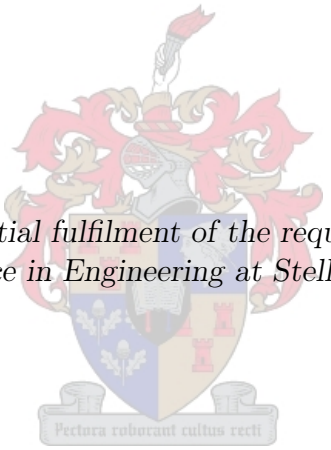
Development of a Satellite Communications Software System and Scheduling Strategy

by

John Sebastian Gilmore



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University

Supervisor: Dr. Riaan Wolhuter
Department of Electrical and Electronic Engineering

March 2010

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2010

Abstract

Stellenbosch University and the Katholieke Universiteit Leuven has a joint undertaking to develop a satellite communications payload. The goals of the project are: to undertake research and expand knowledge in the area of dynamically configurable antenna beam forming, to prove the viability of this research for space purposes and to demonstrate the feasibility of the development in a practical application.

The practical application is low Earth orbit satellite communication system for applications in remote monitoring. Sensor data will be uploaded to the satellite, stored and forwarded to a central processing ground station as the satellite passes over these ground stations. The system will utilise many low-cost ground sensor stations to collect data and distribute it to high-end ground stations for processing.

Applications of remote monitoring systems are maritime- and climate change monitoring- and tracking. Climate change monitoring allows inter alia, for the monitoring of the effects and causes of global warming.

The Katholieke Universiteit Leuven is developing a steerable antenna to be mounted on the satellite. Stellenbosch University is developing the communications payload to steer and use the antenna. The development of the communications protocol stack is part of the project. The focus of this work is to implement the application layer protocol, which handles all file level communications and also implements the communications strategy.

The application layer protocol is called the *Satellite Communications Software System* (SCSS). It handles all high level requests from ground stations, including requests to store data, download data, download log files and upload configuration information. The design is based on a client-server model, with a *Station Server* and *Station Handler*. The Station Server schedules ground stations for communication and creates a Station Handler for each ground station to handle all ground station requests. During the design, all file formats were defined for efficient ground station-satellite communications and system administration. All valid ground station requests and handler responses were also defined.

It was also found that the system may be made more efficient by scheduling ground stations for communications, rather than polling each ground station until one responds. To be able to schedule ground station communications, the times when ground stations will come into view of the satellite have to be predicted. This is done by calculating the positions of the Satellite and ground stations as functions of time. A simple orbit propagator was developed to predict the satellite distance and to ease testing and integration with the communications system. The times when a ground station will be within range of the satellite were then predicted and a scheduling algorithm developed to minimise the number of ground stations not able

to communicate.

All systems were implemented and tested. The SCSS executing on the Satellite was developed and tested on the satellite on-board computer. Embedded implementations possess strict resource limitations, which were taken into account during the development process. The SCSS is a multi-threaded system that makes use of thread cancellation to improve responsiveness.

Samevatting

Die Universiteit van Stellenbosch ontwerp tans 'n satelliet kommunikasieloonvrag in samewerking met die Katolieke Universiteit van Leuven. Die doel van die projek is om navorsing te doen oor die lewensvatbaarheid van dinamies verstelbare antenna bundelvorming vir ruimte toepassings, asook om die haalbaarheid van hierdie navorsing in die praktyk te demonstreer.

Die praktiese toepassing is 'n satellietkommunikasiestelsel vir afstandsmonitering, wat in 'n Lae-Aarde wentelbaan verkeer. Soos die satelliet in sy wentelbaan beweeg, sal sensor data na die satelliet toe gestuur, gestoor en weer aangestuur word. Die stelsel gebruik goedkoop sensorgrondstasies om data te versamel en aan te stuur na kragtiger grondstasies vir verwerking.

Afstandsmoniteringstelsels kan gebruik word om klimaatsverandering, sowel as die posisie van skepe en voertuie, te monitor. Deur oa. klimaatsveranderinge te dokumenteer, kan gevolge en oorsake van globale verhitting gemonitor word.

Die Katholieke Universiteit van Leuven is verantwoordelik vir die ontwerp en vervaardiging van die satelliet antenna, terwyl die Universiteit van Stellenbosch verantwoordelik is vir die ontwerp en bou van die kommunikasie loonvrag. 'n Gedeelte van hierdie ontwikkeling sluit die ontwerp en implementasie van al die protokolle van die kommunikasieprotokolstapel in. Dit fokus op die toepassingsvlak protokol van die protokolstapel, wat alle leêrvlak kommunikasie hanteer en die kommunikasiestrategie implementeer.

Die toepassingsvlaksagteware word die Satellietkommunikasie sagtewarestelsel (SKSS) genoem. Die SKSS is daarvoor verantwoordelik om alle navrae vanaf grondstasies te hanteer. Hierdie navrae sluit die oplaai en stoor van data, die aflaai van data, die aflaai van logs en die oplaai van konfigurasie inligting in. Die ontwerp is op die standaard kliënt-bediener model gebasseer, met 'n *stasiebediener* en 'n *stasiehanteerder*. Die stasiebediener skeduleer die tye wanneer grondstasies toegelaat sal word om te kommunikeer en skep stasiehanteerders om alle navrae vanaf die stasies te hanteer. Gedurende die ontwerp is alle leêrformate gedefinieer om doeltreffende adminstrasie van die stelsel, asook kommunikasie tussen grondstasies en die satelliet te ondersteun. Alle geldige boodskappe tussen die satelliet en grondstasies is ook gedefnieer.

Daar is gevind dat die doeltreffendheid van die stelsel verhoog kan word deur die grondstasies wat wil kommunikeer te skeduleer, eerder as om alle stasies te pols totdat een reageer. Om so 'n skedule op te stel, moet die tye wanneer grondstasies binne bereik van die satelliet gaan wees voorspel word. Hierdie voorspelling is gedoen deur die posisies van die satelliet en die grondstasies as funksies van tyd te voorspel. 'n Eenvoudige satelliet posisievoorspeller is ontwikkel om toetsing en integrasie met die

SKSS te vergemaklik. 'n Skeduleringsalgoritme is toe ontwikkel om die hoeveelheid grondstasies wat nie toegelaat word om te kommunikeer nie, te minimeer.

Alle stelsels is geimplementeer en getoets. Die SKSS, wat op die satelliet loop, is ontwikkel en getoets op die satelliet se aanboord rekenaar. Die feit dat ingebedde stelsels oor baie min hulpbronne beskik, is in aanmerking geneem gedurende die ontwikkeling en implementasie van die SKSS. Angesien die SKSS 'n multidraadverwerkingsstelsel is, word daar van draadkansellasie gebruik gemaak om die stelsel se reaksietyd te verbeter.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- the Holy Father, for keeping me and blessing me with so much;
- my study leader, Dr Riaan Wolhuter, for his continued guidance and support;
- my fiancée, Jacki van der Merwe, for her lasting love, support and understanding;
- Francois Olivier and Shaun Lodder, for their valuable input during the late nights in the lab;
- Dr Gert-Jan van Rooyen for his valuable feedback on the SCSS design;
- Ewald van der Westhuizen for managing the Leuven project and for providing technical assistance;
- Kobus Botha for always being ready to assist with technical issues;
- Japie Engelbrecht, for helping me better understand satellite communication systems;
- the Telkom Centre of Excellence and Stellenbosch University, for their financial aid;
- my parents, John and Coreen Gilmore, for making me the man I am today and making my studies possible;
- the QNX support team, for their prompt and knowledgeable assistance with QNX related implementation issues;
- James Clark, for writing the Expat XML parser library;
- Jean-Loup Gailly and Mark Adler, for writing the zlib compression library.

Dedications

In memory of my mother, Anita Gilmore, and my grandparents: Herman Kotze, Kotie Kotze and Hettie Gilmore. I hope I've made you proud.

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

Satellite orbit characteristics

R_E	Mean Earth radius
h	Satellite altitude
p	Length of one satellite orbit rotation
v	Satellite velocity
T_S	Satellite period
c	Speed of light in a vacuum
g	Nominal Earth gravitational acceleration at sea level
s	Satellite footprint velocity
l	Satellite arc length
t_x	Channel delay time
RTT	Round-trip-time
T_C	Communications time window length
d	Satellite-ground station communications range

Scheduling theory

α	Machine environment
β	Job characteristic
γ	Optimality criteria
Pm	m parallel machines
J_i	Job number i
r_i	Release time of J_i
d_i	Deadline of J_i
C_i	Completion time of J_i
$f_i()$	Cost function of J_i
GS_{dropped}	Total number of viable unscheduled ground stations

Ground station communications

t	Time
g_i	Ground station i
t_{is}	Communications start time of g_i

t_{ie}	Communications end time of g_i
τ_i	Required communications time of g_i

Satellite position prediction

$\mathbf{S}$	Satellite position vector
K	Position vector length
t_{end}	Prediction end time
t_{step}	Prediction step time
$\mathbf{s}_0$	Initial satellite position
x, y, z	Coordinates in $\mathbb{R}^3$
$\Delta\beta$	Satellite step angle
$\mathbf{S}_0$	Initial uninclined satellite position vector
$\mathbf{Q}$	Satellite orbit rotation matrix
$\mathbf{L}$	Satellite inclination rotation matrix
$\mathbf{S}_i$	Inclined satellite position vector
$\mathbf{H}$	Satellite longitudinal rotation matrix
ω	Longitudinal angle

Ground station position prediction

$\mathbf{G}$	Ground station position vector
$\mathbf{R}$	Ground station rotation matrix
$\Delta\sigma$	Ground station step angle
T_G	Period of one Earth rotation (1 day)
ϑ	Latitude
φ	Longitude
h_s	Height above sea level
R_{trans}	Transverse radius of curvature
ε^2	Eccentricity of the Earth ellipsoid
a	Semi-major axis of the Earth ellipsoid
b	Semi-minor axis of the Earth ellipsoid

Visibility and angle prediction

$\mathbf{d}$	Satellite-ground station distance vector
γ_{sat}	Satellite perspective angle
γ_{gs}	Ground station perspective angle
ϕ	Vertical satellite-ground station angle
θ	Horizontal satellite-ground station angle
$\vec{N}$	North vector
$\vec{N}'$	Projected North vector

$\vec{S}'$	Projected Satellite position vector
$\hat{x}$	Unit vector in the x direction
$\hat{y}$	Unit vector in the y direction
$\hat{z}$	Unit vector in the z direction

System design

t_{start}	Scheduled ground station start time
t_{stop}	Scheduled ground station stop time

Subscripts

min	Minimum
max	Maximum
cur	Current
k	The kth vector element
x	The x component of a vector
y	The y component of a vector
z	The z component of a vector

Abbreviations

ACK	Acknowledgement
API	Application Programming Interface
ARQ	Automatic Repeat-request
AWS	Amazon Web Services
CCSDS	Consultative Committee for Space Data Systems
CPU	Central Processing Unit
CSV	Comma-separated Values
CTW	Communications Time Window
DSP	Digital Signal Processing
ECSS	European Corporation for Space Standardisation
FIX	Financial Information Exchange
FPGA	Field-programmable Gate Array
FTP	File Transfer Protocol
GEO	Geostationary Earth Orbit
GPS	Global Positioning System
GPX	GPS Exchange Format
GS	Ground Station
GSL	Ground Station Link
ICMP	Internet Control Message Protocol
ID	Identifier

IDE	Integrated Development Environment
IP	Internet Protocol
IPC	Inter-process Communications
JSON	JavaScript Object Notation
KU	Katholieke Universiteit
LEO	Low Earth Orbit
LNE	Least Number of Exclusions
MIME	Multipurpose Internet Mail Extensions
NASA	National Aeronautics and Space Administration
OBC	On-board Computer
OS	Operating System
OSI	Open System Interconnection
PC	Personal Computer
POSIX	Portable Operating System Interface
QDE	QNX Development Environment
QoS	Quality of Service
QPSK	Quadrature Phase-shift Keying
RCT	Required communications time
RF	Radio Frequency
ROI	Return on Investment
RSS	Really Simple Syndication
RTC	Real-time clock
RTT	Round-trip Time
SCSS	Satellite Communications Software System
SCTF	Shortest Communications Time First
SGP	Simplified General Perturbations
SH	SuperH
SMS	Short Message Service
SSV	Space-separated Values
STK	Satellite Tool Kit
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TLE	Two-line Element
TM	Telemetry
UML	Unified Modelling Language
USA	United States of America
WGS	World Geodetic System
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

XSD	XML Standard Definition
YAJL	Yet Another JSON Library

Chapter 1

Introduction

1.1 Background

The IS-HSII project is a joint project undertaken between the Katholieke Universiteit (KU) Leuven and Stellenbosch University for the development of a satellite borne, electronically beam steerable antenna array. The ESAT-TELEMIC division of the Department of Electrical Engineering, of the KU Leuven, is currently developing techniques for electronic antenna beam steering in space. The purpose of the development is threefold:

- To undertake research and expand knowledge in the area of dynamically configurable antenna beam forming as an academic objective
- To prove the viability of this research for space purposes
- To demonstrate the feasibility of the development in a practical application, where ground-based environmental sensor data would be uploaded to a satellite carrying the steerable antenna array

The project is jointly undertaken between the KUL and the Digital Signal Processing (DSP)-Telecommunications group of the Department of Electrical and Electronic Engineering of Stellenbosch University. The antenna and associated components are developed in Leuven, while the satellite platform, ground station and ground-satellite communications link, are developed in Stellenbosch. The eventual objective is to fly the system on the next South African satellite.

Part of the ground station-satellite communications link is the design of the protocol stack of the communications sub-system and the implementation of all layers thereof, to enable full store-and-forward functionality. A team of people were appointed to implement the communications sub-system.

The implementation consists of both software and hardware development. Most of the hardware was developed by Sunspace for the SumbandilaSat project. The on-board computer used for the project is the same computer as on SumbandilaSat. Some experience could be transferred from the SumbandilaSat project to the Leuven project, but there were also some major differences. A key difference is the Sumbandila satellite is a half-duplex system, whereas the Leuven system is full-duplex. This allows for more freedom and functionality in the on-board software design.

One component of the communications system is the Satellite Communications Software System (SCSS) that resides in the top layer of the protocol stack, in the application layer. The SCSS controls communication times and durations with ground stations, stores files received from ground stations and implements a store-and-forward system to deliver data to destination ground stations. The design and implementation of the SCSS is the focus of this work.

1.2 Objectives and contributions

The objectives of this study was the design and implementation of the SCSS executing on the satellite on-board computer, while taking into account the resource limitations of the hardware. The design includes defining all message formats to be used for communications between the SCSS and ground stations. The purpose of the SCSS is to coordinate all high level communication operations of the satellite.

Functionality expected from the SCSS are:

- Initiate connections with ground stations
- Allow ground stations to communicate with the satellite on a file level
- Allow ground stations to upload and download data
- Allow ground stations to send data to other ground stations
- Manage the communication times and durations to ensure that all ground stations receive equal service.
- Provide a means to store and retrieve uploaded data on the satellite
- Manage the steering of the satellite antenna, to point to the currently communicating ground station.

The SCSS was successfully developed and implemented on the satellite hardware. The SCSS consists of a station scheduler that manages the allowed communication times of all ground stations and the station handlers, which directly handle all requests from ground stations. A file store is also implemented to store all ground station messages, configuration data and schedules. Testing was performed to ensure the SCSS functions as designed and within the required resource constraints.

A substantial amount of time was spent on developing an effective communications strategy. The strategy manages communication times with ground stations, including both allowed start times and communications duration. It was found that the volumetric throughput of the system could be maximised, by actively controlling ground station communication times. The active control is performed by the satellite and involves a schedule calculated off-line, making use of satellite and ground station position predictions.

Predicting the satellite and ground station positions, allows the satellite to be aware of the link quality of every satellite pass. This allows a scheduler to select a ground station, having a high-quality link for the specific pass. In this way, every

ground station is allowed a high quality pass when it communicates, while the strategy also equalises all ground station communication times. The calculated schedule drives the SCSS.

The link predictions also allows for the analysis of the space mission, to calculate average and total communication times and thereby the overall system communications capacity.

Angle predictions were also performed to allow for a directed antenna on the ground stations, instead of a basic omni-directional design. Angle predictions allow for the average angle to be predicted, which the satellite and ground station will communicate in, most of the time. This can be used to achieve an acceptable link margin, in applications where an omni-directional antenna produces an unacceptable link margin.

1.3 Applications

The satellite system under discussion is being designed for remote monitoring applications, as stated in Section 1.1 and depicted in Figure 3.3. A remote monitoring system exists of two types of ground stations, i.e. ground stations collecting data from sensor networks and those aggregating the collected data for further processing, or pass the data on to a server in the Internet for processing. The ground stations collecting the sensor data, are the data sources in the communications system and the ground stations aggregating the data, are the data sinks in the system.

Ground stations collecting sensor data are placed in remote areas, with no Internet or cellular connection. There is, therefore, no way for data collected from these sensors to be processed without manual data collection techniques. This manual process of data collection is very labour intensive and allows little time for data processing during collection, or requires a large team to collect the data.

A remote monitoring system collects the data from rural ground stations and only requires personnel to process the data at the data processing centre where the data are sent. Data are uploaded from rural ground stations and downloaded to the aggregator ground station where all data are stored in a database, or sent to a server for processing. Data mining can then be performed from a central location and different data sets from different sensor ground stations can easily be correlated and compared.

Applications of remote monitoring systems are tracking, and climate and maritime monitoring. Climate monitoring systems monitor meteorological elements such as temperature, humidity, rainfall, wind and atmospheric pressure in a given region. In current times, these systems are of great importance, because they enable the monitoring of the effects of global warming. It allows tracking of the global climate at near to real-time. Large data sets can then be processed after being delivered to a central processing facility with more powerful capabilities than what would be individually available at every monitoring site.

The second example is that of tracking. Environmental research is done at the University of Stellenbosch to track Leopard movements with tracking collars. Another application of the satellite system is to have ground sensor stations monitoring the positions of the tracking collars. Multiple ground stations can then be set up to

monitor leopards moving throughout their habitat. These stations will be set up in rural areas in the mountains and will have no form of communications, except the satellite system. Manual data retrieval is also difficult in these areas as researchers have to track the leopards with tracking equipment in the field. This can be a tedious as well as dangerous expedition.

The final example is one where a satellite based tracking antenna can have a significant impact. This is in maritime monitoring operations. Shipping companies monitor their ships to enable them to gauge their times to arrival as well as other operational parameters. Currently, these ships have systems to communicate with a satellite and these communications are made possible with the use of an antenna enabling contact with the satellite. It is important that the antenna be pointed at the satellite at all times when communicating. The antenna must, therefore, possess stabilisers that maintain the correct pointing direction, even with the destabilising effect of ocean waves.

Antennas currently used on ships are very expensive, as they employ sophisticated stabilisation techniques [1]. With the antenna mounted on the satellite, there is no need for stabilisation on the ship. The satellite is always able to point towards the ship and the movement of the ship will have no significant effect, provided that the ship-mounted antenna is reasonably omni-directional.

For the design of the satellite communications system, the intended applications should at all times be kept in mind. The characteristics of a remote monitoring systems are low data volumes, high number of data sources, intermittent data production, text-based measurement data that are highly compressible and data not requiring immediate processing or transmission.

The low data volumes stem from the nature of the data. The data are measurement data, which would consist of numbers and text. No video or audio data are transmitted. Since there may be many sensor networks and rural ground stations throughout the area being monitored, many data sources using the satellite exist in the system. The sensor system will, however, not be constantly producing data. Measurements are taken at certain times during the day and that data must then be uploaded for aggregation. The measurement data are also considered to be non-real-time off-line data. No immediate data processing is required and a delay in transmission will not adversely affect the system.

1.4 Overview of this work

Chapter 2 provides the required background information on satellite communications and how this applies to LEO satellite systems and specifically to the LEO satellite system being designed. The chapter takes a top-down approach to provide perspective on where the SCSS fits in. It concludes by describing on-board processing and satellite autonomy, which forms part of the focus of this work.

After satellite systems in general have been described, Chapter 3 presents an overview of specifically the satellite being designed. It details the orbit characteristics to show what little time is available for LEO communications. It also describes the basics of how the satellite and the ground station will communicate and presents an overview of the protocol stack developed. The chapter then moves on to describe the

hardware and interfaces of the system as well as to give some background information on the architecture of the operating system used on the on-board computer. The chapter also describes the KU Leuven steerable antenna, around which the project was designed and discusses how the antenna influenced design.

Chapter 4 describes a novel method of how the first function of the SCSS is implemented, namely, acquiring the satellite link. The chapter introduces the link acquisition technique and shows how links are acquired by predicting the positions of the satellite and ground stations in time and using these predictions to calculate a schedule for the system. The chapter also debates the merits of the prediction technique and how this improves the volumetric throughput of the communications system.

Chapter 5 presents the design of the SCSS. Initially, a functional analysis is performed and then the high-level design is described. Two entities are introduced, the station server and station handler and the design of each system with flow diagrams is discussed in detail. Mechanisms used to ensure correct functionality within the limited resource environment are also presented. Message formats are also discussed, along with the choice of markup language. The different messages used by the system are also presented along with the structure of the file store. Another important part of the SCSS is logging and the implementation of this is also discussed.

Chapter 6 discusses some implementation specific details of the SCSS. The communications system is discussed on a program level and the other supporting software and scripts are also discussed. The unique challenges faced when developing for an embedded system are presented, which is followed by testing and performance descriptions. All tests performed on the system are described and the importance of each test is explained. The chapter concludes by illustrating memory and CPU performance figures achieved. These figures illustrate the low resource usage of the SCSS.

Chapter 7 concludes the work with a discussion on contributions made and recommendations regarding the path ahead as well as what sections require further investigation.

Chapter 2

Study of satellite communication techniques

2.1 Introduction

This chapter presents the current state of the art of the various topics covered in this work. Key concepts are also presented, on which the rest of the work is built. An understanding of all key concepts is required, to enable an understanding of the content and the focus of the work.

The chapter is organised to present a top-down description of the satellite system, as well as to present alternative methods. The satellite is first presented from an orbit perspective and both GEO and LEO orbits are discussed. LEO communications and tracking methods are then presented. Next, the difference between little LEOs and big LEOs are discussed and examples presented of each system.

This leads the discussion on to how communication links are established in little LEO systems and what methods are currently in use by the little LEO examples as discussed. Standards dealing with LEO link acquisition are also highlighted. After the discussion of communications, the higher level concepts of on-board processing and satellite autonomy are introduced.

Section 2.2 compares geostationary and low Earth orbits. Section 2.3 describes techniques employed to enable LEO communications in the physical layer and to allow a ground station to determine the range of a satellite with which it wishes to communicate. Section 2.4 discusses big and little LEOs and both the commercial and technical challenges they face. Section 2.5 describes link acquisition techniques used by current satellite communication systems and standards. Section 2.6 presents a brief history of on-board processing and how this has developed into satellite autonomy.

2.2 Geostationary and low-Earth orbits

A satellite may be placed in many possible types of orbits at varying altitudes. A special orbit type is the geostationary orbit (GEO). The orbit has an altitude of 35786 km and a latitude of 0° [2]. The orbit is widely used for broadcasting and was first presented in a paper by the science fiction writer Arthur C. Clarke in 1945 [3].

The article was written more than a decade before the first satellite, Sputnik I, was launched (1957) and almost two decades before the first GEO satellite, Syncom 2, was launched (1963).

What makes this orbit unique, is its period, which is equal to one day. This means a GEO satellite appears stationary to an Earth observer. It is also possible for three cooperating GEO satellites to provide global coverage. The global coverage makes GEO satellites ideally suited for broadcasting, communications, meteorological remote sensing and navigation.

Because the satellite is stationary with reference to a point on the Earth's surface, there is also no need for tracking antennas on ground stations and the link margin remains constant. These factors simplify the task of ground station design.

Because of the high altitude of GEO satellites, there is a significant time delay in communications. One-hop GEO communications can experience a transmission delay from 250 ms to 280 ms and when processing time is included in the estimation, the delay can exceed 300 ms [4]. The delay represented here is for one-hop systems, where the transmission path is from a transmitter to the satellite and back to an Earth terminal. This scenario is an example of regional coverage for a GEO satellite operator. When international coverage is required, the scenario becomes a two-hop or multi-hop system, with longer delay times.

The development and launch of a GEO satellite are expensive, and assurances of adequate return on investment (ROI) are required, before the design of a GEO satellite is considered. Broadcasting and telecommunications provide for adequate ROI and that is why GEOs are regularly used for these purposes.

Low-Earth-orbit (LEO) satellites orbit at altitudes of 300 km to 1500 km. Higher altitudes require more powerful launchers, which cost more, and lower altitudes provide higher spatial resolutions for remote sensing satellites [5]. Higher altitudes also provide greater coverage area and longer delays. From Kepler's third law, satellites travelling at higher altitudes have lower velocities [6]. LEO satellites, therefore, have much higher velocities than, for example, GEO satellites. The high velocities gives LEO satellites a short available communications time window (CTW), in which to communicate. The CTW length is the length of time a ground station is in line of site contact with the satellite. The short CTWs create the need to use the available time as efficiently as possible.

