printf("tdrs8 loc not changed \n");
            }

        }
    }
    i++;
    sleep(1);
} //indefinite loop
```

Appendix C

`std::string` to `std::vector<>` Conversion

The following code is used to convert a comma separated double values stored in a `std::string` to `std::vector<>`.

```
std::string str = "1.2,2.3,3.5,4.2,5.3,6.2";
std::vector<double> vect;
std::stringstream ss(str);
double i;
double arr[6];
int j = 0;
while (ss >> i) {
    vect.push_back(i);
    arr[j++] = i;
    if (ss.peek() == ',') {
        ss.ignore();
    }
}
```