In order for a LEO satellite to cover most of the globe, it should have a highly inclined orbit, where the satellite travels from pole to pole as the Earth rotates underneath it [7]. Another requirement might be for the satellite to visit the same point during approximately the same times each day. This orbit is called a sun-synchronous orbit, which is useful for remote monitoring and remote sensing applications [8].

Remote sensing applications image the surface of the Earth, and passing over a point during the same time each day allows for images of the same time frame to be compared. Remote monitoring applications, where the satellite collects data from ground based sensors, have the same advantages.

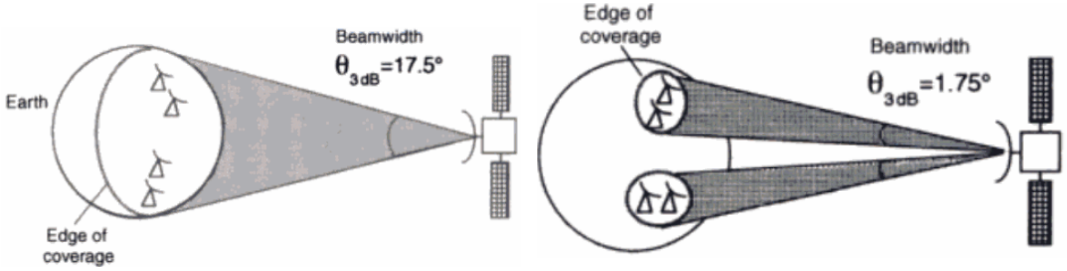
LEO satellites are usually smaller than GEO satellites, and therefore cheaper to build and design. Because of the lower orbit, they are also cheaper to launch. They are well suited for remote sensing and remote monitoring applications as previously discussed in Section 1.3. Data communications also possess very low delay rates, because of the low altitude. An average delay rate is approximately 18 ms round-

trip time (RTT), as later calculated in Section 3.2. This delay is negligible when compared to a 280 ms delay of GEO satellites.

2.3 LEO communications and tracking

An important difference between a LEO and GEO is that the distance between a LEO and a point on Earth changes with time whereas the distance between a GEO and any point on Earth remains constant. This complicates ground station design, because of a dynamic link budget in the LEO case. In other words, the quality of the link varies as the distance between the satellite and a ground station varies.

One solution to this problem is to have an antenna on the satellite with a wide beam width. This allows for a large satellite footprint, where the ground station can be detected early on and stay in the footprint for an adequate amount of time. The issue with this solution is that the antenna gain decreases as the beam width increases [9].



(a) Global coverage with a single beam (b) Coverage by several narrow beams

Figure 2.1: Satellite antenna beam types and coverage [9]

Two methods can be used to improve the single beam system. One is to use multibeam antenna arrays, which produce multiple narrower beams, instead of one large beam [9]. Figure 2.1a shows an antenna array with a single beam and wide beamwidth. Figure 2.1b shows an antenna array with multiple beams, where each beam has a narrow beam width.

Multiple beams allow for single frequency reuse, as different signals are spatially separated, and therefore do not require separate frequencies. The advantage of frequency reuse is efficient utilisation of the frequency spectrum. Although satellite systems are starting to use higher frequencies, lower frequencies are still favoured due to the lower power requirements and lower levels of interference [10].

Another method is to use a scanning antenna array [11], [12]. These antennas consist of single or multiple beams, electronically steerable to enable the antenna to sustain its links to specific areas for longer periods of time at the cost of other uncovered areas. This is an efficient design for mobile applications as the mobile terminals may be tracked while the connection is ongoing. Beams can also constantly scan the desired area to search for possible connections.

A method that can be used to further improve the link margin, is to have a steerable antenna on the ground station. A ground station possesses more power

than a satellite, therefore, a high gain antenna can be used to track the satellite in orbit. As the satellite passes by the ground station, the tracking antenna is constantly updated with the position of the satellite. A disadvantage of this system is the high cost of the antenna rotator and interface equipment. The equipment is responsible for steering the antenna. A quote was received for the antenna rotator and interface for R19155, including shipping. This price excludes the antenna and its design and is per ground station.

Three tracking methods are used by ground stations to track a satellite: monopulse tracking, lobing tracking and program tracking [13]. Of the three methods, the first two make use of beacon signals received from the satellite. These signals are created by a satellite transponder and measuring the characteristics of the signal allows control systems in the ground station to steer the ground station antenna to track the satellite. Program tracking uses known orbital characteristics of the satellite to the predict the satellite orbit forward in time. This technique is known as orbit propagation. Program tracking does not require a tracking feed and tracking receiver and high levels of link quality may still be attained. This technique is used as a backup to the other two techniques or as primary tracking method for low cost designs.

2.4 Big and little LEOs

LEO satellites can effectively be divided into two types. Big LEOs and little LEOs [14]. Big LEOs are, as the name specifies, larger, more complex systems. Big LEOs are LEO satellite systems that can provide many different services. These include voice, data and fax, SMS and paging, search and rescue, disaster services, environmental and industrial monitoring, cargo tracking and location determination.

To provide these services, big LEO systems usually consist of satellite constellations. A satellite constellation consists of a network of multiple satellites with ground station to satellite links as well as inter-satellite links. This enables calls to be handed off to neighbouring satellites when the current footprint passes out of view of the ground station being serviced. This handover technique enables a network of LEO satellites to provide uninterrupted connectivity to an end-user. The drawback is that many LEO satellites are required to provide the service and the service is not operational until all satellites are in orbit. Reserve satellites are also required to replace a LEO satellite that might malfunction.

Two big LEO constellations currently operational are Iridium and Globalstar. The Globalstar network consists of 48 operational satellites, with 4 in-orbit spares and the Iridium network consists of 66 operational satellites and 14 in-orbit spares. These numbers show the large investment that has to be made before any return on investment can be obtained. Iridium required an initial investment of $7 billion and became operational in November 1998. In August 1999 Iridium filed for Chapter 11 bankruptcy protection in the United Stated of America (USA). The Iridium service was later relaunched in March 2001, but neither Iridium nor Globalstar have shown significant profits.

Little LEOs, on the other hand, are designed to be simple, inexpensive store-and-forward communications systems. The application focused on in this work is remote monitoring, as discussed in Section 1.3. Remote monitoring can be effectively

implemented with a store-and-forward system as this system relies on implementing off-line data transfer services instead of real-time telephony services.

Two examples of little LEO systems are Leo One and Orbcomm. While both of these systems are also constellations, the services that they provide differ greatly from big LEO services. These constellations also do not offer continuous coverage, since a little LEO system does not require it. The same services, but on a smaller scale, can also be provided by a single little LEO satellite in orbit.

Store-and-forward systems carry data sent from a source to a destination ground station. A satellite travels over one ground station and connects to it. The ground station can then send data to another ground station by uploading the data to the satellite with instructions on where to send it next. The data are then stored on the satellite until the satellite passes over the target ground station. When that ground station connects, it can receive data destined for it and also send data of its own. This technique promotes the transport of low volumes of data from many stations to many stations and can easily be adapted for remote monitoring systems.

To implement remote monitoring over store-and-forward, all sensor stations send data to a central processing station when the satellite passes over the sensor stations. When the satellite subsequently passes over the processing station, all measurement data are downloaded to the station for processing. This scenario is a special case of store-and-forward, as there are many stations only uploading data and one station downloading data. The store-and-forward implementation is, however, generic and multiple processing stations may exist if the specific remote monitoring application requires it.

2.5 LEO link acquisition

Link acquisition is the process of establishing a communications link between the satellite and a ground station. This process includes both the tracking and acquisition of the satellite to establish communications.

Currently, a ground station-satellite connection is established when the satellite comes within communications range of the ground station. The ground station determines when the satellite is within range by tracking it. The ground station uses a steerable antenna array, while the satellite has a low power directed antenna [15].

Two examples of little LEO link acquisition can be found in LEO One and Orbcomm [14], as mentioned in Section 2.4. Both LEO systems use ground stations to establish connections with the satellite. In the Orbcomm example, a Gateway Earth Station establishes a connection with the satellite based on orbital information received from a Gateway Control Centre. This connection is established on the basis of when the satellite is in communications range of the ground station. The satellite possesses seven antennas to provide multi-user access to the system. Multiple antennas, combined with the gateway station design, significantly lowers the total number of concurrent users that the satellite is able to handle. The Satellite Communication Units used, also operate through a gateway system.

Link acquisition protocols for LEO satellites have received little attention. This protocol is initiated when the satellite is within range of the ground station and is ready to communicate. According to the two major space standardisation or-

ganisations, the Consultative Committee for Space Data Systems (CCSDS) and the European Cooperation for Space Standardisation (ECSS), the specific design of the link acquisition protocol is largely left to the designer. Where the subject of satellite ranging and tracking is discussed, it is assumed ground stations initiate the connection [16]. The reason for this is that ground stations usually possess high power steerable antennas, which they use to initiate connections with the satellite, as discussed in Section 2.3.

For a system with a steerable antenna on one of the communicating entities, the connection has to be initiated by the entity possessing the steerable antenna. Take for example, the situation where the satellite possesses a steerable antenna and the ground station a non-steerable antenna. If the satellite wishes to initiate a connection with the ground station, it has to point the antenna in the direction of the ground station and it will then have sufficient link quality to inform the ground station that it wishes to establish a connection. The ground station will not be able to initiate a connection with the satellite as the satellite antenna might be pointed away from the ground station, and therefore not possess sufficient link quality.

2.6 On-board processing and satellite autonomy

Initially, satellites used transparent transponders to allow ground station communications [17]. No on-board processing was performed and all data received by the satellite, was mixed to another frequency and transmitted again. This greatly simplified the design of the satellite, because all data processing wasSouth performed on the ground stations. This also allowed for an adaptive system, because any form of data could be sent across the satellite link and the format and meaning of the data was only determined by the transmitting and receiving ground stations.

The transparent transponder method was viable as the satellite only had a few tens of communications channels. This number increased to several hundred communication channels at the end of the 1970s [18]. Even with this number of channels, on-board management could still be satisfied by only using a command decoder and a telemetry encoder. This changed in the 1980s, with satellites becoming more complex, with many more required channels (4400 for INTELSAT VI). The increase in required number of channels and the advent of radiation hardened microprocessors, ushered in a modular approach to satellite management.

The modular architecture placed all separate systems on a communications bus. From the bus, they could receive information sent to them and also communicate with other satellite systems. All functions could also be coordinated by a *central terminal unit*. This approach simplified the design cycle, by decoupling the different satellite systems.

Today, all satellite sub-systems are controlled by the *telemetry, command, data handling and processing* sub-system [19]. The sub-system receives telemetry data from all other sub-systems and presents the satellite control interface to the satellite operator. Functions performed by the data-handling sub-system include enabling the flow of housekeeping, receiving and distributing commands, implementing telemetry and telecommand protocols, distributing the system time for synchronisation throughout the system, providing data storage, controlling payloads and sub-systems

and making autonomous decisions.

On-board processing also allows for regenerative techniques to be implemented [17]. Regenerative techniques allow for increased link quality by performing error detection and correction on-board the satellite. Data received by the satellite are no longer only mixed to a different frequency and retransmitted. When a signal is received, the signal may be mixed down to base band and error correction techniques may be applied, before retransmitting the signal.

LEO satellites usually require advanced on-board data handling techniques. The reason for this is the limited LEO communications time, which necessitates that all possible data filtering, processing and compression should be performed on-board the satellite, before the satellite transmits the data. Initially, GEO satellites transmitted all raw data to be processed by the ground stations. As mentioned earlier, this simplified the satellite design. This method is not viable for LEO satellite systems as utilisation of the communications link should be optimal, because of the short window available. This sets the requirement that all processing that might reduce the amount of data transmitted, should be performed on-board, before transmission.

From this mandate of decreasing the channel load and further improving satellite data handling, comes the concept of satellite autonomy. Satellite autonomy states that to reduce the channel as well as operator load, functions that can be performed on-board the satellite, should be. Sub-systems should be able to make decisions and implement those decisions with a certain level of autonomy. Autonomy is defined as a hierarchical structure. The top level receives input, decides which *agent* is best equipped to make a decision regarding the input and relegates the authority, with the received information, to the agent [20]. Agents know how to control specific sections of the satellite and have the autonomy to implement decisions on the sections.

Because of the fragile nature of the satellite communications link and the high cost of building and launching a satellite, fine grained operator control was seen as the safest way to manage the risk. Any anomaly encountered during the mission was handled directly by an operator. The reliance on operator control does, however, degrades the performance of the system while increasing costs. The reason for the degradation in performance is the time operators take to make and implement decisions. Satellite autonomy is shifting the responsibility of satellite operations from the ground operators to the satellite.

This shift from a ground station controlled satellite network to a satellite controlled satellite network also allows other improvements to the rest of the communications system [21], [22]. This shift from a ground station centric system to a satellite centric system forms one of the pillars on which this work is built and is known as satellite autonomy.

2.7 Conclusion

The goal of this chapter is to put into perspective all design decisions taken. To enable the reader to do this, the chapter explained the LEO communications system in a top-down manner. The chapter started by presenting the differences between GEO and LEO satellite systems. GEO orbits and the limitations of GEO orbits were discussed. The two main limitations are long communication delays and the high

costs involved. LEO orbits were then discussed, along with the various advantages. These advantages included shorter delays, lower cost and a shorter development life-cycle.

LEO communications and tracking were further discussed. This included the different types of antennas, how multi-beam antennas improve communications and how satellite tracking and ranging is performed. This section also describes why satellite tracking for LEO systems are important.

It should be noted that any technique to improve communications, improve the link between the satellite and ground station. If link improvement is required, any technique on the ground station or satellite to improve communications will improve the overall link. A trade-off can then be made when a link of a certain quality is required. If a link budget shows a system with omni-directional antennas on both the satellite and the ground station produces a link of insufficient quality, improvements can be made to either the satellite, the ground station or both. It is important to choose on which system the improvements should be made and to explore the different effects every solution will have.

Adding a steerable antenna to each ground station in the system may be very costly compared to only adding a steerable antenna to the satellite. It is also possible to add a steerable antenna to the satellite and then find that the link quality is still insufficient and that a steerable antenna on the ground station is also required. The reason for this argument is to provide perspective on the current system being designed. The system is designed in such a way that the satellite and the ground station will not both require steerable antennas to achieve a sufficient link budget.

The next section introduced little and big LEOs. Big LEOs are defined as complex communications systems able to carry voice and data. Iridium and Globalstar are presented as two examples of big LEOs. Little LEOs are defined as less complex, small satellite systems providing store-and-forward services. Remote sensing and monitoring are presented as examples of this system and Leo One and Orbcomm are presented as examples of little LEO systems.

Link acquisition is then discussed and how this is achieved in little LEO systems is also presented. The lack of standards for well defined link acquisition protocols is also mentioned. The necessity of having a satellite with a steerable antenna initiate the connection to a ground station is also explained for a situation where the ground station possesses no steerable antenna.

The higher layer on-board processing functions of the satellite is then discussed. How the concept of on-board processing evolved is examined and the benifits of on-board processing are also discussed. The conclusion to on-board processing is then presented as satellite autonomy. An autonomous satellite is introduced as not only able to store and regenerate received data, but also implement decisions based on received data, without the necessity of ground operator control.

The structure of this study was chosen so as to present the satellite system under development, but also to present alternative methods of development. The satellite will be a LEO satellite with a single steerable antenna on-board. Ground stations will not possess steerable antennas and will implement program tracking to determine the position of the satellite. The satellite will be a single little LEO satellite, that uses its steerable antenna to initiate connections with ground stations. The satellite link acquisition control is also a possible example of satellite autonomy as the satellite and

not the operator determines when to establish a connection, which it calculates from predicted satellite and ground station positions. A custom link acquisition protocol is developed and forms part of the focus of this work. Another focus of this work is the on-board processing system on the satellite. The developed SCSS implements many of the mentioned on-board data handling techniques, including file storage.

Chapter 3

Satellite System overview

3.1 Introduction

As it is important to be able to place the designed SCSS into perspective, this chapter presents an overview of the satellite communications system, thereby providing the reader with a context from which to view this work. An initial background of the project was given in Section 1.1 and the origins of the project were discussed. In this chapter, the overall communications system design is discussed and some engineering design decisions are also presented.

The literature presented in Chapter 2 gives an overview of current space standards and how these are used in space designs. The satellite system is, however, a unique design and should, therefore, be evaluated on its own grounds. The purpose of this chapter is to present a counterpoint to the literature study, where the unique aspects of the system are shown and discussed.

Section 3.2 presents an overview of the physical characteristics of the satellite under design, while placing particular emphasis on the available communications time of a LEO satellite system. Section 3.3 presents a high level view of the communications system and shows how the satellite system can be incorporated into the Internet. It also presents the protocol stack and describes the protocols and applications implemented on each level of the stack. Section 3.4 discusses the hardware used in the satellite design and various software and hardware interfaces are presented. Section 3.5 introduces the operating system on which all software developed for the SCSS executes. It briefly introduces the architecture of the operating system and explains how this influenced the system design. Section 3.6 discusses the steerable antenna used on the satellite and how this affects the rest of the system design.

3.2 Orbit characteristics

The satellite is in a circular polar sun-synchronous orbit [8] at an altitude of $h = 520\,\text{km}$ and an inclination of $i = 96{,}5°$. Such an orbit ensures a satellite will pass through a certain horizontal line at approximately the same local time for each crossing. A satellite might pass over the equator twelve times a day, each time passing the equator at around 12:00, *local time*. The sun-synchronous orbit of the satellite ensures the satellite travels over South Africa at around the same time each

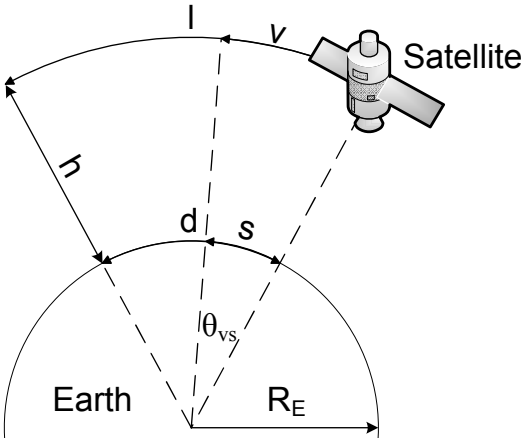


Figure 3.1: Satellite orbit properties

day. The average radius of the Earth is taken as $R_E = 6371\,\text{km}$ [23]. Figure 3.1 shows a diagram of the satellite orbiting the Earth.

From these values, the approximate velocity of the satellite may be calculated. If the Earth is assumed to be circular, p is defined as the distance the satellite travels in a day and is given by the formula for the circumference of a circle

$$\begin{aligned} p &= 2\pi\left(R_E + h\right) \\ &= 2\pi\left(6371 + 520\right) = 43\,297\,\text{km}. \end{aligned} \tag{3.2.1}$$

v is defined as the velocity of the satellite and may now be calculated by

$$\begin{aligned} v &= \frac{p}{T_S} \\ &= \frac{43\,297\,\text{km}}{94{,}95\,\text{min}} = 27\,360\,\text{km/h}, \end{aligned} \tag{3.2.2}$$

where T_S is period of the satellite, which can be calculated using Kepler's third law [6] where

$$T_S = \sqrt{\frac{4\pi^2\left(R_E + h\right)^3}{gR_E^2}} \tag{3.2.3}$$

$$= 5697\,\text{s} = 94{,}95\,\text{min}, \tag{3.2.4}$$

where R_E and h are both expressed in metres and $g = 9{,}81\,\text{m/s}^2$ is the nominal Earth gravitational acceleration at sea level. T_S is also used in Section 4.5, to assist in predicting the position of the satellite as a function of time.

The velocity s of the satellite's footprint, travelling along the Earth's surface, can be calculated next. This is done by projecting the satellite's velocity onto the surface of the Earth. From Figure 3.1, it can be seen that l, the length of the arc the satellite makes when travelling through space, is subtended by the arc d, the length of the footprint's arc on the surface of the Earth. s is then given by

$$s = v\frac{d}{l}. \tag{3.2.5}$$

From the geometry of Figure 3.1, the ratio of the satellite arc length to that of the ground station arc length is given as

$$\begin{aligned} \frac{d}{R_E} &= \frac{l}{R_E + h} \\ \therefore \frac{d}{l} &= \frac{R_E}{R_E + h}. \end{aligned} \tag{3.2.6}$$

When substituting equations (3.2.6) and (3.2.2) into Equation (3.2.5), the new equation becomes

$$\begin{aligned} s &= v\frac{R_E}{R_E + h} \\ &= 2\pi\frac{R_E}{T_S} \\ &= 25\,296\,\text{km/h}, \end{aligned} \tag{3.2.7}$$

which shows that the velocity of the satellite's footprint is not much smaller than that of the satellite itself. The reason for this is the low altitude of the satellite, compared to the radius of the Earth.

$t_{x,\min}$, the minimum channel delay time, can be calculated using

$$\begin{aligned} t_{x,min} &= \frac{h}{c} \\ &= \frac{520}{2,998 \times 10^5} = 1{,}73\,\text{ms}, \end{aligned} \tag{3.2.8}$$

where c is the speed of light in a vacuum [24]. $t_{x,\min}$ gives the minimum travel time of a packet between the satellite and a ground station directly below the satellite.

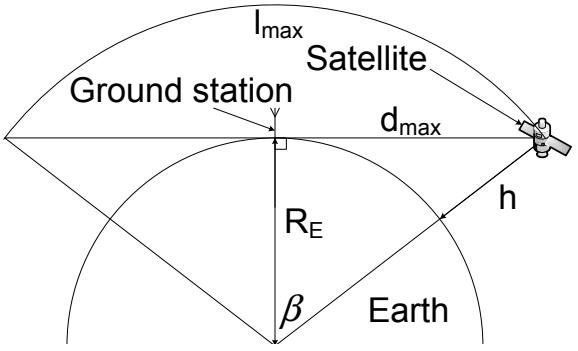


Figure 3.2: Line-of-site parameters used to calculate the maximum satellite-ground station communications distance.

Figure 3.2 shows the point at which a ground station is able to initiate communications with the satellite. This is where the satellite intercepts the tangential horizontal line drawn from a ground station on the Earth's surface. The maximum communications range can now be calculated using basic trigonometry, where

$$\beta = \arccos\left(\frac{R_E}{R_E + h}\right) \tag{3.2.9}$$

$$d_{\max} = R_E \tan\beta, \tag{3.2.10}$$

where $2 \times \beta$ is the angle subtending the arc $l_{\max}$ and $d_{\max}$ is the maximum communications range for a satellite at height h. Using (3.2.10), the maximum communications distance is calculated as 2626 km for $R_E = 6371$ km and $h = 520$ km. This distance corresponds to an elevation angle of 0°. $d_{\max}$ is later used in Section 4.9, to predict the length of time for which the ground stations are able to communicate. The maximum time a packet can travel can now be calculated using

$$\begin{aligned} t_{x,\max} &= \frac{d_{\max}}{c} \\ &= 8{,}76\,\text{ms}, \end{aligned} \tag{3.2.11}$$

where $t_{x,\max}$ is the maximum packet travel time. The maximum round-trip time (RTT) is then $\text{RTT}_{\max} = 2 \times t_{x,max} = 17{,}52\,\text{ms}$. The minimum and maximum channel delay times are calculated here, to show they will not significantly influence the communications time of a ground station compared to a GEO satellite. The maximum RTT is insignificant when compared to a ground station communications time of several minutes. The RTT, therefore, has no bearing on the calculations performed in Chapter 4, where ground station communication times are predicted.

To obtain the Communications Time Window (CTW) length, the length the satellite travels through space in visible range of the ground station is given as

$$\begin{aligned} T_C &= \frac{l_{\max}}{v} \\ &= \frac{2\beta(R_E + h)}{v} \\ &= \frac{T_S}{\pi} \arccos\left(\frac{R_E}{R_E + h}\right), \text{ from (3.2.2)} \\ &= 11{,}8\,\text{min}, \end{aligned} \tag{3.2.12}$$

where $l_{\max}$ represents the visible path length of the satellite and T_C represents the CTW length. Equation (3.2.12) calculates the maximum time possible for a ground station to communicate with the satellite. This is a best-case scenario, assuming an adequate link budget. The scenario assumes the satellite is passing directly over the ground station, which is less likely to happen than the satellite passing by somewhere on the periphery of visual range. The CTW length calculated is, therefore, a maximum length, which is corroborated with the simulation results shown in Section 6.3.

3.3 Communications overview

Figure 3.3 shows a high level overview of the satellite ground station communications system and how it connects the Internet to rural networks. The main message this figure attempts to convey, is that the goal of the satellite system is to link rural networks to the Internet or to other rural networks. This link is only open for a short period of time as explained in Section 3.2, so the link can be used to transport batches of non-real-time data. The main application of this system is remote monitoring, as discussed in Section 1.3. Figure 3.3 shows three networks: the Internet, the satellite communications network and a rural network.

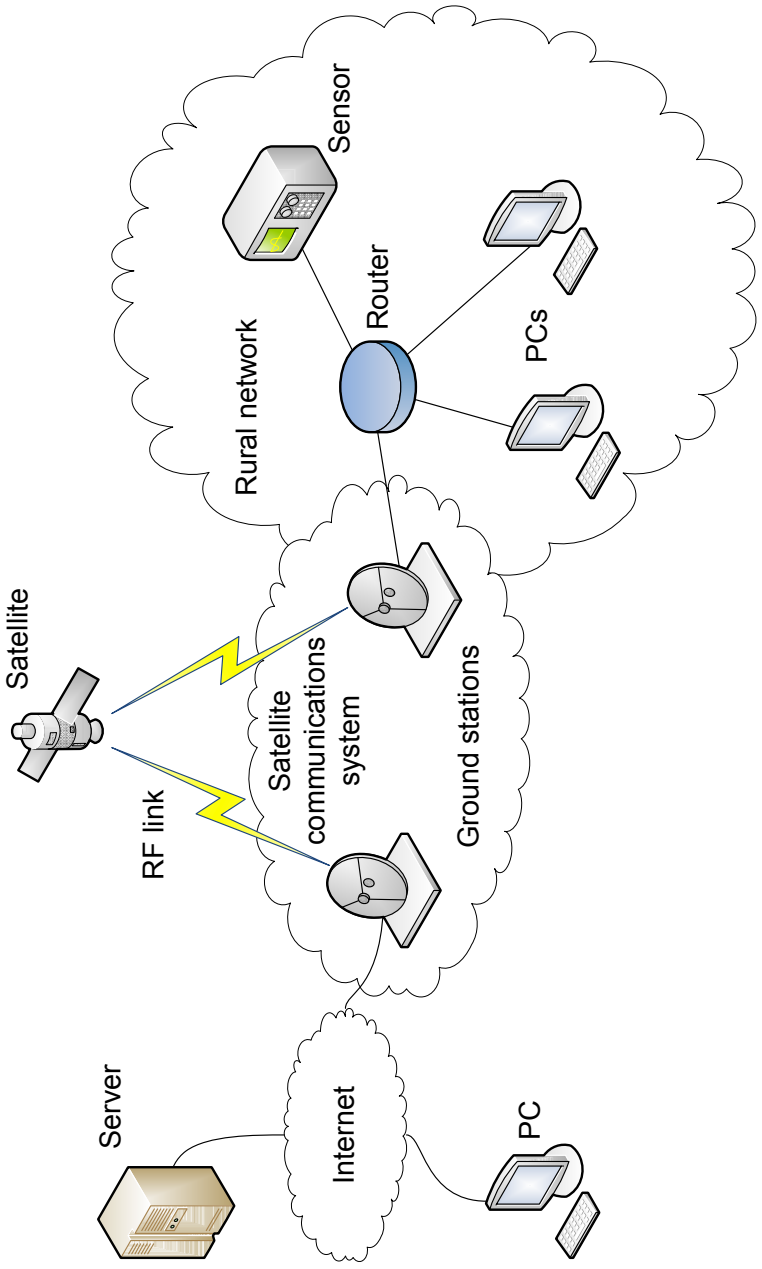


Figure 3.3: An overview of the satellite communications system.

The rural network is defined as some network possessing no forms of connectivity other than that of the satellite network. The rural network might consist of only the ground station, where all data to be delivered are delivered to the ground station and all data to be sent are produced by the ground station. The rural network might also consist of a network of devices such as sensors and computers, connected to the ground station through a router or switch. This further illustrates the remote monitoring application, discussed in Section 1.3.

The satellite network consists of only the ground stations, transmitting data over the radio frequency (RF) link, to the satellite. The satellite is, therefore, unaware of any other devices connected to the network. Ground stations use the satellite network to send data to other ground stations and for this reason the satellite can be viewed as a transmission medium rather than a separate node in the network. This is the simplest design satisfying all applications for which the system is designed.

Some ground stations may be connected to the Internet, to allow any remote computer to transmit data to a rural computer. All user permissions are controlled by the ground station, which can be seen as the gatekeepers of the satellite system. The permissions of the ground stations themselves are in turn controlled by the satellite as described in Section 5.4.3. Nodes that wish to send data to a remote node must do so via a gateway ground station. A TCP/IP connection is set up to the gateway ground station and the data are delivered to that ground station. It is then the duty of the gateway ground station to ensure the data are sent to the correct rural ground station, which will in turn deliver it to the correct rural node.

What should also be made clear from Figure 3.3, is that the satellite-ground station communications system is isolated from the greater Internet. This is because of the substantial delay encountered when transmitting data over the satellite link, which would adversely effect an Internet-based protocol stack. The delay is caused by the time it takes for the satellite to become available for communications. This delay can be anything from a few minutes to eight hours, depending on what the epoch of the satellite is. See Section 4.9. The epoch is defined as the position of the satellite in the satellite orbit track.

The transport control protocol (TCP) will severely reduce its throughput rate after waiting for a significant amount of time. The reason is that the protocol recognises the delay as network congestion and activates its congestion avoidance mechanism [25], [26]. This leads to underutilisation of the satellite link. One solution is to implement an adapted TCP protocol which is designed for space links with long delays. There are still many issues with TCP over space links, and while some solutions have been provided, substantial research still has to be done on the subject [27].

For the store-and-forward system implemented, the TCP protocol for space links is more complex than required. TCP is an end-to-end protocol, meant to create a reliable link over a multi-hop path. Its function is also to ensure data delivery and maximally utilise the link. For a remote monitoring system this functionality is not required, since the volumes of data are very low and the amount of time that a link is open is very short. The added complexity, combined with the issues of space communications, makes the TCP protocol unsuitable for the current implementation. The default protocol stack of the on-board operating system makes use of normal TCP and is, therefore, not suitable for space communications as an off-the-shelf

solution.

The isolated nature of the satellite system removes the requirement for a network layer on the satellite protocol stack. The function of the network layer is to provide routing between different nodes in the network. For the isolated satellite network, where a connection is always only from the satellite to one ground station at a time, the connection is only one hop long, and therefore requires no routing. A routing layer might, however, be required on ground stations, to enable messages sent to one ground station over the Internet to be delivered to a remote host on a rural network.

The implemented satellite protocol stack is loosely based on the Open System Interconnection (OSI) reference model [28]. This model has a layered design to encapsulate functionality in different layers. Every protocol layer adds a different layer of functionality to the overall communications system that can be used by higher layers. The design uses a four layer protocol stack as shown under "OSI layer" in Figure 3.4. These layers are the physical layer, data link layer, transport layer and application layer.

OSI Layer	Implementation	Location
Application layer	SCSS	OBC
Transport layer	ARQ	
Data link layer	TM Space data link protocol sub-layer	
	TM Synchronisation and channel coding sub-layer	FPGA
Physical layer	QPSK	Modem

Figure 3.4: Satellite communications protocol stack, showing OSI layer, implementation and hardware type.

The physical layer is the RF layer. This layer performs modulation and demodulation to transmit and receive RF information from the transmit and receive antennas. The data link layer controls the data connection between two adjacent nodes in the communications system and also implements error correction and detection. The transport layer adds reliability to the network stack. Usually, the transport layer also supports end-to-end connectivity between multiple nodes in the network, but the satellite-ground station network only consists of the satellite, which provides the communications path, and the ground stations that transport data over that path as explained previously. The application layer implements high-level file-based communications. It is in this layer that the SCSS and the communications strategy are implemented.

In each protocol layer, a communications protocol is implemented to provide the described features as shown under "Implementation" in Figure 3.4. The physical layer is the wireless link between the satellite and ground stations employing a Quadrature Phase-Shift Keying (QPSK) data carrier system [29]. The data-link

layer uses the Telemetry (TM) protocol, designed by the CCSDS. The TM protocol is an unreliable protocol and so reliability is implemented in the transport layer in the form of Automatic Repeat-Request (ARQ) [30]. Correct data transmission is not ensured in an unreliable protocol. Packets may be dropped and the protocol will make no attempt to notify the sender that the data were not sent successfully. ARQ adds reliability to the protocol stack, by resending ARQ packets that failed to reach their destination and employing an acknowledge mechanism to inform the sender of a successful data reception. The application layer consists of the high layer SCSS further described in Chapter 5 and which forms the focus of the thesis. The SCSS initiates connections with ground stations and handles requests by these stations. Requests are in the form of "message" files, uploaded to the SCSS.

The TM data link layer consists of two sub-layers: the channel coding and synchronisation sub-layer [31] and the data link protocol sub-layer [32]. The synchronisation and channel coding sub-layer of the data link layer implements error-control coding, frame validation, synchronisation and pseudo-randomisation. A change was made to the CCSDS standard, to remove the three error-coding schemes mentioned and to implement Low Density Parity Check (LDPC) codes [33], [34]. These codes provide a quality of error detection and correction potentially near to the Shannon limit [35]. LDPC codes can be used to replace the codes specified in the standard, because these codes are also sparse graph block codes, similar to Turbo codes [36]. The data link protocol sub-layer implements the data connection between two adjacent nodes in the communications system. Fixed-length protocol data units are used to support simple, reliable and robust data transfer over the low quality space link.

The protocol stack is implemented across different hardware as shown under "Location" in Figure 3.4, which are discussed in further detail in Section 3.4. The application layer, transport layer and TM space data link protocol sub-layer are all implemented on the on-board computer (OBC). The TM synchronisation and channel coding sub-layer is implemented on the FPGA and the QPSK data carrier is implemented on the modem board.

3.4 Hardware and interfaces

Figure 3.5 shows the different hardware components in the satellite, the different interfaces that exist as well as data flow for a transmission. The satellite communications hardware consists of three parts. These are the field-programmable gate array (FPGA) board, the DSP modem board and the on-board computer (OBC) board. All analogue modulation and demodulation are done on the DSP modem for low level RF communications. The modem board implements the physical layer of the protocol stack as described in Section 3.3. Error correction and detection, encoding and decoding as well as the routing of all communication flows are performed on the FPGA board. The board implements the synchronisation and channel coding sub-layer of the protocol stack as described in Section 3.3. The FPGA board is well suited for error control coding, because of the ability of the error control coding operations to be segmented and calculated concurrently. The board on which all high level applications execute is the OBC processor board. The OBC consists of a CPU, memory and secondary storage. It also houses a real-time operating system

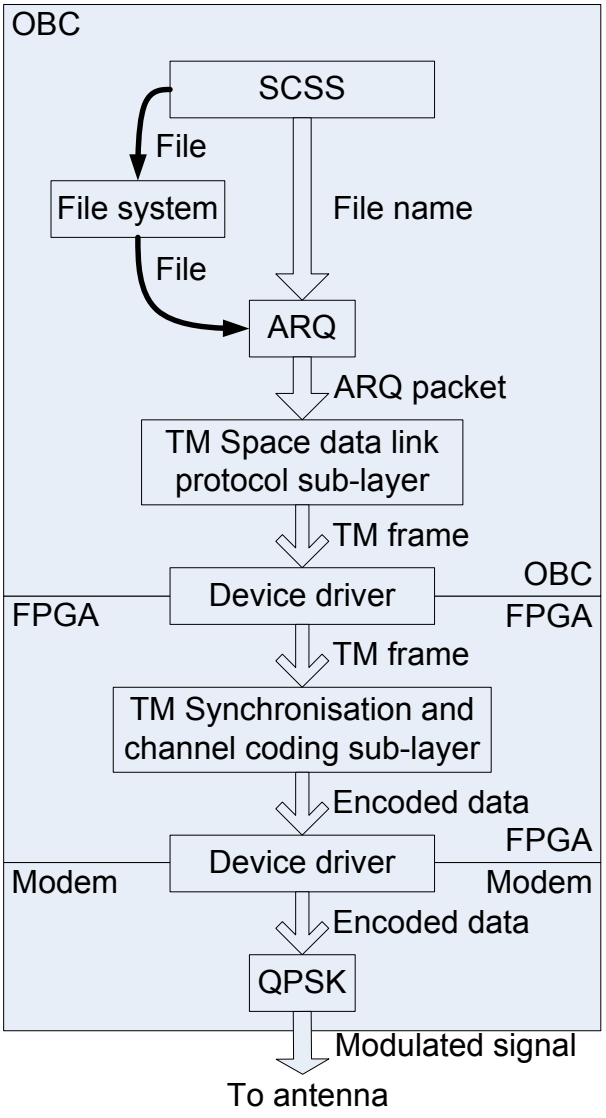


Figure 3.5: Flow of a transmission message through the satellite from the OBC, through the FPGA to the modem, showing all entities present in the different hardware.

called QNX, described in Section 3.5. The application layer, transport layer and the communications sub-layer of the data link layer are all implemented on the OBC board as explained in Section 3.3.

There are two hardware communication interfaces in the satellite system: the OBC-FPGA interface and the FPGA-modem interface. Device drivers are implemented to enable communications between the OBC and FPGA and between the FPGA and modem. Many software interfaces exist. From the perspective of the SCSS, these include the interfaces between the SCSS and the antenna control system, the real-time clock, the satellite bootstrapper and the lower level protocol stack as shown in Figure 5.2. The interfaces between the SCSS and the lower level protocol stack and the SCSS and the antenna control system are implemented as inter-process communications interfaces between two separately executing programs. The real-time clock interface is integrated into the operating system and the time functions of the QNX library can be used to retrieve the current time. The satellite real-time clock updates the OBC board with the current time when the OBC board initialises.

The interface between the SCSS and the lower level protocol stack in fact consists of two interfaces. The lower level protocol stack in turn also has multiple interfaces to enable ground station communications. The two interfaces of the SCSS to the protocol stack are the IPC interface mentioned earlier and a file system interface. Data are stored in secondary storage when received from the transport layer and the file name is sent across the IPC interface. The SCSS then reads the file from secondary storage and processes it, as discussed in detail in Section 5.7. The next protocol layer interface between ARQ and the TM space data link protocol sub-layer is also an IPC interface. An IPC interface is depicted by a 45° arrow in Figure 3.5. The next two interfaces between the TM space data link protocol sub-layer, the TM synchronisation and channel coding sub-layer and the physical QPSK layer are all interfaces spanning different hardware and so making use of device drivers. Device driver interfaces are depicted by a slanted arrow in Figure 3.5. The final interface is the RF link between the physical layer of the satellite and the physical layer of the ground station. This link is depicted by a 60° arrow in Figure 3.5.

For the SCSS to communicate with the ground station, it has to send data to the lower level transport protocol. For the purpose of transmitting data, the SCSS connects to the transport layer in a client-server fashion, where the SCSS is the client, the transport layer is the server and the service provided is communication with a ground station. As shown in Figure 3.5, to transmit a file from the satellite to a ground station, a file name is sent to ARQ. ARQ then reads the file from secondary storage. ARQ segments the file into ARQ packets and sends the packets to the TM protocol, making use of the first device driver. The TM protocol further segments the data and produces TM frames, which are encoded in the TM synchronisation and channel coding sub-layer. The encoded data are then sent to the modem, where they are modulated and transmitted by the antenna over the RF link to the ground station.

3.5 Operating system

The OBC described in Section 3.4 houses the real-time operating system called QNX Neutrino. QNX is a microkernel-based real-time operating system implementing the Portable Operating System Interface (POSIX) application programming interface (API) and is designed to run on embedded systems [37]. The key concept which QNX is built around is scalability.

At the core of the QNX architecture lies the microkernel design. The only functionality residing in the QNX kernel is the functionality to enable process and thread creation and to support thread synchronisation a well as message and signal passing between different threads and processes. All other standard services, usually found in the Operating System (OS) kernel such as the file-system and the TCP/IP stack are implemented as user-level services. This provides an easily extensible architecture, where new modules can be added to the OS and only modules that are required for the embedded implementation have to be included in the OS when it is deployed.

QNX implements the POSIX API as specified in 1003.1, as well as some extensions later included to enable threads and support real-time applications [38]. The POSIX API provides a standard interface for developers, even though the underlying architecture is much different from that of a traditional UNIX system. A standard API such as POSIX, supports code and developer reuseability. It also supports portability between different POSIX based systems. Practically, from a design perspective, this enabled the use of standard C library functions and knowledge of the standard functions, to build a portable system quickly and efficiently, without having to learn a new propriety API.

The combination of standardisation and scalability is what makes QNX an OS very well suited for embedded applications. All development for QNX is done in the QNX Momentics development environment, described in Section 6.2. QNX supports a full POSIX C library, which simplifies testing and ensures portability if no OS specific functions are used. The QNX operating system is loaded onto the OBC as an image, containing all modules required by the satellite communications system.

3.6 Radio Frequency communications

One of the main factors driving the development of this particular satellite system is the satellite antenna being developed by the Katholieke Universiteit Leuven in Belgium [39]. The design uses digital beam forming techniques to implement a steerable satellite antenna to communicate with ground stations. The steerable antenna significantly improves the satellite-ground station link quality, and therefore reduces power requirements.

The designed antenna has a steering angle of approximately 60°. This is the angle through which the antenna beam can rotate, before starting to lose significant gain. The lower this steering angle, the shorter the available communications time per ground station. The visible range is the range calculated in Section 3.2 and occurs when the only factor influencing the satellite-ground station link is line-of-sight.

The system design is driven by this steerable antenna and the designed system is also responsible for driving the steerable antenna. The SCSS executing on the

satellite is responsible for steering the antenna. The antenna has to be pointed in the direction of a ground station, before the satellite can communicate with that ground station.

The antenna can not inform the SCSS that it has turned successfully, because no simple mechanism can be implemented on the digital antenna, to allow it to know whether the phase by which it is driven, is correct. The SCSS can only command the antenna to turn and must then assume the antenna position is correct. It is the duty of the antenna control software to update the antenna position as the SCSS commands and also keep track of the satellite attitude.

3.7 Summary

This chapter presents an overview of the satellite communications system. The satellite orbit characteristics were calculated to mainly illustrate the limited time that each ground station has to communicate with the satellite and to also give an idea of the speed at which the satellite travels through space.

With the satellite characteristics given as a base, an overview of the satellite ground station communications was presented. This showed how the satellite communicates with a ground station, within the framework of the limited communications time of the LEO satellite. From this overview, the communications protocol stack was developed and the different layers described, as well as the specific protocols implemented on each layer.

The hardware section indicated the different parts of the satellite as well as explaining which layers of the protocol stack are implemented on which piece of hardware. The basic hardware description served to explain why the different protocol layers were implemented on the chosen hardware. To fully describe the OBC, a description also had to be presented of the operating system, which executes on the OBC. The QNX operating system was described and the benefits of the OS were mentioned i.t.o. the SCSS implementation.

The steerable satellite antenna was then introduced as the driving technology behind the satellite system. The antenna is controlled by the SCSS and is used in communications with the ground stations. These antenna characteristics influence the implementation of a scheduler as described in Chapter 4 to drive the communications system.

Chapter 4

Link Acquisition Control

4.1 Introduction

This chapter presents a communications strategy which addresses some of the shortcomings in current link acquisition schemes as discussed in Section 2.5. The satellite mountable, low power, high gain antenna introduced in Section 3.6, drives the devised link acquisition strategy. The antenna creates the opportunity of removing link control from ground stations and placing it on the satellite. It is now possible to have the steerable antenna on the satellite, instead of the ground station. If the link allows this, it leads to significant cost savings. To show how potentially expensive this can be, a quote was requested from an reseller. The cost of an antenna with an interface box, including shipping is R 19,155.00. This is the cost per ground station. For a system with 60 ground stations, this cost is R1,1 M.

The work performed in this section has also been published as two conference articles over a period of two years. The first article considers the dynamic satellite scheduling problem and describes a simulator developed to investigate this problem [40]. The second article considers the static scheduling problem, presents the prediction techniques used and shows how these techniques were combined to perform the visibility prediction as also shown later in this chapter [41].

As a central node, the satellite has knowledge of all ground stations. The benefits of centralised satellite control are: reduced ground station- and operational costs with increased cost utilisation and decreased response times [21]. Response times are increased, because no operator interaction is required.

To enable the transfer of satellite link control from the ground station onto the satellite, some requirements first have to be met. The first requirement, which has already been mentioned, is that the steerable antenna be located on the satellite, rather than on the ground stations, to enable contact initiation by the satellite.

To enable the satellite to control the antenna, the locations of all ground stations must be known to the satellite. This creates the requirement of having ground station information located on the satellite. This information necessarily includes ground station positions and required access times, but ground station information to enable quality of service (QoS) may also be added. This requirement is implemented as the station information file described in Section 5.4.3.

The question of when a ground station should be contacted, must also be ad-

dressed. The process of selecting one ground station from a list, for every instant in time, becomes a scheduling problem. Two possibilities were investigated. Initially, a dynamic scheduler was developed, which discovered ground stations as the satellite travels. After some investigation, it was determined that if the order in which ground stations would be discovered were known, a more efficient algorithm could be developed.

It was also determined that both the positions of the satellite and the ground stations can be predicted, and therefore the order can be calculated. This leads to the second scheduling algorithm, developed to increase the efficiency of link acquisition. The algorithm uses predicted satellite and ground station tracks to determine the order in which ground stations will be discovered.

Satellite orbit propagation is investigated and a basic orbit propagator is designed, implementing an interface which is easily accessible for the purpose of conducting experiments, and also satisfies the requirements of the scheduling algorithm. The scheduling algorithm is implemented, and simulated satellite and ground station tracks are used to calculate a schedule. The predicted communications times are also compared to those predicted by the well known Satellite Tool Kit (STK) software package and found to match very closely.

Section 4.2 presents a background on basic scheduling theory to enable the description of the satellite communications problem in scheduling terms. Section 4.3 discusses the motivation behind the move to a static scheduler and presents the advantages of using a static scheduler to a dynamic one. Section 4.4 develops the static scheduler design and the scheduling algorithm. Section 4.5 discusses satellite orbit propagation techniques and describes the techniques used to implement the custom orbit propagator. Section 4.6 describes how the ground station positions were predicted as objects orbiting a point in the Earth's axis of rotation. Section 4.7 combines the position prediction techniques used, to predict the satellite-ground station distance over time. Section 4.8 adds angle prediction to the system and discusses how this can improve the overall system performance. Section 4.9 uses the predicted distance and maximum communications range, to predict the satellite visibility over time. The predictions are also compared to predictions by the STK. Section 4.10 discusses how the scheduling scheme and the shift to centralised network control can help to improve the overall volumetric data throughput of the overall satellite-ground station system.

4.2 Satellite communications as a scheduling problem

Before a solution can be found to the satellite scheduling problem, the problem first has to be investigated. Section 4.2.1 initially presents the scheduling problem notation used and then develops the scheduling terms used during the rest of the work. When each term is defined, the analogous component of the satellite system is also introduced.

Three parts exist to every scheduling problem: the machine environment, job characteristics and optimality criteria. Section 4.2.2 describes the machine environment, Section 4.2.3 describes the job characteristics and Section 4.2.4 describes optimality criteria. Section 4.2.5 presents the satellite communications system as a

scheduling problem.

The concept of "required communications" time is also introduced in this section and will be used throughout the rest of the work.

4.2.1 Scheduling problem notation

The scheduling problem notation given in [42] will be used. The notation has the form:

$$\alpha|\beta_0, \beta_1, ..., \beta_i|\gamma$$

where α is the machine environment, β_i is the ith job characteristic and γ is the optimality criteria. The combination of α, β_i and γ in the notation uniquely describes a specific scheduling problem. For more information on Sections 4.2.2, 4.2.3 and 4.2.4, see Chapter 1 of [43].

4.2.2 Machine environment

A machine or server is the entity in a scheduling problem servicing the job. Throughout this chapter the word "machine" is used interchangeably with "server". The machine environment describes the characteristics of the servers in the scheduling problem. In single stage machine environments, jobs are only processed once. This is not to say that jobs may not be processed by more than one machine, but that no job, after being processed, needs to be processed again. Two common single-stage machine environments are: "1", there is only one machine and "Pm", there are m parallel identical machines.

The satellite system developed has one communications channel. In scheduling terms, this is a single machine problem where the channel in which GSLs have to scheduled is the machine or server of the system. This shows that the machine environment is "1".

4.2.3 Job characteristics

Jobs are units of work that need to be performed. Jobs are usually entities that need to be serviced/scheduled by some server or machine. To understand how to better schedule jobs, certain job characteristics are defined to characterise different types of jobs. β defines the job characteristics in the scheduling notation.

Some common job characteristics are: "r_i", each job has a specific release time, before which it cannot be scheduled. "d_i", each job has a specific due date or deadline, after which it cannot be scheduled. "$size_i$", each job requires a certain number of servers or machines on which it must be scheduled simultaneously. "prec", a precedence relation is given for each job, for example job i may only be scheduled after job x has completed. "pmtn", job pre-emption is allowed, meaning execution of jobs may be halted and resumed later.

The ground stations to be allowed to communicate are the jobs that must be scheduled on the communications channel. The main property of the ground station link (GSL) is its communications time. A GSL is the connection between a ground station and the satellite. The properties of a GSL are the properties of the communications link. The lengths of communication times can either be determined by the

ground station, or by the satellite. When the ground station determines communications time, the ground station may choose to use all the time it possibly can, in order to transport the maximum amount of data.

In an on-line system, this can adversely affect other ground stations near to the one currently communicating. Ground stations may not have sufficient communications times available to them to successfully complete their communications. This path may then lead to starvation of ground stations. Another method is where the satellite strictly controls allowed communications time. The strategy in this case is that a ground station requests a communications time length, before any ground stations are scheduled. This is a time agreed on, by both the ground station operators as well as the satellite operator. When the SCSS then schedules the ground stations, it knows beforehand how long each ground station will communicate and can strictly enforce the communications time.

The scheduler must ensure that the required communications time is awarded. If the time cannot be awarded, the ground station is not allowed to communicate as it is assumed that the ground station cannot perform meaningful work in a period less than its required time. This also means every ground station will always be assigned at least its required communications time, as further described in Section 4.4.

Although it is possible to implement pre-emptive scheduling, the overhead switching costs will be too high and a significant amount of complexity will have to be added to the satellite's administration communication system.

4.2.4 Optimality criteria

For every scheduling problem, some objective function must be minimised. How well the scheduling function is able to minimise the objective function is a measure of the performance of the scheduling function. It is possible to achieve an absolute minimum value for some combinations of objective functions and scheduling algorithms. When this can be done the scheduling scheme is said to be optimal.

Three common objective functions are

$$F_1 = \max\{C_i | i = 1, \ldots, n\}, \qquad F_2 = \sum_{i=1}^{n} C_i, \qquad F_3 = \sum_{i=1}^{n} w_i C_i,$$

where C_i denotes the finishing time of job J_i and $f_i(C_i)$ denotes the associated cost. F_1 is the maximum completion time, also called the "Makespan", F_2 is the total completion time of all jobs, also called the "Total flow time" and F_3 is the total weighted completion time of all jobs, also called the "Weighted total flow time". These functions are all common functions used when the processing time of jobs have to be minimised. For the satellite scheduling algorithm the total completion time is less important to the system.

What is important, is that every ground station is able to complete its communications within its available CTW. The criteria that should then be minimised is the total number of ground stations that could communicate, but were not scheduled. This form of scheduling is used in deadline driven scheduling schemes or real-time scheduling schemes. Soft deadline schemes exist, where scheduling past the deadline is allowed, but some cost is then incurred. The objective is then to minimise the

cost or the lateness as it is called. Hard deadline schemes do not allow a job to be scheduled after its deadline. If there is no way by which to have the job make its deadline, the job is not scheduled. The objective function in this case is then to minimise the total number of unscheduled jobs. This is the approach taken for the satellite system.

The reason for this is that the end of the CTW is the time after which it is no longer possible for the satellite to communicate with a ground station. If the station is scheduled after its deadline, it would, therefore, not be able to communicate. The fact that no communication is possible sets the requirement for the scheduler to be a strict deadline scheduler.

4.2.5 Scheduling problem statement

All sections of the scheduling problem are now known. The full scheduling problem is then

$$1|\ r_i, d_i\ |\mathrm{GS}_{\mathrm{dropped}}. \tag{4.2.1}$$

Equation (4.2.1) presents a summary of the single machine ground station scheduling problem. The jobs or connections have characteristics of specific release dates, stipulating when ground stations can initiate communications and due dates stipulating when ground stations are no longer able to communicate. The optimisation criteria is to minimise GS_{dropped}, the total number of ground stations able to communicate, but not scheduled.

4.3 Static vs. Dynamic scheduling

Initially, a dynamic scheduling approach was taken. This involved the satellite broadcasting beacons that a ground station receives when it is within communications range. When a beacon is received, the ground station responds with an acknowledge message and after the satellite receives the acknowledgement, the ground station is allowed to communicate or placed in a queue if there is already a ground station communicating. This system will function, but possesses some inefficiencies as described later in the section.

The ground station queue is controlled by a scheduler determining the order in which ground stations are added to the system. This allows the system to optimise the overall data throughput through the system by efficiently scheduling ground stations that wish to communicate. Greater efficiency is achieved by employing a shortest communications time first (SCTF) scheduling scheme. This is the time that a ground station requires to communicate with the satellite. This time is assigned when the ground station operator purchases capacity on the satellite. The operator requests a communication time for the ground station and a time of that length is assigned to the ground station. This information is stored on the satellite and the satellite uses the time to schedule the ground station.

The SCTF scheme minimises the "Total flow time" for the current set of ground stations [44]. The scheme can, however, never be optimal when viewed from a dynamic scheduling perspective [45], because although the scheme is optimal for a given collection of ground stations, more ground stations are regularly added. In

other words, the complete input set is not known beforehand. This reduces the efficiency of the scheduling algorithm. The issue with this solution is that the next ground station in the stream is not known before that station has been detected.

Another issue with the dynamic scheduler is the satellite will have to continuously broadcast discovery packets, even while communicating with a ground station, for the scheduler system to work. If no queues are used, the system can cease beacon transmissions while communicating with a ground station, but this system will have reduced ground station throughput, because of ground station set-up time overhead. These times may seem small, but when the system only has a few minutes to communicate, every minute saved is valuable.

A static scheduling algorithm removes the need for beacons as the algorithm is aware of the whole input set at the start of execution. It is also able to leverage the well developed field of static scheduling theory to develop an optimal and powerful scheduler. At the start of the system, the complete input set is known, which allows for all ground stations to be taken into account when scheduling every ground station. For the static scheduler, a custom scheme was developed in an effort to reduce the number of stations not allowed to communicate.

For these reasons, it was decided to employ a static scheduling scheme. To enable this implementation, the positions of both the satellite as well as the ground stations have to be predicted. These predicted values can then be used to generate a schedule. The following sections describe the static scheduling algorithm and then explains the position prediction techniques used.

A disadvantage of an off-line schedule is that when a ground station has no data to communicate, the slot cannot be filled by another ground station that has data to communicate. This is not such an issue for remote monitoring applications. The reason is that all sensor ground stations are constantly collecting data, and therefore will always have data to send. If a ground station does not have data to send, that ground station is malfunctioning. It is important to see the difference between a remote monitoring system, which constantly produces data, and a user oriented system that does not always have data to transmit.

4.4 Scheduling algorithm

In order to create a static scheduling scheme, the order in which ground stations are detected must be known. The order in which ground stations are detected is determined by the position and velocity of both the satellite and the ground station. To be able to predict the order of detection, the positions of ground stations and the satellite, therefore, have to be predicted as well. The positions of ground stations can be predicted, as they move with the rotation of the Earth. The position of a satellite can also be predicted, as it has a known orbit.

The distances between the satellite and ground stations can then be calculated as vectors in time. These distances can be compared with a minimum communications distance, to determine times when the satellite will be within communications range. This then provides sets of times in which every ground station may communicate with the satellite. These times are start-stop tuples, previously defined as Communication Time Windows (CTWs). Every ground station has multiple CTWs, one for every

time the satellite passes within communications range.

It is important to note that in this stage, all the parameters of the link budget may be given to the satellite position prediction algorithm. The algorithm then uses the parameters, such as the maximum antenna steering angle and minimum communications distance, to calculate the size of the satellite footprint. After the size has been calculated, all link budget requirements have also been taken into account.

It is not possible for a dynamic scheme to take into account the length of the CTW. If scheduling is done in real-time with no prediction, there is no way to know when the satellite will be out of communications range. Taking CTW length into account, enables the system to determine which ground stations will not be able to complete their communications and schedule other ground stations that will.

The scheduling scheme implemented attempts to minimise a quantity called "exclusions" along with equalising the total amount of scheduled communications time. This is implemented using a selection process described below. This selection process is executed when multiple ground stations are concurrently able to communicate. Exclusions are the number of ground stations currently able to communicate, but not allowed to, because of another station being scheduled.

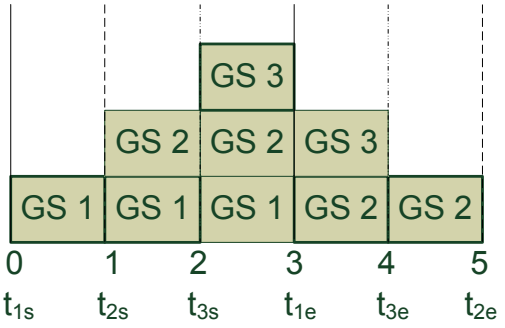


Figure 4.1: Example of a stream of ground stations able to communicate with the satellite at different times, where each ground station is in view for a different amount of time and also possesses a different required communications time.

For example, Figure 4.1 shows three ground stations able to communicate at various times. The start of the CTW of ground station g_i is indicated by t_{is} and the end by t_{ie}. g_1 has a required communications time of $\tau_1 = 2$ and a CTW from 0 to 3. g_2 has a required communications time of $\tau_2 = 1$ and a CTW from 1 to 5. g_3 has a required communications time of $\tau_3 = 2$ and a CTW from 2 to 4. In the figure, five areas can be identified. These are $t_{1s} - t_{2s}$, $t_{2s} - t_{3s}$, $t_{3s} - t_{1e}$, $t_{1e} - t_{3e}$ and $t_{3e} - t_{2e}$. These areas along with the different ground stations able to communicate, are also shown in Figure 4.1.

To generate a schedule, one ground station must be selected within every area. Preemption is not considered an option, as every switch from one ground station to another would incur extra antenna steering overhead and added complexity to the lower communication layers. This would include requiring all lower communication

layers to save their states. This especially creates problems on the FPGA as buffer space is very limited in this layer.

Another reason for not considering preemption is that a ground station is typically expected to produce approximately 60 KB of data. This translates to a communications time of approximately 25,6 seconds for a communications bandwidth of 19200 bits per second. The time that will be saved when using preemption will be in the order of seconds. It is for this reason that the possible gain does not seem to outweigh the added implementation complexity.

At the start of the schedule, the ground station with the lowest start time is selected and scheduled. In this case, g_1. g_1 would start its communication at time $t = 0$ and will complete at time $t = 2$. At time $t = 2$, there are two ground stations that still have to communicate. If g_2 is chosen, it will complete at time $t = 2+1 = 3$. This will exclude ground station g_3 from communicating as $t + \tau_3 = 3 + 2 = 5 < t_{3e} = 4$. If g_3 is selected to communicate after g_1 and then g_2, both g_2 and g_3 will be able to complete their communications. This example illustrates the principle of exclusions and how they can be used to minimise the number of dropped ground stations.

Figure 4.2 shows a flow diagram of the scheduling algorithm. If there are unscheduled CTWs, all CTWs able to communicate at the current time, are selected. Stations not able to conclude their communications, as well as the previously scheduled station, are deselected. If no stations satisfy the selection criteria, the next station in time that does, is selected. In this case, the current time is set to the selected station's CTW start time. If there are selected stations, the number of exclusions are calculated for each station. The station with the minimum number of exclusions and the minimum amount of total scheduled time is selected. A selected station is added to the schedule list with the start time set to the current time and the stop time set to the stop time of the selected station's CTW.

The amount of time every ground station receives is equalised by also selecting for minimum amount of total scheduled communication time. From this criteria, priority scheduling may be implemented where the station's total scheduled time is weighed against its priority. This will allow the operator to assign more time to certain clients than others. This quality of service mechanism is possible, because of the centralised control of the satellite link.

The end time of the previously scheduled station is set to the start time of the currently scheduled station, if the currently scheduled station's start time is less than the end time of the previously scheduled station. The end time is determined in this manner, as it is not possible to know when the next station will start before it has been scheduled. The maximum time is then first chosen as an end time and if the next station is scheduled before the previous station's maximum end time, then the previous station's end time is adjusted.

The scheduled station is then tagged as the current station and the scheduled station's required communication time is added to the current time to obtain the new current time. This is how the scheduler moves forward in time. After this has been done, the scheduler repeats the unscheduled CTW check.

The generated schedule is uploaded to the satellite from where it drives the communications system as explained in chapter 5. The schedule generator was implemented as a C program and is explained in Section 6.7.

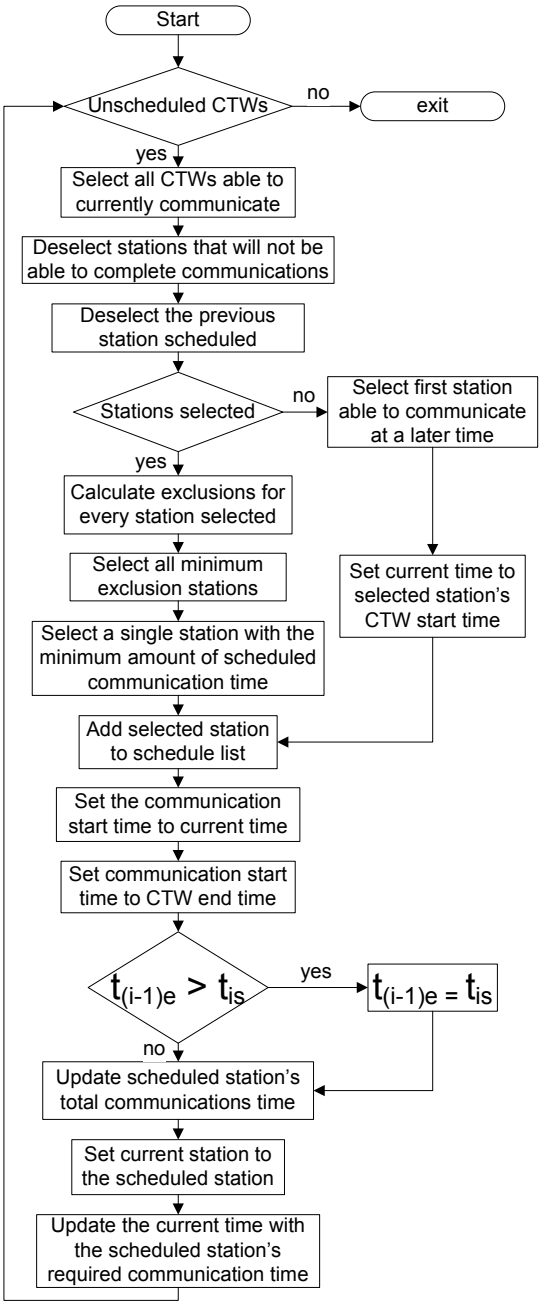


Figure 4.2: Flow diagram depicting the scheduling algorithm used to produce a schedule of ground stations.

4.5 Satellite position prediction

To calculate the distance between the satellite and ground station, the positions of these objects must first be known in time. Satellite orbit prediction is made very complex by orbit perturbations, which the satellite experiences as it travels along its track. Types of perturbations include: the oblateness of the Earth, atmospheric drag, the Earth's irregular gravitational field, solar radiation pressure, the attraction of the Moon and Sun, and motor thrust [46].

Currently, there are algorithms that may be used to predict the orbit of a satellite forward in time and take into account the various forces which affect the satellite's orbit. For satellites with orbital periods of less than 225 min, this includes LEO satellites, the SGP4 propagator algorithm is mostly used [47]. The algorithm may be considered accurate to within 1 km in position, but the degree of accuracy decays as the TLE used, ages.

The TLE is a set of values that fully describe the satellite's measured orbit at a certain date. TLEs can be downloaded from a satellite tracking organisation like Spacetrack [48]. These TLE's are input into NASA's SGP4 satellite position prediction algorithm, to predict the satellite position forward in time [47]. The TLE for a satellite is kept up to date by actively tracking the satellite and updating the TLE as the measured orbit parameters change.

For the prediction results shown in this chapter, a custom satellite prediction algorithm is implemented. It is important to note that although a less accurate algorithm is used for predicting the satellite path, this prediction merely generates a satellite position vector which could just as easily have been generated by the SGP4 algorithm. What is important, is how these vectors are *used* to predict the link quality over time. The reason for implementing a custom satellite propagator is to simplify the experimentation and data gathering processes.

The implemented algorithm models a satellite orbit with an eccentricity of zero. That is to say, a perfectly circular orbit. The satellite position vector ($\mathbf{S}$) is a $3 \times K$ matrix, where K is the total number of iterations given by $K = t_{\text{end}}/t_{\text{step}}$. Where t_{end} is the end time, in seconds, of the prediction and t_{step} is the step time. The step time determines the resolution of the prediction and is used to make all position vectors temporally meaningful.

The satellite orbit is created by using an initial point within the zy plane, denoted by $\mathbf{s}_0$, and rotating that point about the x axis. The static rotation matrix is

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Delta\beta) & -\sin(\Delta\beta) \\ 0 & \sin(\Delta\beta) & \cos(\Delta\beta) \end{bmatrix}. \tag{4.5.1}$$

This matrix applies a rotation around the x-axis to the point with which it is multiplied [49]. $\Delta\beta$ is the incremental angle by which the satellite vector is rotated every time step and is expressed as follows:

$$\Delta\beta = 2\pi \frac{t_{\text{step}}}{T_S} \tag{4.5.2}$$

where T_S is the period of the satellite. The satellite period is given by Equation (3.2.3) in Section 3.2.

The initial uninclined satellite position vector is then given by $\vec{S}_{0k} = \mathbf{Q}\vec{S}_{0(k-1)} = \mathbf{Q}^k\mathbf{s}_0$. The recursive form of the equation simplifies computation as the rotation matrix is then static.

To create an inclined satellite orbit, another vector rotation is performed about the y axis. The rotation matrix is

$$\mathbf{L} = \begin{bmatrix} \sin(i) & 0 & -\cos(i) \\ 0 & 1 & 0 \\ \cos(i) & 0 & \sin(i) \end{bmatrix} \tag{4.5.3}$$

where i is the angle of inclination as measured from the equator (xy plane) [49]. The inclined satellite position vector is then $\mathbf{S}_i = \mathbf{L}\mathbf{S}_0$. After $\mathbf{S}_i$ has been calculated, the satellite orbit must still be positioned about the z axis. The z axis rotation matrix is

$$\mathbf{H} = \begin{bmatrix} \cos\omega & -\sin\omega & 0 \\ \sin\omega & \cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5.4}$$

where ω is the positive angle from the y axis in the xy plane [49]. The satellite position vector then becomes $\mathbf{S} = \mathbf{H}\mathbf{S}_i = \mathbf{H}\mathbf{L}\mathbf{S}_0$.

4.6 Ground station position prediction

The ground station position vector ($\mathbf{G}$) generation process is similar to the satellite position vector, except that the ground station is rotated about the z axis. The ground station experiences a rotation about the z axis, because the equator is defined to be in the xy plane. The z axis rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \cos(\Delta\sigma) & -\sin(\Delta\sigma) & 0 \\ \sin(\Delta\sigma) & \cos(\Delta\sigma) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.6.1}$$

Similar to $\mathbf{S}$, $\mathbf{G}$ is a $3 \times K$ matrix. Where $\Delta\sigma$ is the incremental angle by which the ground station vector is rotated and

$$\Delta\sigma = 2\pi\frac{t_{\text{step}}}{T_G} \tag{4.6.2}$$

Where $T_G = 86\,164{,}098\,903\,691\,\text{s} = 23{,}93\,\text{h}$ is the period of one Earth stellar day and thus also the ground station rotation period [50]. The ground station position vector can now be recursively defined as $\vec{G}_k = \mathbf{R}\vec{G}_{k-1} = \mathbf{R}^k\mathbf{g}_0$, where $\mathbf{g}_0$ is the starting position of the ground station.

To generate the satellite position vector, the starting point $\mathbf{g}_0$ must first be chosen for the ground station. The starting point is chosen to be the Cartesian coordinates of the ground station, with $y = 0$ the universal prime meridian of the geodetic system, i.e. where the longitude is zero.

To position a ground station in a Cartesian coordinate system, a conversion from latitude and longitude points to xyz coordinates is required. Such a conversion should take into account the fact the the Earth is not a perfect sphere. The conversion

is done by viewing the Earth, not as a sphere, but as an ellipsoid. The x, y and z coordinates are [51]

$$g_{0x} = (R_{\text{trans}} + h_s)\cos\vartheta\cos\varphi \tag{4.6.3}$$
$$g_{0y} = (R_{\text{trans}} + h_s)\cos\vartheta\sin\varphi \tag{4.6.4}$$
$$g_{0z} = ((1-\varepsilon^2)R_{\text{trans}} + h_s)\sin\vartheta \tag{4.6.5}$$

where ϑ is the latitude, φ is the longitude both in radians and h_s is the height of the ground station above sea level. R_{trans} is the transverse radius of curvature and ε^2 is the eccentricity of the Earth ellipsoid where

$$R_{\text{trans}} = \frac{a}{\sqrt{1-\varepsilon^2\sin^2\vartheta}} \tag{4.6.6}$$
$$\varepsilon^2 = \frac{a^2-b^2}{a^2} \tag{4.6.7}$$

where $a = 6378{,}137$ km is the semi-major axis and $b = 6356{,}752$ km the semi-minor axis of the Earth ellipsoid as given by the WGS84 geodetic system [23].

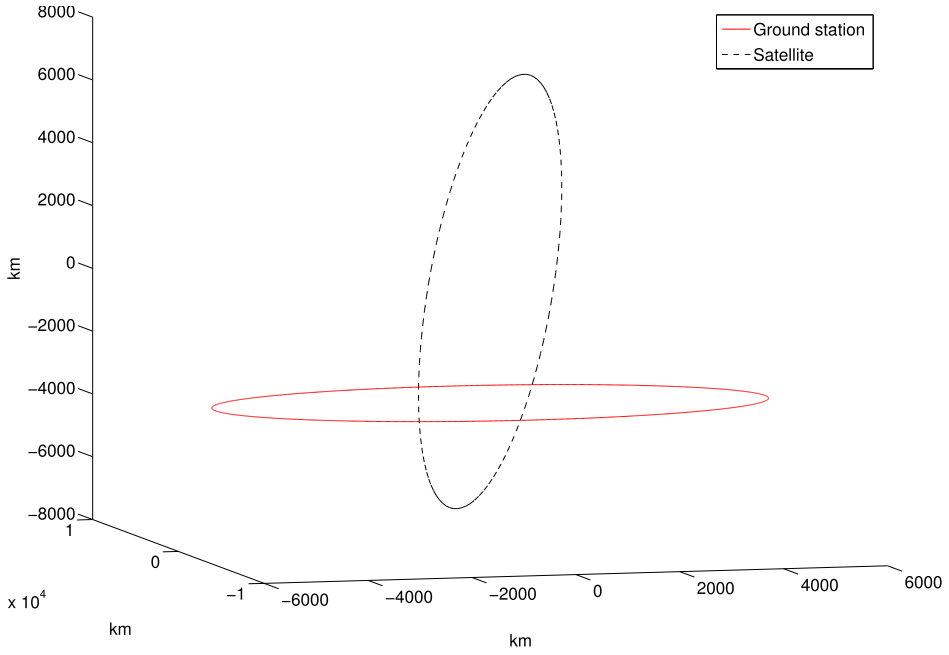


Figure 4.3: Satellite orbit, and Stellenbosch ground station moving with the rotation of the Earth.

Figure 4.3 shows the satellite orbit, as well as one ground station rotating on the surface of the Earth. It was generated using a satellite altitude of $h = 520$ km and an inclination of 96,5°, which is typical for a sun-synchronous LEO satellite [8]. With a mean Earth radius of $R_E = 6371$ km [23], the satellite period is calculated from (3.2.3) as $T_S = 5697$ s. The ground station is positioned in Stellenbosch, with a latitude of $\vartheta = -33{,}92°$ and longitude of $\varphi = 18{,}86°$ and a height above sea level

of $h_s = 111$ m. From (4.6.7), $\varepsilon^2 = 0{,}0067$ and from (4.6.6), $R_{\text{trans}} = 6384{,}7955$ km. This gives

$$\mathbf{g}_0 = \begin{bmatrix} 5013{,}848\,\text{km} \\ 1712{,}7142\,\text{km} \\ -3539{,}1484\,\text{km} \end{bmatrix} \text{ and } \mathbf{s}_0 = \begin{bmatrix} 0 \\ 0 \\ 6891\,\text{km} \end{bmatrix}$$

from (4.6.3), (4.6.4) and (4.6.5), where $\mathbf{s}_0$ was chosen to be the point in the satellite's orbit on the z axis. The satellite's angle about the z axis was chosen as $\omega = 0$ radians.

4.7 Distance prediction

The kth distance between the satellite and a ground station is

$$\begin{aligned} d_k &= \|\vec{GS}_k\| \\ &= \sqrt{gs_{kx}^2 + gs_{ky}^2 + gs_{kz}^2} \text{ where} \qquad (4.7.1) \\ \vec{GS}_k &= \vec{S}_k - \vec{G}_k. \qquad (4.7.2) \end{aligned}$$

Figure 4.4 shows a diagram of the vectors used in this section.

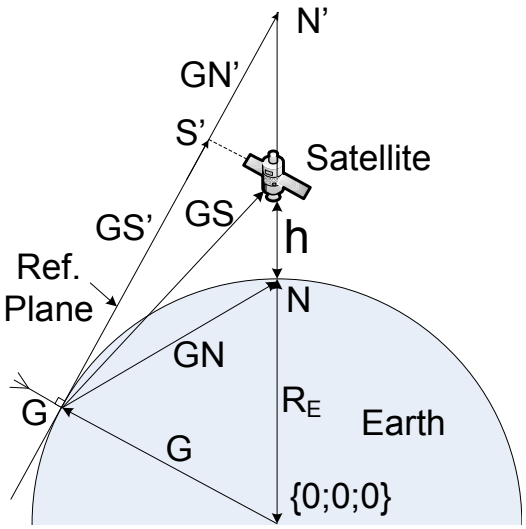


Figure 4.4: Diagram showing satellite, ground station and reference vectors.

The distance vector may be mapped to three perspectives. The first is the temporal perspective, where k is interpreted as a number of time steps. $\mathbf{d}$ can thus be plotted as a function of time, where $t = k \times t_{\text{step}}$. The satellite and ground station perspectives are obtained by mapping the time scale to the orbital angles for the satellite, denoted by γ_{sat}, and ground station, denoted by γ_{gs}, where

$$\gamma_{\text{sat}} = t\frac{2\pi}{T_S} = (k \times t_{\text{step}})\frac{2\pi}{T_S} \qquad (4.7.3)$$

$$\gamma_{\text{gs}} = t\frac{2\pi}{T_G} = (k \times t_{\text{step}})\frac{2\pi}{T_G}. \qquad (4.7.4)$$

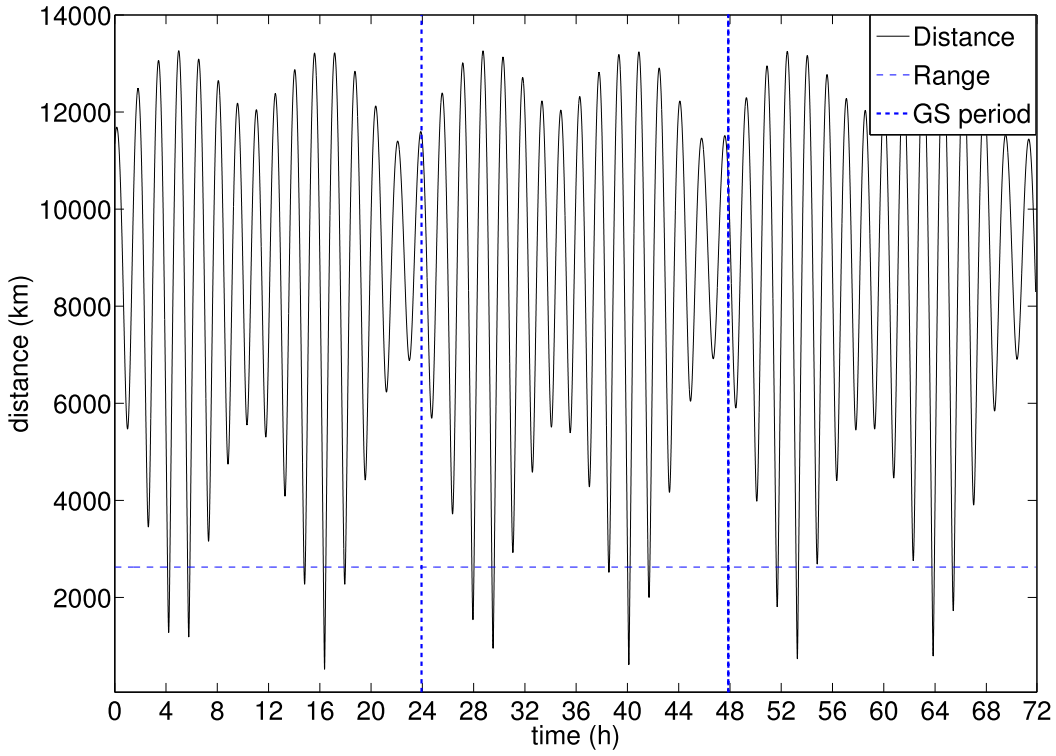


Figure 4.5: Graph showing the distance between the satellite and a ground station as a function of time as well as the calculated maximum visible communications range for a period of three days.

Statistic	Distance (km)
Maximum	13263
Minimum	519,5
Mean	8877
Median	9385
Standard deviation	3044

Table 4.1: Satellite-ground station distance statistics, generated by satellite visibility prediction.

Figure 4.5 shows the distance between the satellite and the ground station in Stellenbosch as a function of time over a period of three days. It also shows a horizontal line at the distance where the satellite is within visual range of the ground station. Visual range is $d_{\max} = 2626$ km, as calculated in Section 4.9. Two vertical lines are present at the times of one and two stellar days. The figure shows the periodicity of the distance vector.

Table 4.7 shows another simulation run performed for a period of two months and a time step of one minute. The statistics of the longer simulation are presented here, for improved accuracy. The minimum distance is smaller than the satellite altitude, because of the height above sea-level of the ground station.

As seen in Figure 4.5, the satellite passes within range of the ground station

four to fives times per day. One set during the day and the other during the night. Two types of pass sets can occur, one where there are two medium quality passes on both sides of the ground station. The other type of pass set is three passes where the satellite passes over the ground station for one of the passes, and therefore has a high quality link and the other two passes are at the edges of visibility with a low quality. The same profile was observed in a satellite tracking system called Orbitron [52]. Orbitron does not predict the satellite-ground station distance over time, only the satellite position.

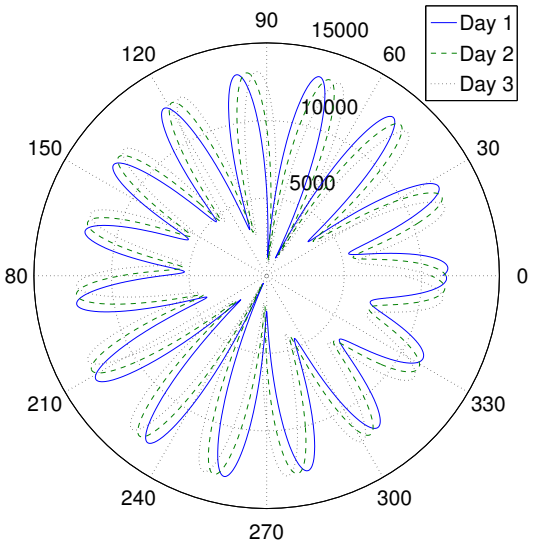
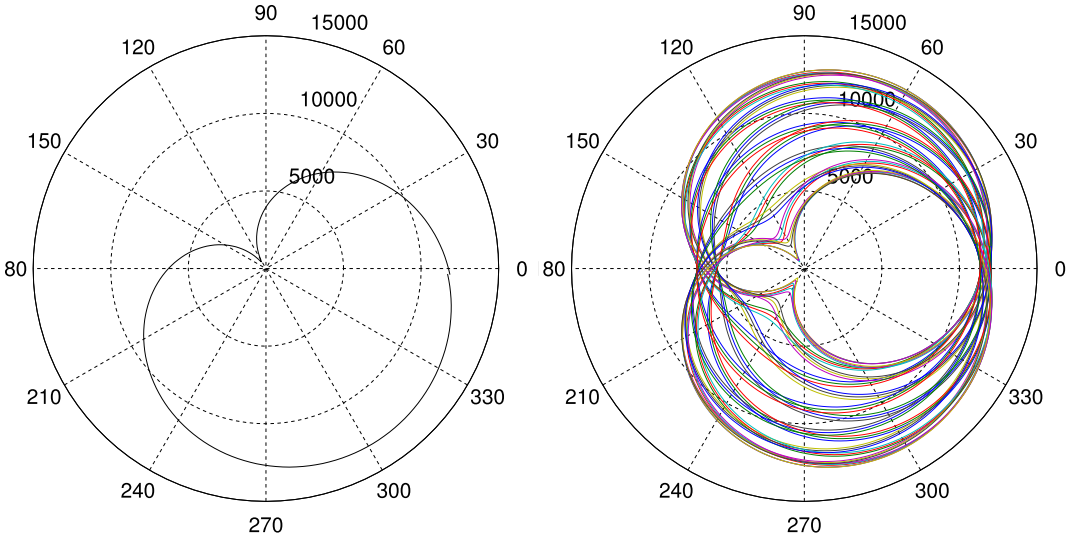


Figure 4.6: Satellite-ground station distance for three days from the ground station perspective.

Figure 4.6 shows the satellite-ground station distance from the ground station perspective for a period of three days. From this figure it is clear that the speed of the satellite is much greater than the speed of the ground station. The points closest to the origin are where the satellite passes over the ground station's track. This distance is a minimum when the ground station is underneath the satellite as it passes over.

Figure 4.7a shows the distance between the ground station and the satellite from the satellite's perspective for a single rotation where the satellite is directly over the ground station. As the satellite moves towards 125°, it is at its closest point to the ground station. This minimum distance may differ depending on where the ground station is in its rotational path.

Figure 4.7b shows the complete distance vector from the satellite perspective for the period of three days. What is depicted in the figure is essentially many versions of Figure 4.7a, swept over time as the satellite moves around the Earth. The angles should be interpreted as angles from the true North pole, as the satellite's initial position is on the z axis, and therefore where $\gamma_{\text{sat}} = 0°$. The two minimum points are then recognised to be in the Southern hemisphere which is as expected.



(a) Single pass (b) Three days

Figure 4.7: Satellite-ground station distance over time from the satellite perspective.

4.8 Angle prediction

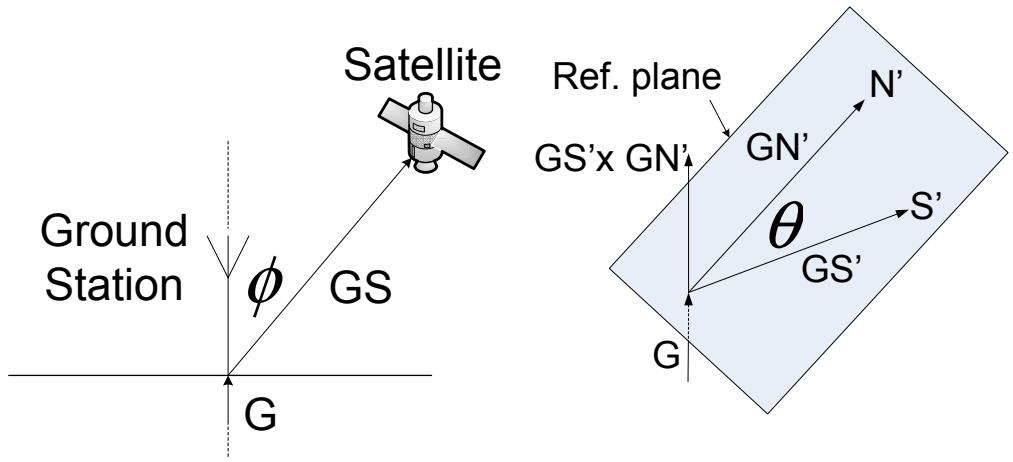
An angle prediction is required to calculate the optimal angle in which a ground station antenna should be pointed, to achieve maximum volumetric throughput of the complete satellite communication system. With the goal of positioning a ground station antenna in mind, two complimentary reference angles, vertical and horizontal, are selected to enable a complete specification of the direction of the satellite from the ground station.

To calculate the vertical angle, denoted by ϕ, two vectors are required. The first vector is the vector from the Earth's centre to the ground station given by $\vec{CG}_k = \vec{G}_k - \vec{C}_k = \vec{G}_k$, where $\vec{C}_k = \mathbf{0}$ as the Earth's centre is the origin of the coordinate system. The second vector is the vector from the ground station to the satellite given by $\vec{GS}_k$. For an observer standing at the ground station, the angle between the two vectors would be the angle the satellite makes with an upright antenna on the ground station. Figure 4.8a shows the vertical reference angle with the vectors required to calculate it.

To obtain the angle between the two vectors, the dot product of the vectors may be used to obtain

$$\phi_k = \arccos\left(\frac{\vec{GS}_k \cdot \vec{G}_k}{(\|\vec{GS}_k\|)(\|\vec{G}_k\|)}\right). \tag{4.8.1}$$

Figure 4.9 shows the calculated vertical angle from the ground station perspective. The ground station perspective is used to present the angle calculations as the goal is to position an antenna on the ground station. It is apparent that this figure is very similar to Figure 4.6. Intuitively, this is expected as the vertical satellite angle will degrease as the satellite nears the ground station and increase as the satellite moves further away from it. The vertical angle is zero where the satellite passes directly over the ground station. The vertical angle has a range of $0° \leq \phi_k \leq 180°$.



(a) Vertical reference angle (b) Horizontal reference angle

Figure 4.8: Satellite-ground station reference vectors and angles, used for angle prediction.

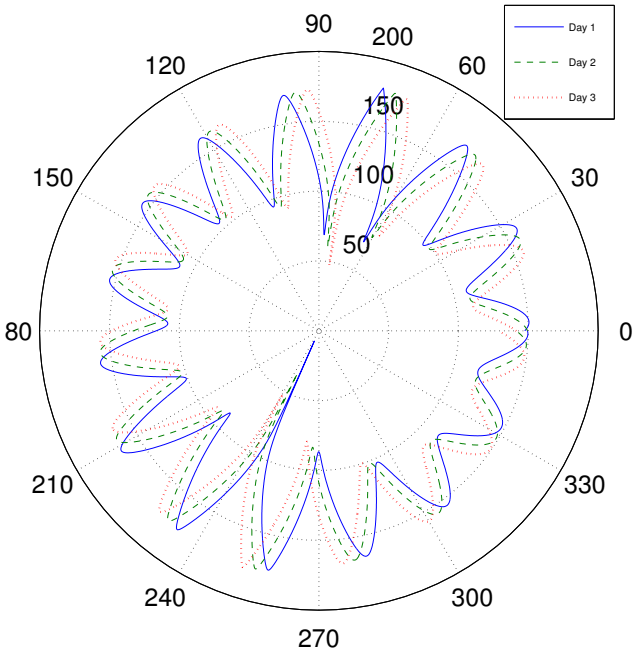


Figure 4.9: Vertical angle between ground station and satellite from the ground station perspective.

The angle between the satellite and true North is chosen as the horizontal angle. When an angle is measured from North, it is measured in a plane tangential to the point on Earth where the observer is standing. This is true when using a compass as a compass can only measure the angle to North in the plane it is positioned in. The compass mentioned here is a corrected compass, GPS compass or Gyroscopic compass, which points to true North. Therefore, in order to calculate the horizontal angle, the two vectors between which the angle will be calculated, must lie in the reference plane tangential to the point on Earth where the ground station is located.

In order to obtain these vectors, the vector from the ground station to North must be projected onto the reference plane, as well as the vector from the ground station to the satellite, see Figure 4.4. The angle can then be calculated in the same way as the vertical angle is calculated. Figure 4.8b shows the horizontal angle and the projected vectors required to calculate it in the reference plane.

The projected vector from the ground station to North is given by

$$\begin{aligned} \vec{GN'_k} &= \vec{G}_k - \vec{N'} \text{ where} \\ \vec{G}_k &= g_{kx}\hat{x} + g_{ky}\hat{y} + g_{kz}\hat{z} \text{ and} \\ \vec{N'} &= n'_z\hat{z} \text{ then} \\ \vec{GN'_k} &= -(g_{kx})\hat{x} - (g_{ky})\hat{y} + (n'_z - g_{kz})\hat{z} \end{aligned} \tag{4.8.2}$$

where N' is a point on the z axis and $\vec{GN'}$ is a vector in the tangential plane. The position of N' may be calculated by recognising the the vector will be at a 90° angle to the normal vector from the centre of the Earth. For perpendicular vectors their dot product is zero which gives

$$\vec{GN'_k} \cdot \vec{G}_k = 0 \tag{4.8.3}$$

and by then using the definition of the dot product

$$\begin{aligned} g_{kx}(-g_{kx}) + g_{ky}(-g_{ky}) + g_{kz}(n'_z - g_{kz}) &= 0 \\ -g_{kx}^2 - g_{ky}^2 - g_{kz}^2 + g_{kz}n'_z &= 0 \\ \frac{g_{kx}^2 + g_{ky}^2 + g_{kz}^2}{g_{kz}} &= n'_z \end{aligned} \tag{4.8.4}$$

By substituting 4.8.4 into 4.8.2, the equation becomes

$$\vec{GN'_k} = -g_{kx}\hat{x} - g_{ky}\hat{y} + \left(\frac{g_{kx}^2 + g_{ky}^2}{g_{kz}}\right)\hat{z}. \tag{4.8.5}$$

This shows that the vector is only dependant on the position of the ground station. As the ground station moves with the Earth, so will the direction of the reference vector change.

To project $\vec{GS}_k$ onto the plane, the satellite position as a point in $\mathbb{R}^3$ must first be projected onto the plane using [53]

$$\vec{S}'_k = \vec{S}_k - \frac{(\vec{GS}_k \cdot \vec{G}_k)\vec{G}_k}{\|\vec{G}_k\|^2}. \tag{4.8.6}$$

By substituting $\vec{S}_k = \vec{S}'_k$ into (4.7.2), the projected vector from the ground station to the satellite in the plane becomes

$$\begin{aligned} \vec{GS}'_k &= \vec{S}'_k - \vec{G}_k \\ &= \vec{S}_k - \vec{G}_k - \frac{(\vec{GS}_k \cdot \vec{G}_k)\vec{G}_k}{\|\vec{G}_k\|^2} \\ &= \vec{GS}_k - \frac{(\vec{GS}_k \cdot \vec{G}_k)\vec{G}_k}{\|\vec{G}_k\|^2}. \end{aligned} \tag{4.8.7}$$

The angle between $\vec{GS}'_k$ and $\vec{GN}'_k$ is calculated similarly to that of (4.8.1) where

$$\theta_k = \arccos\left(\frac{\vec{GS}'_k \cdot \vec{GN}'_k}{\left(\|\vec{GS}'_k\|\right)\left(\|\vec{GN}'_k\|\right)}\right). \tag{4.8.8}$$

The range of the horizontal angle is $0° \leq \theta_k \leq 180°$. A method is required to distinguish degrees left from North from degrees right from North. This is done by using the cross product where

$$\alpha_k = \vec{G}_k \cdot (\vec{GS}'_k \times \vec{GN}'_k) \text{ is } \begin{cases} > 0 \text{ when } \theta_k \text{ is right of North} \\ < 0 \text{ when } \theta_k \text{ is left of North} \end{cases} \tag{4.8.9}$$

The equation uses the fact that the cross product generates a third vector that is perpendicular to the plane containing the two vector. This plane is also the reference plane which has a normal vector from the Earth's centre through the ground station. Applying the cross product then either generates a vector in the same direction as the normal vector of the reference plane, denoted by **G**, or it generates a vector in the opposite direction to the normal vector. Applying the dot product to the vector generated by the cross product and the reference plane normal vector, a measure of the magnitude of the one vector i.t.o. the other is obtained. This value is positive if the vectors are in the same direction and negative if the vectors are in the opposite direction.

The horizontal angle may now be correctly represented as positive for right of North and negative for left of North. Figure 4.10 shows the horizontal angle as a function of time. This angle changes quickly and the real value lies in determining what the angle is when the satellite is within communications range.

4.9 Link quality and visibility prediction

An important aspect of a satellite communication system is to know how much time every ground station will have to communicate. The communication system can then be optimised to provide maximum volumetric data throughput through the system. The predictions should therefore show no predicted time greater than 11,8 min. The system comprises the satellite as well as all ground stations. The time ranges when a ground station can communicate is called its communication time windows (CTWs).

To determine CTWs for the generated distance vector, a distance is chosen corresponding to the maximum reliable communication range. This can be the visible

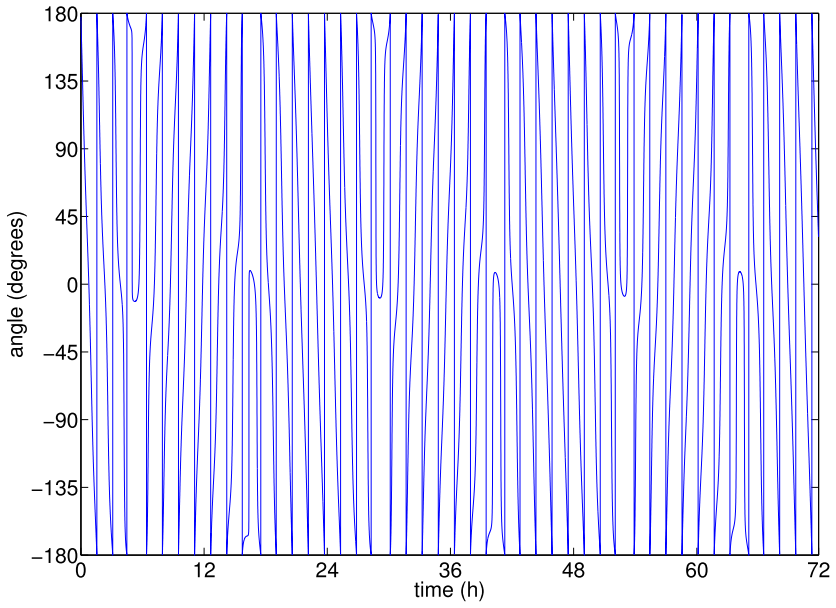


Figure 4.10: Horizontal angle from the ground station to the satellite as a function of time.

communications range, or can also take into account maximum antenna steering angle, as well as other link budget aspects. For the results shown in this section, the maximum distance is chosen as the visual range calculated using Equation (3.2.10).

In order to verify the performance of the visibility prediction algorithm, it was compared to commercial software, namely the Satellite Tool Kit (STK). STK is a software package that allows for the design and simulation of satellite communication systems. Initially, a scenario is created, to which objects can be added. Two objects were added for this simulation: a satellite and facility. The facility was placed at the Stellenbosch location. The satellite was created with a circular orbit, at an altitude of 520 km and an inclination of 96,5°, which matches the orbit characteristics specified Section 3.2. Two orbit propagators were tested: the J4Perturbation propagator and the SGP4 propagator [47].

The SGP4 propagator was briefly described in Section 4.5. The J4Perturbation propagator only takes into account the oblateness of the Earth, when predicting satellite movement.

A custom propagator was implemented in Matlab as described in Section 6.3. The implemented propagator shows a maximum communications time of 700 s, a minimum communications time of 40 s, a mean communications time of 537 s and a median communications time of 590 s with a standard deviation of 163 s. The satellite passes within communications range of the ground station an average of 4,75 times per day. The total average communication time is 42,5 min per day.

It should be noted that the maximum communications time of 700 s = 11,7 min is indeed lower than the maximum communications time of 11,8 min as calculated using Equation (3.2.12). The reason the predicted time is lower than the theoretical maximum, is because the satellite did not pass directly over the ground station for

	Custom propagator (s)	J4Perturbation (s) (%)	SGP4 (s) (%)
Minimum duration	40	59 (47,5)	12 (70)
Maximum duration	700	705 (0,71)	709 (1,29)
Mean duration	537	556 (3,54)	555 (3,35)

Table 4.2: Comparison of minimum, maximum and mean communication times, as predicted by the implemented propagator, the J4Perturbation propagator and the SGP4 propagator.

the simulated period. This was was the scenario used to calculate the maximum theoretical time in Equation (3.2.12).

The access times for the Stellenbosch ground station and the satellite were calculated for a period of two months. The results from all propagators are summarised in Table 4.9, along with a percentage deviation from the implemented propagator.

Table 4.9 shows the minimum, maximum and mean communication times of all three propagators tested. The percentage deviation from the custom propagator is shown in parenthesis, next to the values for the other two propagators.

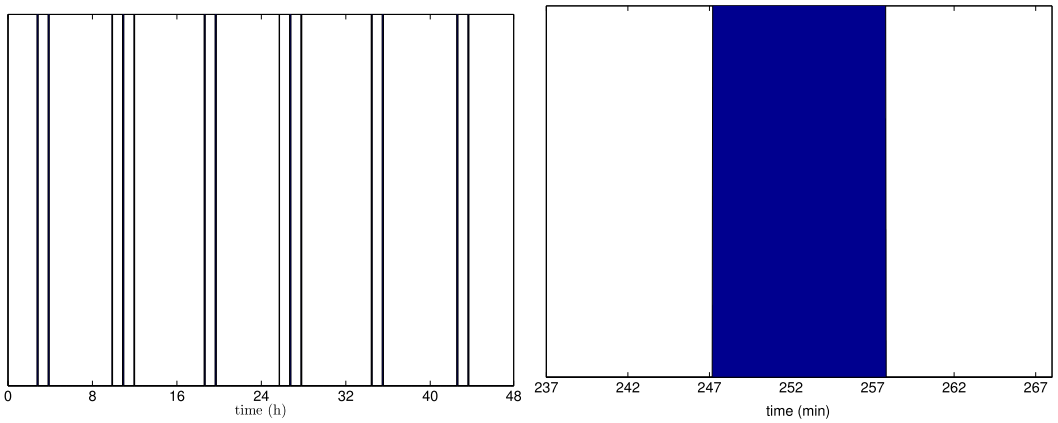
The table shows a high deviation for the minimum predicted times, 47,5 % and 70 % respectively. This high percentage deviation is partly due to the low number being compared against. Even though there exists a high percentage deviation, the deviation is only 19 s and -28 s respectively. On average, this difference only constitutes $(28 - 19)/(2 \times 708) = 0,636\%$ of the maximum theoretical CTW length. The difference is therefore not significant when compared to the possible length of a CTW and also not comparable to the average length of a CTW.

All maximum values match closely: 0,71 % and 1,29 % respectively. This is because of the theoretical boundary of 708 s, when the satellite moves directly over the ground station. All three values approach this theoretical maximum value. The inaccuracy of the custom propagator is due to the use of a step time length of 10 s, which establishes the maximum temporal prediction resolution.

The comparison of the mean communications time is the metric that best presents an estimate of the accuracy of the implemented propagator. When comparing the mean communication times, the custom propagator compared very well to the other two propagators with a deviation of only 3,54 % and 3,35 % respectively.

Table 4.9 showed the fitness of the custom propagator to drive the communications system. When program tracking is performed, very high accuracies are required, to enable the ground station tracking antenna to effectively communicate with the satellite. Such high accuracies are, however, not required for the satellite scheduler, as this does not adversely affect the satellite's ability to communicate. A variation in predicted times of a few seconds will not significantly influence the calculated schedule.

Figure 4.11a shows a graphical representation of all the CTWs for the three day Stellenbosch system. It is apparent that the CTWs are small compared to the total system time, showing the importance of optimising LEO satellite system communication times. Figure 4.11b shows the first CTW of the ground station. The CTW has a length of 620 s. From the mean communications time of 537 s, it is apparent that this is an above average quality link.



(a) Complete three day prediction (b) Enlarged view of first CTW

Figure 4.11: Communication time windows (CTWs) of a ground station

Link quality may be inferred by the time the ground station is within communications range. Links with higher quality have longer communication times. The link budget of the satellite system can now also be predicted for every ground station where the power loss as a function of distance and also time can be more accurately modelled.

In order to decrease ground station costs, a non-steerable ground station antenna can now be used, given a sufficient link margin. The goal is then to optimise the angle in which the antenna points, for every ground station in the communications system. The optimal angle for a ground station can be found as the mean of the predicted angles during the time when the satellite is within communications range. As seen from Figure 4.10, the horizontal angle quickly sweeps from 0° to 180° during one satellite pass. To determine the optimal angle to use, the angle at which the satellite spends most of its time must be calculated. This calculated angle will also change from one pass to the next and so a best fit must be chosen to select the optimal horizontal angle.

4.10 Maximising volumetric throughput

The goal of this chapter is to present techniques that increase the volumetric data throughput of the satellite communications system. Volumetric data throughput is an important concept in communications system engineering. In this work, it is defined as the total data amount that may be transported over the communication links in the system per unit time. When volumetric throughput is increased, the overall amount of data transported in the communications system is increased. From a system perspective, it is desirable to have the maximum possible amount of volumetric throughput. Although this quantity is very important from a systems perspective, it does not hold much importance with end users of single nodes.

It is possible for a system to have an increased volumetric throughput with some users having a lower average throughput rate. Priority scheduling schemes may have this effect on a system. By improving volumetric throughput, the average data

transfer rate of the system is improved and so the average data transfer rate of an average system user is increased.

The implemented communications scheme not only increases the volumetric throughput of the system, but also increases the average amount of data any user may transfer. This is done by eliminating station discovery times and efficiently scheduling only those stations having sufficient time to communicate. Stations allowed to communicate, but not able to perform productive work, because of short communications times, lower the overall system throughput.

The solution was to not allow these stations to communicate, thereby reducing the total time that the station may communicate for, but increasing the total useful transfers completed. The reduction in average wastage, therefore, increases the efficiency of the system.

The reduction is made possible by the centralised network access control implemented in the SCSS, while making use of the steerable antenna. It should be noted that the system is not limited to a satellite system with a steerable antenna, but can be implemented on any system where the access times of all entities can be predicted. If ground stations have steerable antennas, a schedule may still be centrally calculated and the data disseminated to all ground stations from where they can contact the satellite at the optimal times. The only factor in such a configuration is synchronising ground station times. Synchronisation will prevent issues of ground stations contacting the satellite at sub-optimal times and possibly contacting satellite at the same time.

4.11 Conclusion

In this section, the issue of link acquisition was discussed in detail. The reasons for moving towards a centralised network control were discussed. The main reason was to enable the administrator to maximise the total data throughput in the system. The methods employed to implement this move were then discussed. The methods employed involved satellite and ground station position predictions to enable the calculation of the distance between the satellite and the ground station. This distance was then used, along with the calculated visibility range, to predict the times the satellite will be able to communicate for.

Communication statistics were calculated and compared to STK, to determine the fitness of the visibility prediction algorithm implemented. The implemented predictor values were found to compare well to those predicted by the STK.

The visibility prediction allows for a schedule to be created that optimally schedules ground stations for communications. The angle between the satellite and the ground station as a function of time was also predicted. The reason for this was to improve satellite links that are too weak to sustain communications, while sacrificing part of the total ground station communications time. Finally, volumetric data throughput was discussed and the proposed increase thereof.

This chapter describes the driving force behind the communications system design as presented in Chapter 5. It lays the foundation on which to build the system. In Chapter 5 it can be seen that the calculated schedule drives the system and that the system cannot function correctly without such a schedule.

Chapter 5

Communication System Design

5.1 Introduction

In this section the design of the SCSS is described. The system was designed by making use of the relevant satellite communications knowledge, as described in Chapter 2, and applying that knowledge to the satellite under development as described in Chapter 3. Specifically, the steerable antenna used on the satellite enables a greater measure of satellite autonomy by means of a schedule, as specified in Chapter 4.

The software system is, therefore, driven by a schedule that determines when each ground station may communicate, and also specifies the duration of the communications time. The SCSS coordinates all high-level communications with ground stations. It initiates connections, steers the receive antenna, maintains the communications schedule, stores all files uploaded from ground stations and handles all high-level ground station requests. The SCSS is defined by all the software components required to enable ground stations to upload and download file data to and from the satellite.

Section 5.2 discusses the functional considerations taken into account for the SCSS design. The section presents the SCSS as a usable entity, and specifies how another entity would use the SCSS. Section 5.3 presents a high level domain model of the complete SCSS. It discusses what modules the SCSS consists of, what the important external interfaces are, and how all the SCSS components interact. Section 5.4 describes the different file formats used for storing and transporting data and discusses why these formats were selected. Section 5.5 discusses the structure of the file-store and explains how the other systems use the file store to maintain the SCSS and support ground station to ground station communications. Section 5.6 describes the station server and how it coordinates the different components of the SCSS to provide high-level communications capabilities to ground stations. Section 5.7 describes the station handler and its role in handling all ground station requests and how it interfaces with the file store. This section also discusses the challenges of real-time deadline driven programming and presents the solutions implemented. Section 5.8 presents all messages used by the SCSS and explains how each message is handled. It also shows an example of each message handled. Section 5.9 elaborates on the importance of effective logging mechanisms and discusses how these were integrated into the overall SCSS design.

5.2 Functional overview

For a functional overview, a Unified Modelling Language (UML) use-case diagram of the SCSS is shown in Figure 5.1. The tasks the SCSS are able to perform are:

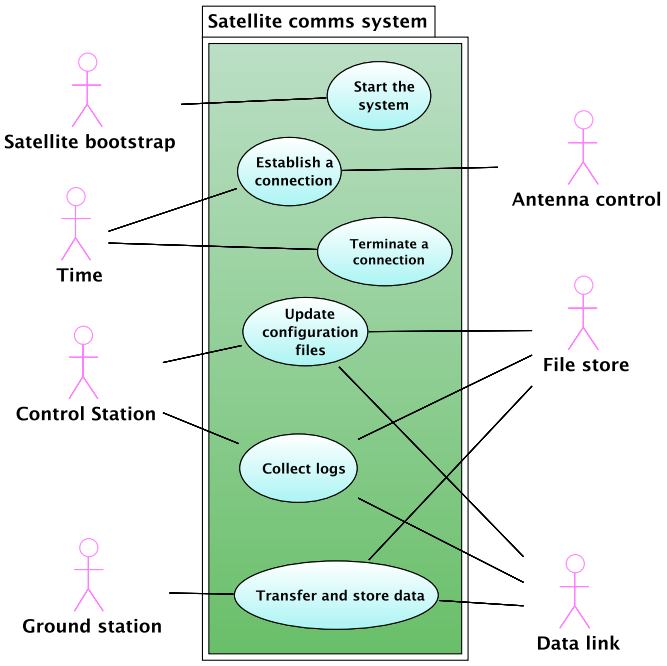


Figure 5.1: UML use-case diagram of the SCSS

1. Start the system.
2. Establish a connection.
3. Terminate a connection.
4. Update configuration files.
5. Collect logs.
6. Transfer and store data.

The satellite bootstrap scripts will start the SCSS when the satellite is within communications range of the first ground station. Ground station communication activations and terminations are activated at certain times, according to when the satellite is in communications range of a ground station. Configuration updates can be uploaded, and log downloads can be requested, from special ground stations called “control stations”. These stations have more permissions than normal “client stations”. Control and client stations may transfer and receive files to and from other stations. To establish a connection, the external antenna control interface is

used to steer the antenna towards a ground station. All data transfers occur to and from the file-store and over the data-link in the application layer. All systems are developed to fulfil these functional requirements and testing focuses on ensuring that all functional requirements are fully met.

It is important to specify the relationship between the satellite and the ground stations. More specifically, it is important to specify which entity controls the communications. As the system is a store-and-forward system, where the satellite provides a service (storage space) and the ground stations use this service, the method of interaction fits into a client-server communications model. In this case, the satellite is the server, as it is providing the service of storage space and the ground stations are the clients that use this service. It is according to this relationship that the SCSS is developed.

5.3 High level domain model

Figure 5.2 shows a high level UML domain model of the complete SCSS along with all external interfaces. The external interfaces are the satellite bootstrapper, Real-time Clock (RTC), antenna control and the communications interface.

The SCSS consists of five parts: the file-store, schedule, station server, station handler and messages that are passed between the different entities. The entities shown in figure 5.2, are those interfaced with the different parts of the SCSS and use the external interfaces to perform the various system tasks. The station server, described in Section 5.6 and driven by the schedule, coordinates all communications with ground stations. The station handler, described in Section 5.7, handles all requests from a specific ground station. The file store stores all messages handled by the SCSS, data to be sent to other ground stations, as well as logs and SCSS configuration information, as set out in Section 5.5.

Messages are transferred on an application level in the form of Extensible Markup Language (XML) files. Messages are files uploaded from, and downloaded to, ground stations. When uploaded from ground stations, the messages are requests for service from the ground station; when messages are downloaded, they are responses to service requests. A complete description of the message structure and query types can be found in Section 5.8.

The interfaces from which outside input are received, are the bootstrapper and the RTC. When the satellite is out of range of all ground stations, the on-board systems are shut down to save power. The satellite bootstrapper's function is to start the SCSS before it is within range of the first ground station, which occurs once every orbit rotation. The RTC is used with the schedule to determine when a ground station should be scheduled.

The Antenna Control Interface is one of two output interfaces of the SCSS as described in Section 3.6. The station server uses this interface to steer the receive antenna in the direction of the ground station currently in communications range. The only parameters presented to the server are the ground station's latitude, longitude and height above sea level. The antenna control software, which is a separate program with which the station server interfaces, continuously adjusts the antenna beam direction over time, taking into account the current satellite position.

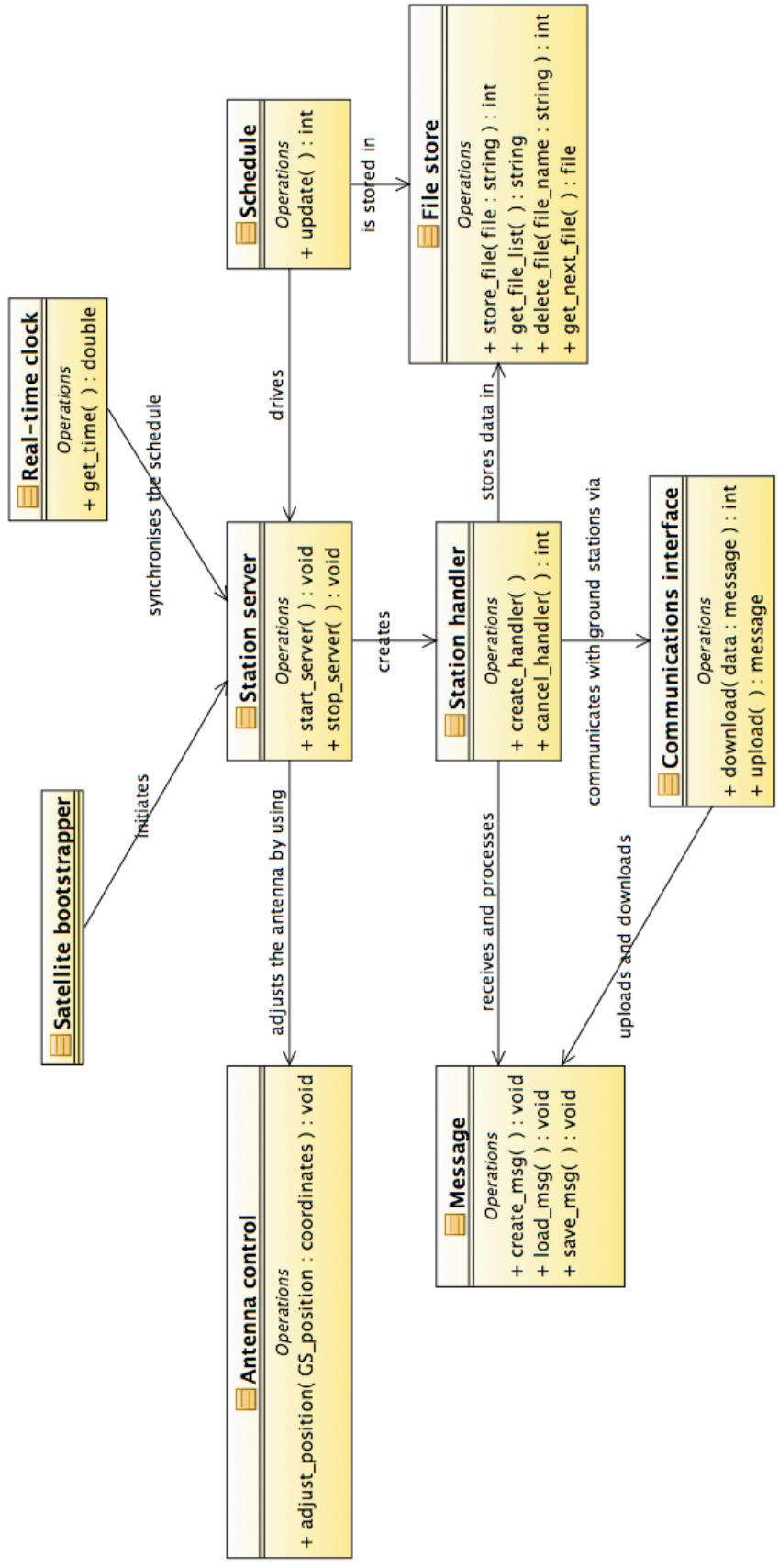


Figure 5.2: High level domain model of the SCSS, showing the main SCSS entities, external interfaces and operations that should be able to be executed on the entities.

The communications interface is capable of both output and input data. All data uploaded or downloaded travel through this interface. The interface consists of an Inter-process Communications (IPC) link between the station handler and the lower lever communications protocol, as described in Section 3.3. Ground stations upload messages to the SCSS via this interface and the satellite responds to these messages via this interface.

The schedule drives the communications system by specifying when which ground station should communicate. The schedule is the implementation of the theory described in Chapter 4. The schedule determines when each ground station can be contacted and for how long it may communicate. The off-line schedule is generated by the scheduler, described in Section 6.7, which computes the schedule by using the predicted position of all ground stations as well as the satellite.

5.4 File formats

This section presents the file formats used in all files located in the file store. The first format is that of the transmission file. It is a file format for transporting messages and is the format in which the ground station and the satellite communicates in the application layer. The second format is the schedule file format, used by the station server to determine when to contact a ground station. The third format is that of the station information file, which is used to store all station information relevant to the communications system. The fourth and final format is that of the system parameters file, which is used to define system limit values as well as file and folder names.

5.4.1 Transmission file

Two text-based serialisation file formats were investigated for transporting messages. These were Extensible Markup Language (XML) [54] and JavaScript Object Notation (JSON) [55]. XML being the more well known format, it has been extensively used in data communications systems in various fields of application. Some well known examples include: Amazon Web Services (AWS) [56], the Financial Information Exchange (FIX) [57], the GPS Exchange Format (GPX) [58], Adobe Flex [59], Really Simple Syndication (RSS) 2.0 [60], Scalable Vector Graphics (SVG) [61] and the Extensible Messaging and Presence Protocol (XMPP) [62]. JSON is a light-weight file format for transmitting structured data over a network connection. For this purpose, JSON output is made available by both Yahoo [63] and Google [64] as an alternative to XML formatted data. JSON is mainly used in Ajax web application programming [65].

Both of these file formats are completely text-based. The benefit of this is that all messages between the ground station and the satellite are human readable. Text based file formats promote extensibility and encapsulation of data and simplify testing and debugging [66]. There are two issues with text-based messages: the first is that they are less dense than binary data representations. Since link capacity on satellites is limited, it is important to find a solution supporting maintainability as well as efficiency. The solution is to compress all messages transmitted over the

Listing 5.1: XML standard definition schema

```
<xs:element name="stationdata">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="sourceid" type="xs:integer"/>
      <xs:element name="destid" type="xs:integer"/>
      <xs:element name="querytype" type="xs:integer"/>
      <xs:element name="file_name" type="xs:string"/>
      <xs:element name="file_data" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

link. Text-based messages with compression allows for both maintainability as well as efficiency.

The second issue is that the data section of the messages must also be text-based, which can be a severe limitation to a general data store-and-forward system. The solution is to use Multipurpose Internet Mail Extensions (MIME) Content Transfer Encoding, which allows binary files to be converted to text and inserted into a text-based messaging system [67]. The encoding scheme used is called base64 [68]. It is packaged as a simple program that reads a file name as an argument and outputs the text-converted data to the standard output, which can be piped into a new file.

XML is a powerful file format, in that it allows for complexly nested structures in the document. The disadvantage is that this usually makes XML parsing libraries large, which has the program using more memory at run-time. For an embedded system this is a significant disadvantage, as explained in Section 6.4 . As an example, the well known libxml2 library [69] (version 2.6.32) is 1.3 MB in size. There are, however, libraries available that are more light-weight with less features. These libraries can be used when only simple XML parsing and generation is required. One such library is the *expat* library [70] (68 KB), which is also supported by QNX and distributed with the operating system.

JSON on the other hand is a simple format, not supporting all the features of XML. Since the file format requirements of the SCSS are not complex, JSON would be sufficient. JSON is also somewhat smaller than the expat library, at 41 KB for the "Yet Another JSON Library" (YAJL) [71] JSON parser and generator written in C. XML was chosen as the communications file format mostly because the expat library is already integrated into the QNX operating system, which simplifies generating and parsing XML in the QNX environment and because users who have to use the log file will more likely be familiar with XML. These advantages were seen to outweigh the advantage of the 27 KB that would be saved.

The XML file contains the following fields: source address, destination address, query type, file name and file data. All files queued for transmission must conform to the XML Standard Definition (XSD) schema [54] as shown in listing 5.1.

The source address is the address of the entity where the message originated from. The destination address is the address of the entity to which the message is sent. The

query type field specifies what type of message is being transferred. The receiving entity determines how to handle the data section from the query type. The file name is used as the unique identifier when the file is saved after a file transfer is done. The file data section contains the data of the file being transmitted, converted into a string value using base64 encoding as described earlier. This element is, therefore, a string type and not a binary data type, as an XML file cannot contain binary data.

The file format is also designed in such a way that there are no required fields. The reason is that there are some messages where some of the fields have no meaning. For example, the request to download all log files on the system does not require a file name to be sent to the satellite, as the satellite downloads all logs to the requesting ground station.

5.4.2 Schedule file

The format in which the schedule is presented is also a text format, but in this case a Comma-separated Values (CSV) format is used. This format works well for short records of tabular data starting with a name [66]. In this format, it is important that the delimiter not occur within the range of valid values. This was achieved by having the station name only consist of alphanumeric characters and the time decimals separated by full stops.

```
(station name:string[alphanumeric]),(start time:double),(stop time:double)
```

The string shown above shows the format of one record of the file. The station name is specified first, followed by a comma, followed by a start time and stop time also separated by a comma. The station name is an alphanumeric string as previously stated and the times are of type double. The schedule can contain any number of these records.

5.4.3 Station information file

The station information file containing the information of all ground stations has a Space-separated Value format (SSV) shown below, where all fields are on the same line. The reason for choosing this format is the same as for the CSV format in Section 5.4.2. The fields contained in the file are the station name, Station Identifier (ID), Required Communications Time (RCT), permissions, latitude and longitude.

```
(station name:string[alphanumeric]) (station ID:integer) (RCT:double)
  (permissions:integer) (latitude:double) (longitude:double) (height:double)
```

The station name is the same name as defined in the schedule file in Section 5.4.2. The station ID is the numeric identifier of the station. Both an ID and a string name are used in this system to support logging and debugging functionality. Whenever log files are written, the station names are used to make the files easy to read. Whenever a message is sent between the satellite and ground stations, the numeric station ID is used in the source and destination ID fields, to reduce the amount of data transmitted. The required communications time is a field used by the scheduler, to determine how much time to assign to each ground station. How the required communications time is used can be seen in Section 4.4. The station permissions field defines whether a station is a control station, a normal client station or an

emergency station. This is used to determine what types of requests are allowed from every ground station, as presented in Section 5.8. The latitude, longitude and height specify the position of the ground station on the surface of the Earth. This information is used by the station server to steer the satellite antenna, as set out in Section 5.6.

5.4.4 System parameters file

All system parameters are recorded in a name-value text format in the system parameters file. The format of this file, along with the recorded parameters and their values during testing, is shown below:

```
String_length_max 30
File_name_max 50
Path_length_max 30
Type_length_max 3

ARQ_retry_time 1
ARQ_retries_max 3

GS_NAMES "GS_names.dat"
CONFIG_FILENAME "config.dat"
SCHEDULE_FILENAME "schedule.csv"
NEW_SCHEDULE_FILENAME "new_schedule.csv"
GS_RANGES_EXT ".ctw"

MAILBOX_ROOT "mailboxes"
CONFIG_ROOT "config"
INCOMING_ROOT "incoming"
OUTGOING_ROOT "outgoing"
LOGS_ROOT "logs"
```

The first group of parameters is the limit definitions. These parameters define maximum values for the system, mostly with regards to string lengths. `String_length_max` is used as the maximum length of one schedule record and also the maximum ground station name length. `File_name_max` is the maximum length of a file name. `Path_length_max` is the maximum length of the string describing the path to a file. `Type_length_max` describes the maximum string length of the query type parameter explained in Section 5.4.1. This length is required when generating the string for the command acknowledge response, as per Section 5.8.5.

The second group of parameters is the ARQ retry values. `ARQ_retry_time` gives the time in seconds, which the SCSS will wait before retrying a download after a failure response from ARQ. `ARQ_retries_max` gives the number of times the SCSS will retry a download after receiving failure responses from ARQ. These variables are used in the send retry mechanism (Section 5.7.1).

The third group of parameters is the file names group. This group specifies the names of the ground station information file, the system parameters file and the current and new schedule files. It also specifies the extension of the CTW files created by the scheduler, which are used to generate the schedule (Section 6.3).

The fourth and final group of parameters give the folder names of the different parts of the file store (Section 5.5). They are, in order: the names of the mailboxes

folder, system configuration folder, incoming messages folder, outgoing messages folder and logs folder.

5.5 File store

The file store is the area on the satellite where all data are permanently stored. The file store is a well defined folder structure where data are stored in the form of files. The complete folder structure of the file store is shown in Figure 5.3.

The file store contains the station mailboxes, configuration files, logs, incoming messages and outgoing messages. The configuration files are the station information file, the schedule file and the system parameters file (Section 5.4). The SCSS logs all runtime execution and these logs are cycled out on a weekly basis and stored in the logs folder. This is essential for debugging purposes and to ensure the system is running as intended, as discussed in Section 5.9. The transport protocol stores all received messages in the incoming folder and the SCSS stores all outgoing messages in the outgoing folder, before these messages are passed down to the lower transport layer for transmission.

When ground stations perform upload queries, these queries apply to a certain ground station's mailbox. The station mailboxes are where all uploaded files are stored. A station may add files to any mailbox of any ground station registered on the system, but may only download files from its own mailbox, as described in Section 5.8. Every station has its own mailbox folder where files may be sent. These folders are contained in the root mailbox folder.

5.6 Station server

The station server coordinates all communications with ground stations. This includes initiating a connection between the satellite and a ground station, steering the receive antenna to point to the scheduled station and creating a station handler to handle all ground station requests. Coordination is done by monitoring the calculated schedule to establish when a station handler for a specific station should be created and when that handler should be cancelled. When there are multiple communications channels, multiple station handlers may be created as separate threads. This will allow concurrent ground station communications on an application layer.

The following sections make use of flow diagrams to illustrate program flow. Before this is done, some symbols are defined in Figure 5.4a. The first symbol, the elongated circle, is a terminator. This symbol is used to define the entry and exit points of all flow diagrams. The second symbol, the rectangle, is a process. It shows actions taken or instructions executed. The third symbol is a predefined process, which is a process later also defined by a flow diagram. The fourth symbol, the diamond, is a decision. It contains a logical expression with multiple possible outcomes. The fifth symbol is a non-standard symbol, created to show a process with a return value, and therefore multiple possible outcomes. The return values are *success* or *failure*. Figure 5.4b shows an equivalent flow diagram for the symbol and also illustrates why the symbol was created. Most processes in the system have return values, which means that space would have been wasted if the equivalent flow

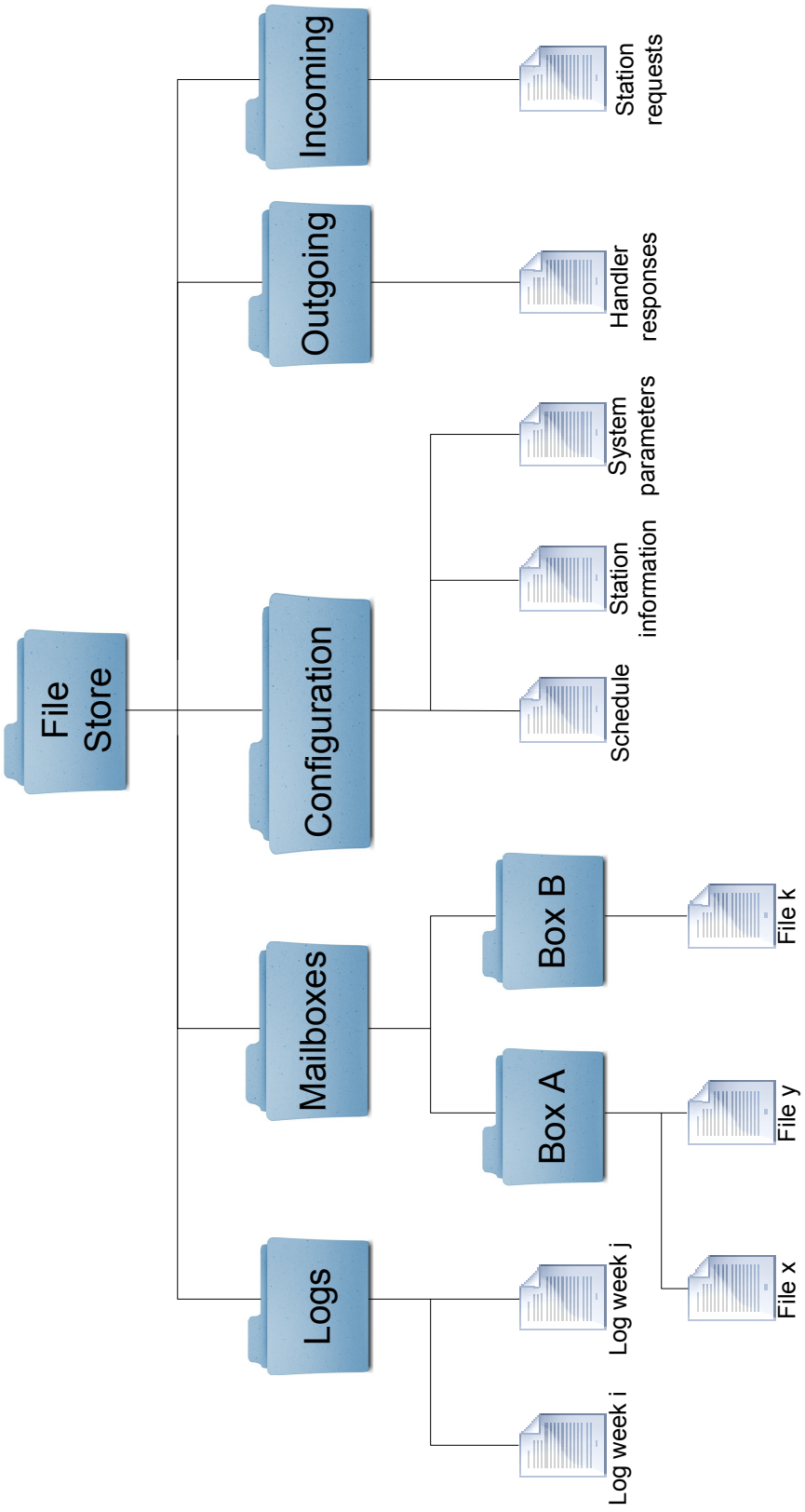
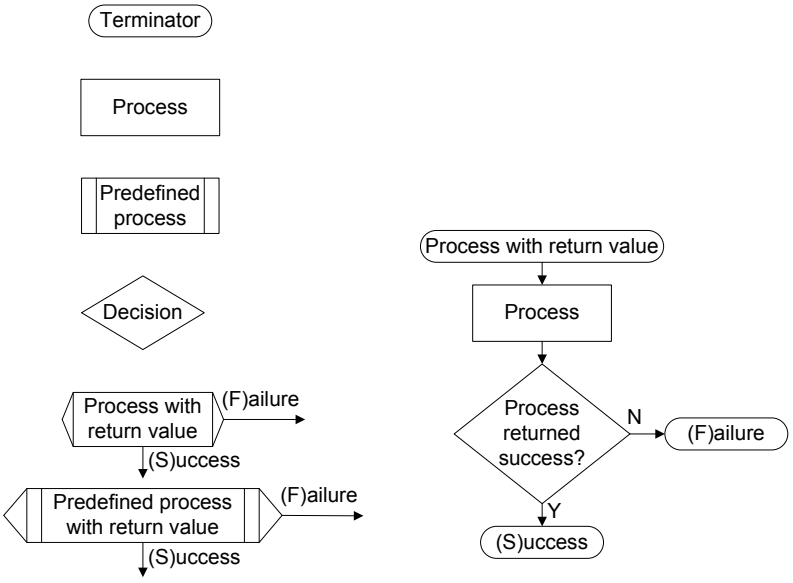


Figure 5.3: File store hierarchy, showing all files and folders present in the file store.



(a) Shape definitions

(b) Equivalent flow diagram of the process with multiple return values

Figure 5.4: Flow diagram definitions used

diagram was used. The sixth symbol represents a pre-defined process with a return value. The flow diagram for this process will necessarily have more than one return terminator.

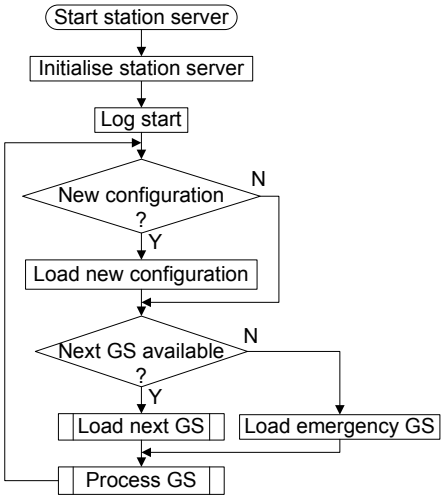


Figure 5.5: Flow diagram depicting the execution of the station server.

Figure 5.5 shows the high level flow diagram of the station server. The server initialises all global variables and logs its version information, name and start time after the station server is started by the on-board system. The next step is to check whether any new configuration files are available. Configuration files include the

ground station information file and the system parameters files (Section 5.4). The new versions are loaded if any of these files are available. The current files are loaded if no new files are available and it is the first time the server runs. This mechanism is required as the SCSS runs continuously. The check for new files is done every time a ground station has been processed, because any ground station could have changed a configuration file, which would require the updated file to be loaded. This cannot only happen after the SCSS initialises, otherwise configuration parameters will not be able to change in real-time as could be required.

The schedule item for the next ground station is loaded after a successful loading process. This process is described in Figure 5.6. The schedule item is processed by the server after it has been loaded. This process is explained in Figure 5.7. The new (also called next) schedule file is loaded, after which there are no more records in the schedule to process. This forms part of the schedule update mechanism explained in Section 5.6.1. The SCSS loads an emergency ground station, which activates the emergency communications mechanism if the next schedule cannot be found. This mechanism is described in Section 5.6.3 and enables the SCSS to receive a new schedule file and avoid critical failure. The SCSS continues with normal operation after a new schedule has been received.

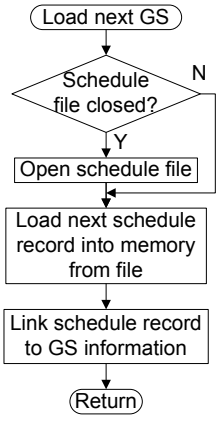


Figure 5.6: Flow diagram depicting the process of loading the next ground station.

Figure 5.6 depicts the process of loading the next ground station schedule record. First, a check is done to determine whether the schedule file is closed; if so, the file is opened. This forms part of the schedule update mechanism further described in Section 5.6.1. Secondly, the next record is loaded from the schedule file, described in Section 5.4.2. Only one record is loaded at any time to save memory, as described in Section 6.4. The record is linked with the station information it pertains to, after it has been loaded into memory. This simplifies look-up functions in the station handler.

Figure 5.7 illustrates the steps followed to process one ground station schedule record. Firstly, if the start time of the ground station (t_{start}) is less than the current time (t_{cur}), the station server sleeps for the difference of the two times. This is the mechanism enabling the server to block (sleep) for periods where there are no ground

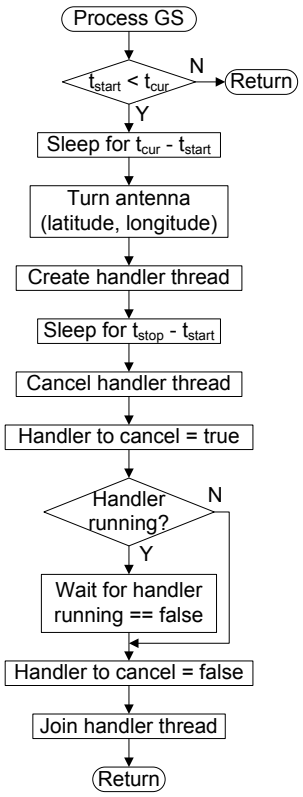


Figure 5.7: Flow diagram depicting the process of processing the loaded schedule record.

stations scheduled. Blocking often, when there is no work to be done, reduces the CPU usage of the SCSS, as described in Section 6.5. The next step, is to turn the antenna towards the ground station. As described in Section 3.6, the antenna steering interface consists of sending the latitude, longitude and height information of the ground station to the antenna control software. Thirdly, the station handler thread is created, which starts executing the *Start Handler* process, shown in Figure 5.9.

The server sleeps for the difference between the scheduled station stop time (t_{stop}) and start time (t_{start}) after the thread has been created. This is the amount of time for which the handler is allowed to communicate. The server sends a cancel command to the handler thread after the communication time expires. When threads are cancelled, care has to be taken to ensure that all the thread memory is freed and that all mutex locks are released. Section 5.7.2 describes how this is achieved. To ensure the handler thread has been cancelled and as part of the cancellation lock-step mechanism described in Section 5.6.2, a global variable *Handler to cancel* is set to true. The server then waits for the handler to terminate and resets the *Handler to cancel* variable to false after the handler has terminated. The server thread then joins with the handler thread to receive a return value from the thread. The exit status of the handler thread is logged.

5.6.1 Schedule update mechanism

Since the communications system is driven by a schedule, a mechanism is required to update it. Two schedule update methods are implemented, namely on-line and off-line update methods. The on-line method allows the schedule to be updated while the server is running.

In this method, a file with the same name as the current schedule is uploaded as a configuration upload query, as per Figure 5.14 on Page 74. This triggers the on-line schedule update mechanism. The station handler will then close the schedule file opened by the *Load next GS* process, illustrated in Figure 5.6. It will then replace the old schedule file with the new one.

The station server again performs the *Load next GS* process after the station handler completes its execution. The schedule file is opened again by the server and the first record is loaded. When the record is processed, as illustrated in Figure 5.7, all stations with start times less than the current time are ignored, and eventually the next scheduled station is loaded. From this point the schedule drives the server normally.

The on-line method can be used for corrections of the schedule. The old schedule can be replaced by the corrected schedule after new TLEs have been loaded and a corrected schedule has been generated on a control ground station. Without this method, one would have to wait for the schedule to complete before corrections could be made. The communications payload could be endangered if these changes are critical.

The off-line schedule update method is used when a schedule file with a new name is received from a ground station using the configuration upload command (see Figure 5.8.4). This file will only replace the current schedule file when the station server has processed all schedule records. When all schedule records have been processed, a check is done, as illustrated in Figure 5.5, to determine whether a new schedule should be loaded to replace the old schedule. The off-line schedule update method is used to ensure continuity of communications in the station server.

5.6.2 Cancellation lock-step mechanism

The cancellation lock-step mechanism ensures that no handler threads are created, without being cancelled, when the handler thread executes in a region where it has disabled cancellation. A thread will disable cancellation when it uses a library that is not cancellation safe.

The POSIX standard specifies some library functions that must be cancellation safe and others that may be cancellation safe, depending on the implementers [38]. Only functions specified as cancellation safe by the implementers of the library can be considered as such. All other libraries have to be protected from thread cancellation. These functions are protected by the cancellation lock-step mechanism.

The handler thread can turn off thread cancellation if it is executing in a volatile section where it may not be cancelled. Cancel commands sent to a thread while it has thread cancellation turned off, will not be delivered to that thread. That is to say, after the thread leaves the volatile section, a cancel signal from the station server would not be delivered to it, since thread cancellations are not queued as signals

are [72]. This creates the need for a mechanism that allows the station server to ensure that the handler thread was cancelled after it executed the cancel command.

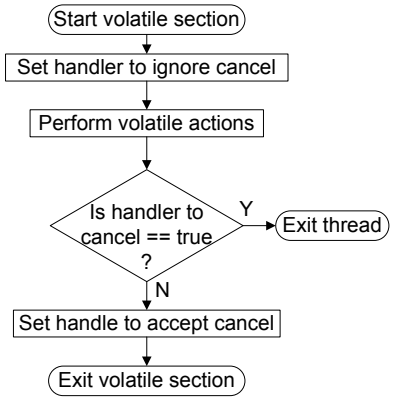


Figure 5.8: Flow diagram depicting the cancellation lock-step mechanism.

Figure 5.8 shows the station handler part of the cancellation lock-step mechanism. When the handler is executing in a section that is not cancellation safe, the handler ignores cancellation signals and performs the actions in the volatile section. The handler checks whether a cancellation was sent by the station server after all volatile actions have been performed. This is done by checking the "handler to cancel" flag. If the "handler to cancel" flag is set, the station server expects the handler to cancel and so the handler calls the thread exit function. This function terminates the thread after popping all the functions from the clean-up stack. The clean-up stack is used to ensure cancellation safety as described in Section 5.7.2. One of the functions pushed onto the clean-up stack is the handler clean-up function. When the clean-up function executes, it sets the "handler running" flag to false, to inform the server that it has cancelled and then signals the server thread to wake up. The handler again accepts cancellation signals and resumes normal operation if the "handler to cancel" flag is not set after a volatile section is completed.

In Figure 5.7, the server thread is executing in the "Process ground station" procedure. This function also shows the server part of the cancellation lock-step mechanism. The server sets the "handler to cancel" flag to true and waits for the "handler running" flag to be set to false after it calls for the station handler to cancel. The server is then unlocked by the station handler when it cancels. The station handler has then set the "handler running" flag to false and so the station server can continue to process the next ground station.

Without this mechanism, the station server would have called for the station handler to cancel, without knowing whether the handler did in fact cancel. The SCSS will continue to function, but every thread unsuccessfully cancelled uses extra memory, which is a precious commodity in embedded systems as further discussed in Section 6.4.

5.6.3 Emergency communications mechanism

The station server should always be actively attempting to communicate with new ground stations. This ensures maximum system communications time and so maximises the volumetric throughput of the complete satellite system. To enable the station server to continuously communicate, the server always requires an active schedule. An active schedule is a schedule containing a ground station, where the start time of the ground station is greater than the current time. In other words, a schedule with a ground station that can still be scheduled.

As no schedule can have an infinite length, the schedule update mechanism described in Section 5.6.1 is used to replace old schedules with new active schedules. New schedules should be added to the communications system at regular intervals to ensure the correct functionality of the SCSS. The duration of these intervals depends on the accuracy of the orbit propagator used. An acceptable length of time is usually every ten days for the SGP4 orbit propagator [47]. If, however, there is no new active schedule available, the emergency communications mechanism is used to prevent a critical failure of the SCSS.

As shown in Figure 5.5, if there is no next ground station to load, an emergency ground station is loaded. This is a ground station with special emergency station permissions, as specified in the station information file, as presented in 5.4.3. The start time of the emergency station is set to be the stop time of the last station in the schedule and the stop time is set to be ten minutes from the start time. This will have the satellite continuously polling the emergency ground station to attempt to establish a connection, and checking every ten minutes whether a new schedule has been uploaded. The purpose of the emergency ground station is, therefore, to upload an updated schedule to the satellite. Normal system functionality will return when a new schedule is uploaded, since the next ground station will be available for the server to schedule.

5.7 Station handler

The sole purpose of a station handler is to service requests received from the ground station to which it is bound. The station handler is cancelled by the station server when its communications time expires. The process of handling each request message, is described in more detail in Section 5.8. The start handler process is illustrated in Figure 5.9. This is the thread created by the station server when processing a ground station, as depicted in Figure 5.7.

Figure 5.9 depicts the start of the station handler. Firstly, the handler is initialised and the version number and start time are logged. Then the *Handler clean-up* function as well as the *Log handler stop* function is pushed onto the clean-up stack. The clean-up stack used to ensure handler thread cancellation safety is described in Section 5.7.2. The handler then establishes a communications link with the ground station, as shown in Figure 5.10.

The handler reads the first request from the ground station if the link was successfully established. This process is shown in Figure 5.11. The *Fetch query* function receives an expected type parameter, which is the type of message the system is expecting to receive. The expected parameter ensures early notification to the SCSS if

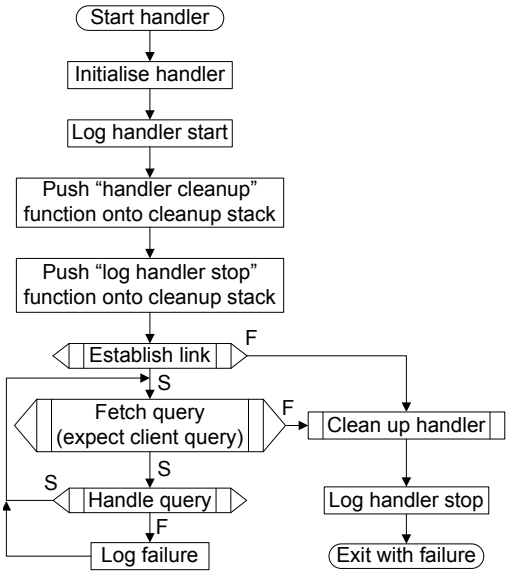


Figure 5.9: Flow diagram depicting the process of the start of the station handler.

the ground station loses synchronisation. At this step, the handler expects to receive any valid client query other than an activation acceptance. If the link establishment or query reception fails, a critical error is logged, the handler clean-up function is called and the handler exists returning a failure mode to the server.

The next step is to invoke the *Handle query* process shown in Figure 5.12, which determines what type of query was received, and then takes steps to handle the query appropriately. A query handling failure is logged and the SCSS then fetches the next query. A failure of the query handling process is not a critical failure of the system; as such, a failure can usually be attributed to incorrect information being received from the ground station.

It is worth mentioning at this stage that there seems to be no *Exit with success* path in the graph. The reason for this is that successful termination is only achieved after the station server cancels the handler, as shown in Figure 5.7. This is the point where the communications time of the handler has expired. The successful exit path is, therefore, situated in the handler clean-up function called by the clean-up stack and success is always returned by this function.

Figure 5.10 illustrates the process of establishing a communications link with a ground station. Firstly, it is ensured that the current time (t_{cur}) is smaller than the station stop time (t_{stop}). If this is not the case, the ground station is no longer allowed to communicate with the satellite and the function returns failure. The allowed communications time is calculated as the difference between t_{cur} and t_{stop} if the CTW has not yet expired. An activation offer is then sent for the calculated communications time. If the activation offer is successfully sent, a query is fetched with the expected type being an activation acceptance message. The "fetch query" process is shown in Figure 5.11. The same handle query process mentioned in Figure 5.9 and illustrated in Figure 5.12 is used to handle the activation acceptance.

Failure is returned if the activation offer is rejected and success is returned when

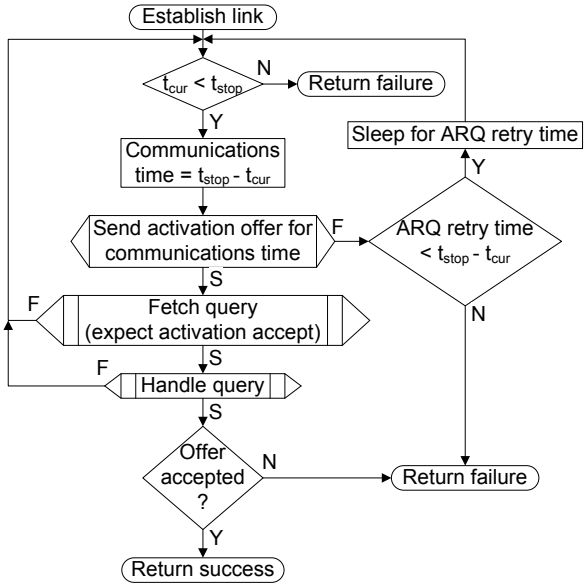


Figure 5.10: Flow diagram depicting the process of link establishment.

it is accepted. Failure is also returned if the query fetch or handling processes fail. If the previous send failed, but the *ARQ retry time* + t_{cur} is still before t_{stop}, the handler waits for *ARQ retry time* and then retries the process of sending the activation offer. Failure is returned if there is not enough time left.

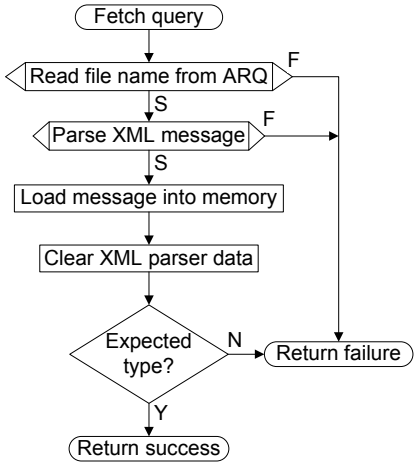


Figure 5.11: Flow diagram depicting process of fetching a query.

Figure 5.11 illustrates the process of fetching or uploading a query from the ground station. The first step is to read the file name from the lower level ARQ protocol. The station server sends a read request to the lower ARQ protocol and receives the file name of the file reconstructed by ARQ. The read blocks if no file is immediately available. The XML file is parsed in order to obtain a tree of data

elements if the read function succeeds. Each element is then stored in memory to be processed later by the handler function if the XML parsing process succeeds. The XML file data are cleared after the message has been loaded into memory. The process returns success if the received message type matches the expected message type. The process returns failure if the expected type does not match, or any of the previous functions failed.

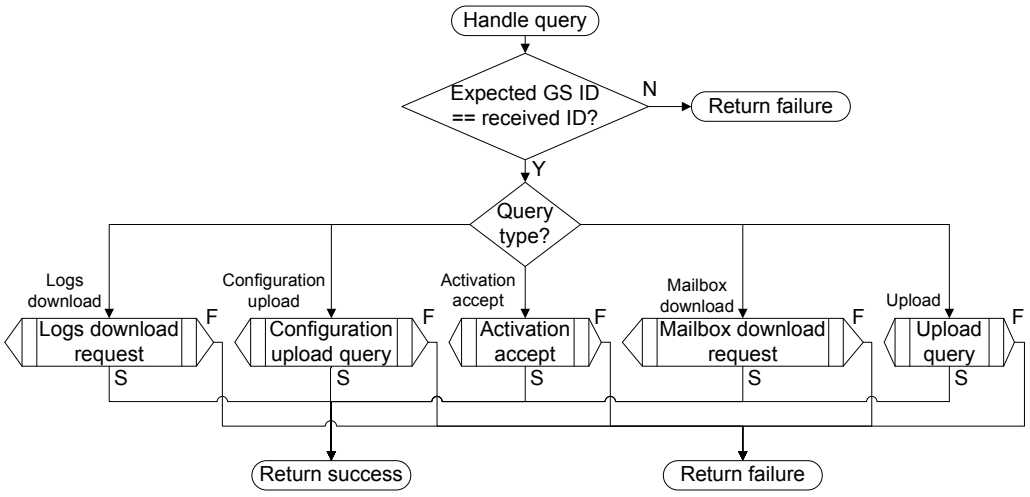


Figure 5.12: Flow diagram depicting the process of handling a query.

Figure 5.12 shows the process of handling an incoming client query. The first is a safeguard that checks whether the received station ID is the same as the expected station ID. It ensures that even if a station other than the one with which the connection was set up, manages to send a message to the satellite and that messages manages to pass through the ARQ level, it will not be processed by the satellite. These measures are taken, because the lower level protocols are separate projects. Inserting defensive programming structures into the SCSS, ensures that even if a failure of the lower level protocols occur, it will not lead to a failure of the complete communications system. In the worst case, it could lead to a graceful failure of the system, with all system errors logged. Failure is returned if the IDs do not match.

The query type is inspected to determine which handler function to use if the IDs match. Every query type has its own handler function, called by the handle query process. Each handler function can return with failure or success. The return value of the handle query function is the return value of the specific handler functions.

The process of handling every query, along with what fields are required, is explained in detail in Section 5.8. The message handling function is also designed with extensibility in mind. It uses the message handler structure to simplify the addition new messages. All that is required to add a new message type is to implement a message handler for that message type, add the type to the valid types list and add the message handler to the handle query function. Extensibility is an important part of software engineering as requirements might change in the future. Development costs are kept to a minimum for any future developments if a program is extensible [73].

5.7.1 Send retry mechanism

The send retry mechanism performs a transmission resend whenever a failure return value is received from the lower communications protocol layer. This mechanism is present in the processes of link establishment and file downloads, as shown in figures 5.10 and 5.15 respectively. This mechanism addresses transient loss of communications with a ground station. A transient loss of communications can occur when terrain is blocking the communications path. As the satellite moves at a low altitude, as described in Section 3.2, the communications channel is susceptible to being blocked by terrain. These circumstances can occur when the satellite first moves into view of the ground station and again right before the satellite moves out of view of the ground station as the elevation angles at these times are very low [74].

The send-retry mechanism is implemented as a measure to address these times of high probability of failure. If the satellite is at the start of the CTW, the elevation angle is increasing and the distance to the ground station is decreasing. Both factors improve the link quality over time and, thus, the probability of retransmission success of a file is also improved. If the ground station is nearing the end of its CTW, the link quality is decreasing, but retransmission of the file increases the chances of a successful transmission by the duplication of the transmission itself.

After every retransmission, the SCSS waits for a certain amount of time before retrying the transmission. This period is in the order of milliseconds, since the satellite moves at great velocities, as discussed in Section 3.2, which translates to a high rate of change of the satellite link.

After a maximum number of retries have been reached, the transmission is labelled as a failure and the transmission of the next file is attempted. When the station handler time expires, this cycle is also stopped.

5.7.2 Ensuring cancellation safety

In real-time systems, using strict deadline scheduling, a thread can be cancelled at any point in the code execution path. Strict deadline scheduling is implemented by the station server, which sleeps for the allowed station communications time and then cancels the handler thread after its communications time has expired, as illustrated in Figure 5.5. If proper measures are not used, allocated memory and locked mutexes can be lost. This leads to memory leaks and system instability. If memory leaks occur, the system's memory usage will increase without an increase in productivity. This will not only lead to the failure of the SCSS, but to the failure of all processes executing on the OBC, since none of the processes will have enough memory to function.

To ensure memory leaks cannot occur, the clean-up stack mechanism is used. This mechanism defines a set of functions assisting with the prevention of memory leaks and lost mutex locks. A function freeing memory is pushed onto the clean-up stack, after memory is allocated to a variable. An example code section is shown in listing 5.2.

After a data storage variable is created in line 1 and allocated memory in line 3, the memory free function is pushed onto the clean-up stack in line 4. The first parameter of the push function is the name of the function to execute and the second

Listing 5.2: Cancellation safe code section

```
1 char *data;
2
3 data = malloc(sizeof(char)*length);
4 pthread_cleanup_push((void *)free, (void *)data);
5
6 perform actions...
7
8 pthread_cleanup_pop(1);
```

parameter is the parameter sent to the executed function. The function name and parameter have to be cast to void types, since this is what the push function expects. Actions are then performed and, when the actions complete and the data variable is no longer required, the allocated memory may be freed. This is done by popping the previously pushed function from the clean-up stack. The pop function call in line 8 removes the function from the stack and executes it.

No memory is lost if the station server cancels the handler before line 3 or after line 8. That is to say, if the handler is cancelled before memory is allocated or after memory is freed, no memory can be lost. If the station server cancels the handler while it is performing actions, i.e. before it can release the allocated memory, and a clean-up stack is not used, the allocated memory is lost. The clean-up stack, however, automatically pops all functions in the sequence they were pushed onto the stack if a thread is cancelled. The free function is, therefore, popped if the server cancels the handler while it is performing actions. Use of the clean-up stack functions, therefore, ensures that no memory is lost if a thread is cancelled.

Two types of cancellation exists, namely asynchronous and deferred cancellation [72]. Asynchronous cancellation immediately cancels the thread being executed, which makes it difficult for implementers to protect against this type of cancellation. The effect of this is that most library functions are not asynchronous cancellation safe. The second type of cancellation, deferred cancellation, uses cancellation points in the code to define areas where it is safe to cancel a function. Some standard functions are also defined as natural cancellation points, for example the sleep function. The SCSS only uses deferred cancellation to enable it to better manage memory deallocation and better make use of deferred cancellation safe functions. The time that a cancellation is delayed in deferred cancellation, is in the order of milliseconds and does not adversely effect the functionality of the SCSS.

5.8 Message handling

There are eight query types, which can be further divided into client queries, command queries and handler responses. The client queries are general communication messages, while command queries are system set-up and logging commands. The handler responses are messages originating at the satellite, as a response to a ground station query. The client queries are: “Upload”, “Download request”, and “Activation acceptence”. The control queries are: “Retrieve logs” and “Configuration update”.

Listing 5.3: Activation offer transmission file

```
<stationdata>
    <destid>5</destid>
    <querytype>1</querytype>
    <file_data>6.3</file_data>
</stationdata>
```

The handler responses are: “Activation offer”, “Command acknowledge” and “Download”.

Control commands may only be generated by control stations. These are stations with special permissions to allow them to change configuration files and download system log files. The permissions are specified in the station information list, as per Section 5.4.3.

Whenever the satellite receives an unknown query type, an error file is written to the originating station’s mailbox. Whenever a file is received that does not match the specified XML format, that file is discarded and the event is logged by the Station handler, as described in Section 5.7.

The following section presents the different query types and specifies which transmission elements in the transmission file are required for the query to function correctly. It also illustrates the logical flow of every query to show which steps are taken to successfully handle every query.

5.8.1 Activation offer

An activation offer is sent by the station handler to a ground station to allow it to communicate with the satellite, as illustrated in Figure 5.10 on Page 67. The required fields in the transmission file are the destination ID, query type and the communications time inserted into the file data section. A new file is created with a recalculated communications time value if a ground station does not reply to an activation acceptance or the transmission fails.

The communications time is the time which the ground station is allowed to communicate for, when receiving this query. The time is calculated by subtracting the allowed communications time, given by the schedule, from the time that has passed, since the station was allowed to communicate. This gives the remaining allowed communications time, which is communicated to the ground station.

Listing 5.3 shows an activation offer transmission file, where 5 is the ID of the destination ground station, the query type of 1 corresponds to an activation offer and 6.3 is the allowed communications time in the file data section. Note that none of the other transmission file elements, described in Section 5.4.1, are present, since they are not required. This saves space when data are transmitted over the satellite link, and is permitted as discussed in Section 5.4.1.

5.8.2 Activation acceptance

The activation acceptance is received by the station handler as a response to an activation offer. Required elements are the source ID, query type and file data.

Listing 5.4: Activation acceptance transmission file

```
<stationdata>
        <sourceid>5</sourceid>
        <querytype>13</querytype>
        <file_data>ACCEPT</file_data>
</stationdata>
```

Listing 5.4 shows an activation acceptance file, where the source ID contains the ID of the ground station accepting the communications offer, the query type is the activation acceptance query type and the file data section should either contain the string "ACCEPT" or "REJECT".

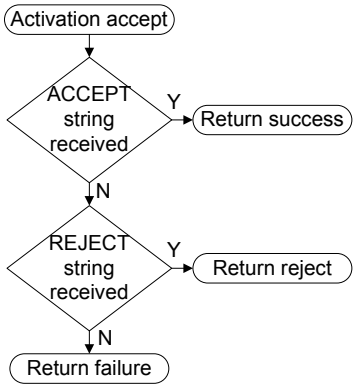


Figure 5.13: Flow diagram depicting the activation acceptance procedure.

Figure 5.13 shows the process of handling an activation offer. The file data section is checked and if the "ACCEPT" string is received, success is returned. Reject is returned if the "REJECT" string is received and failure is returned if unknown data are received.

5.8.3 Upload query

The upload query is received when one ground station has a file to send to another ground station. The required fields for this file are the source ID, destination ID, query type, file name and file data sections. Listing 5.5 shows an example transmission file of an upload query, where the source ID is the ID of the ground station which sent the file, the destination ID is the ID of the ground station to which the file must be sent, the query type is the upload request type, the file name contains the name of the file transmitted and the file data section contains the file itself. Note that the file data section shows a picture file (gif), which is converted to text using base64 conversion as explained in section 5.4.1. The original file is 598 bytes in size, while the base64 converted version is 816 bytes in size.

Listing 5.5: Upload query transmission file

```
<stationdata>
        <sourceid>2</sourceid>
        <destid>1</destid>
        <querytype>11</querytype>
        <file_name>satellite_icon.gif</file_name>
        <file_data>
R0lGODlhEAAQAOZwAP7+/klJSVFRUUdHR6urqzQ0NOjo6Nvb20JC
Qjo6Ojw8PD4+Pk1NTWhoaGJiYiwrKaGgoEhISBUVEPX19SwsLC8v
LR8gHFBQUM/Pz0FAPvLy8qWlpdbW1hYWFfn5+Tk5OVhYWJeXlzAw
MJqamj09Pf38/W1tbZmYmUhHRi0tLVNTU/38/jM0L9LS0nZ2dhcY
E+Dg4Ds7O15eXqGhoObm5kFBQWdnZ0JEQFRUVCYlIkRERC0tLB4f
GrGxsEhHSDY2NiEhHBEREcjIyXh5eZKSknFwbvHx8Ts6OCkoJ5CQ
kEtLSbCxr/39/UpKSvr6+nFxcURDQXh2eLGxsff3+KKipCUkHzEx
MURFRezt7TMzMCkqJdPT0zc3N1FQUPz8/L+/v5iYmltbWzY1NR8e
HiYmJi0uKJ2dnUZGRnx8fCgpKDo6O3Fzbzc3MqioqISEhP//////
/wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAACH5BAEAAHAALAAAAAAQABAAAAezgHCCg4SFhExSbmYw
Hm+Cb46DbyE/AgwDYT0lK2BUkYIOCAI4FzpcURBXQp9wBAsRAQwB
A2JII6xwBwUIJAoLalAPSgaFTgIfFCIFCRlZL0VehG8bMSpBCU0o
bCwWS58AbQ0ERFY7R0BaEmsAgwcOMk8NA2k5VWU8N1iDXy41KWcg
yIx5UKHDEFZvtpiwgcHIiS4+Zkwx9KYFmiTE4LAzJIhGAAUEOBbS
wOHNRpGCAEwQGQgAOw==</file_data>
</stationdata>
```

When a file upload request is received, the handler stores the file data section in the mailbox of the destination ground station under the specified file name. From there it can be downloaded to the destination ground station when that station performs a download request, as per Section 5.8.6.

5.8.4 Configuration upload command

Figure 5.14 shows the steps taken to process a configuration upload query. This query is very similar to the data upload query, except for permission and file name filtering and some added type checking to distinguish between off-line and on-line schedule updates.

Initially, the status variable is set to success. A check is then done to ensure that the ground station has the required permissions as specified in the ground station information file, defined in Section 5.4.3. If the required permissions are not met, the status variable is set to false and all other processing is skipped. The file name is then checked for validity if the permission check succeeded. Valid file names are any of the configuration file names, namely: `GS_names.dat`, `config.dat`, `schedule.csv` or `new_schedule.csv` as described in Section 5.4.4. If the file name is not valid and the rest of the processing is skipped, the status variable is set to failure.

If the received file is a schedule file, the message is recognised as a schedule

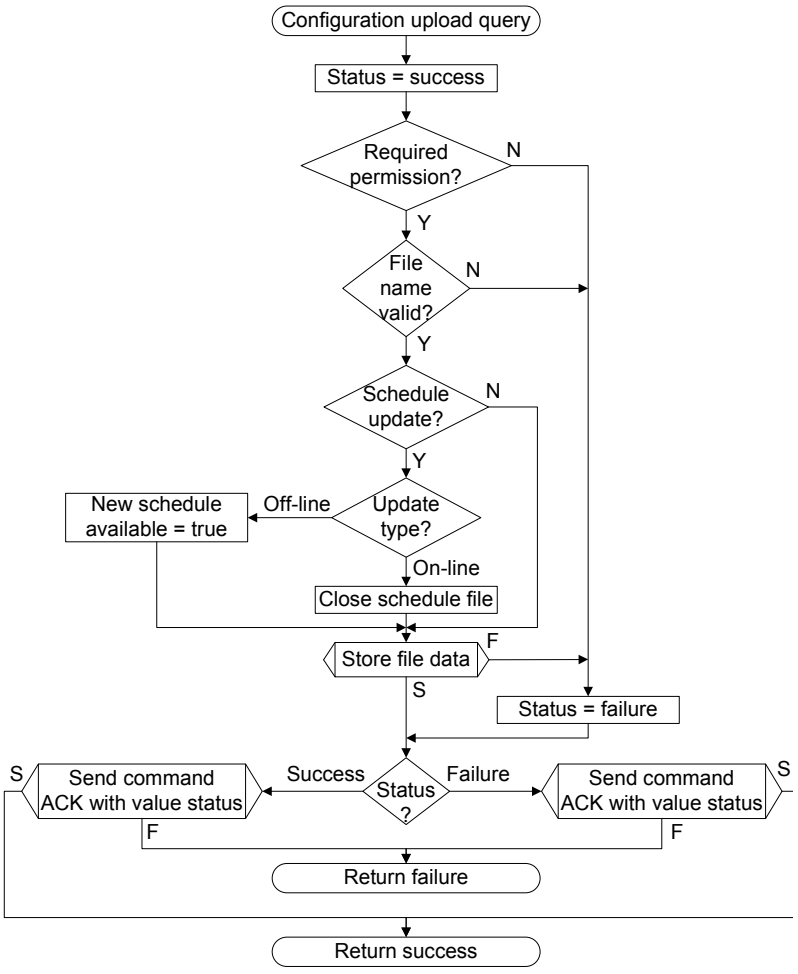


Figure 5.14: Flow diagram depicting the process of handling a configuration upload query.

update type. The type is determined by examining the schedule file name. The update mode is off-line if the off-line file name has been received and on-line if the on-line file name has been received. For the current implementation, the off-line file name is `new_schedule.csv` and the on-line file name is `schedule.csv`. The new schedule available variable is set to true if the schedule type is off-line and the schedule file is closed if the schedule type is on-line. This forms part of the schedule update mechanism described in Section 5.6.1.

The data section of the XML file is stored in the configuration folder described in Section 5.5, under the name specified in the file name section after the schedule type has been determined or it has been determined that the configuration update is not a schedule update. The previous configuration file is overwritten if a configuration file with the same name is uploaded. The status variable is set to failure if the store procedure fails. The next step is to transmit the command acknowledge response to inform the ground station whether the command was successful. A command acknowledge failure message is transmitted if the upload was unsuccessful and a command acknowledge success message is transmitted if the upload was successful.

Listing 5.6: Schedule upload command transmission file

```
<stationdata>
        <sourceid>5</sourceid>
        <querytype>21</querytype>
        <file_name>schedule.csv</file_name>
        <file_data>npt,2.00,7.00
pmb,7.00,25.00
cpt,25.00,27.00
kbl,27.00,29.00
</file_data>
</stationdata>
```

The command acknowledge message is described in Section 5.8.5. The configuration upload function returns failure if the acknowledge function returns failure and success if the command acknowledge function returns success.

The reason for using the value of the acknowledge function as the function return value, is to be able to log whether the ground station was informed about the status of the command or not.

Listing 5.6 shows an example transmission file of a schedule configuration upload command, where the source ID is the ID of the ground station uploading the new configuration, the query type is the configuration upload type, the file name is the name of the configuration file to store and the file data are the configuration file data to upload. The file is legible as the configuration files use a text format as described in Section 5.4.2.

5.8.5 Command acknowledge response

The command acknowledge response fulfils two roles. The first is to inform the system administrator in what state the SCSS is in, after a configuration update is made. The configuration update is described in Section 5.8.4. This is done by acknowledging the command received and relaying whether it was a success or failure, to the source ground station. It is crucial that the system administrator knows whether a command was correctly executed on the satellite to know how future commands and communications will be handled by the satellite.

As an example, if a new emergency ground station, described in Section 5.6.3, is added and an old one is removed, it is important for the system administrator to know whether the new ground station has in fact become the new emergency station or whether the old one should still be used to upload an active schedule to the ground station.

The command acknowledge response is also used to signal the end of a download session, described in Section 5.8.6. This informs the ground station when all files have been received. After this point, the ground station may upload the next query to the satellite. For this purpose, the message is used in both the log download command as well as the mailbox download request. The required fields for both modes are the destination ID, the query type and the file data section. As this message is only

Listing 5.7: Command acknowledge response transmission file

```
<stationdata>
    <destid>5</destid>
    <querytype>2</querytype>
    <file_data>21: SUCCESS</file_data>
</stationdata>
```

Listing 5.8: Schedule upload command transmission file

```
<stationdata>
        <sourceid>5</sourceid>
        <querytype>12</querytype>
</stationdata>
```

sent after all files have been downloaded, a ground station receiving this message is assured that all the files in the source folder have been downloaded. A ground station regularly not receiving this message should have its required communications time extended to allow it to receive all data destined for it. This should be done on a policy level, where more time is requested from the administrator of the system schedule. He or she will then be able to allocate more time to the system. Listing 5.7 presents an example transmission file of a command acknowledge, where the destination ID is the ID of the ground station to which the acknowledgement is sent. This is the source ID of the received query, which is being acknowledged. The query type is the command acknowledge type and the file data section contains the query type being acknowledged, along with either the string “SUCCESS” or “FAILURE”. In this example, it is acknowledged that the configuration update command was handled successfully.

5.8.6 Mail download request and log download command

Both the mail download request and the log download command initiate a download by the satellite to the ground station. The only difference is the log download command initiates a log download from the logs folder, whereas the mailbox download request initiates a download from the station mailbox of the ground station that uploaded the request. The only required fields for both queries are the source ID and the query type.

Listing 5.8 presents an example of a mailbox download request transmission file, where the source ID is the ID of the ground station that sent the message and the query type is the mailbox download type. The log download command message only differs from the mail download request, in that the query type is the log download command type, which is 22. In both cases the source ID serves as the destination ID for the satellite download response, described in Section 5.8.7.

Figure 5.15 shows the process flow of the general download request. Initially, the status variable is set to success. The next step is to check for the required permissions. For a log download request, the station must be a control station, but for a mailbox

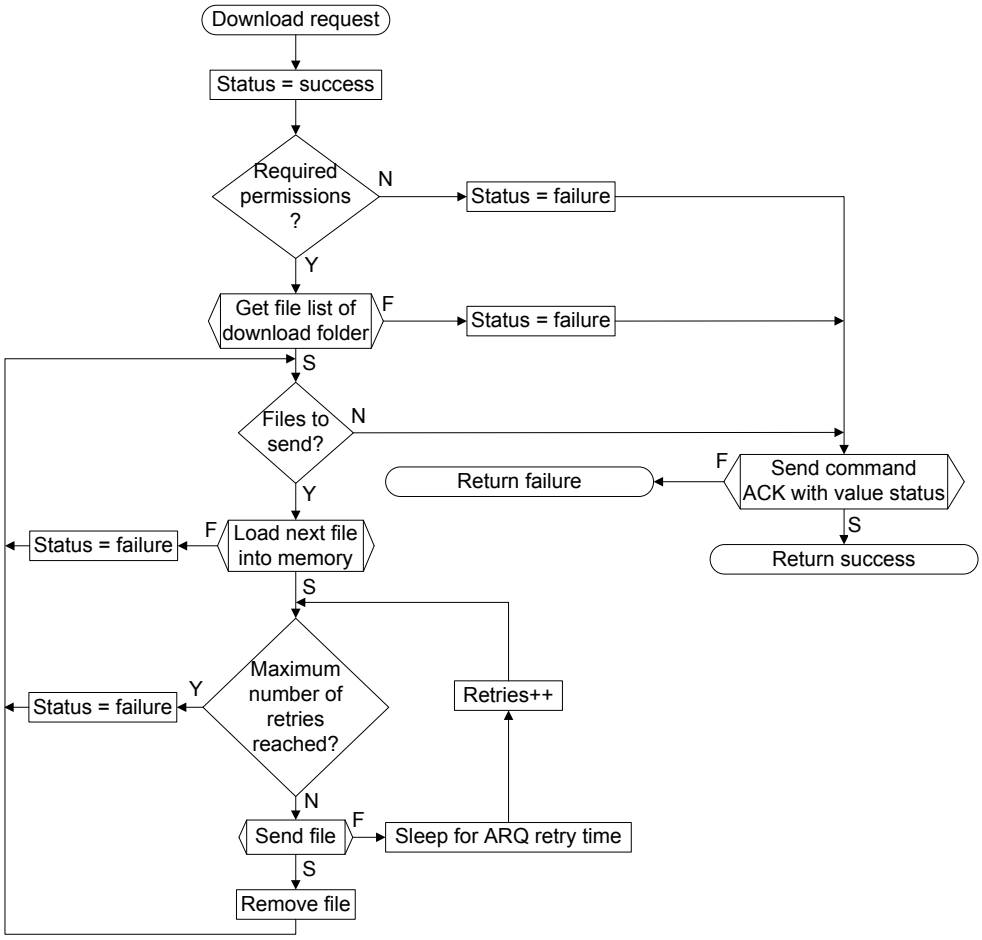


Figure 5.15: Flow diagram depicting the process of handling a general download request.

download any station type is sufficient. A list of file names in the download folder is retrieved if the required permission are met. The download folder is either the mailbox of the source station or the logs folder for a log download. The files have to be sent sequentially, after the list of files has been successfully loaded. If there are no more files to send, or the functions to retrieve the file list failed, or the required permissions were not met, a command acknowledge response is sent with the value of the status variable determining a success or failure response. The download request function returns success if the command acknowledge function returned success, and returns failure if the command acknowledge function returned failure.

In case there are still files left to send, the next file is loaded into memory. The status variable is set to failure if the function fails and the next file is then handled. The necessary steps are taken to send the file to the ground station after the file has been successfully loaded into memory. This entails creating an XML file from the loaded file data and passing that file name on to the ARQ protocol. The file is removed from the file store and the next file is handled after the file has been successfully sent. Downloaded files are removed to save space on the satellite. A failure in the file send process will trigger the retry mechanism. The SCSS will sleep

Listing 5.9: Download response transmission file

```
<stationdata>
    <destid>5</destid>
    <querytype>3</querytype>
    <file_name>satellite_icon.gif</file_name>
    <file_data>
R0lGODlhEAAQAOZwAP7+/klJSVFRUUdHR6urqzQ0NOjo6Nvb20JC
Qjo6Ojw8PD4+Pk1NTWhoaGJiYiwrKaGgoEhISBUVEPX19SwsLC8v
LR8gHFBQUM/Pz0FAPvLy8qWlpdbW1hYWFfn5+Tk5OVhYWJeXlzAw
MJqamj09Pf38/W1tbZmYmUhHRi0tLVNTU/38/jM0L9LS0nZ2dhcY
E+Dg4Ds7O15eXqGhoObm5kFBQWdnZ0JEQFRUVCYlIkRERC0tLB4f
GrGxsEhHSDY2NiEhHBEREcjIyXh5eZKSknFwbvHx8Ts6OCkoJ5CQ
kEtLSbCxr/39/UpKSvr6+nFxcURDQXh2eLGxsff3+KKipCUkHzEx
MURFRezt7TMzMCkqJdPT0zc3N1FQUPz8/L+/v5ubmWzY1NR8e
HiYmJi0uKJ2dnUZGRnx8fCgpKDo6O3Fzbzc3MqioqISEhP//////
/wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAACH5BAEAAHAALAAAAAAQABAAAAezgHCCg4SFhExSbmYw
Hm+Cb46DbyE/AgwDYT0lK2BUkYIOCAI4FzpcURBXQp9wBAsRAQwB
A2JII6xwBwUIJAoLalAPSgaFTgIfFCIFCRlZL0VehG8bMSpBCU0o
bCwWS58AbQ0ERFY7R0BaEmsAgwcOMk8NA2k5VWU8N1iDXy41KWcg
yIx5UKHDEFZvtpiwgcHIiS4+Zkwx9KYFmiTE4LAzJIhGAAUEOBbS
wOHNRpGCAEwQGQgAOw==</file_data>
</stationdata>
```

(block) for the same ARQ retry time length described in Section 5.7. The number of retries is incremented and the SCSS tries to resend the file if the maximum number of retries has not been reached. The status variable is set to failure if one of the files cannot be sent and the maximum number of retries has been reached. The next file is then loaded.

It is important to ensure that a file is not deleted if it has not been downloaded successfully. This is done by implementing all functions in the download request process with return values. The file will not be deleted if any of these functions fail and the file is not sent to the ground station. Only after all functions, including the function controlling data transmission, have returned success, will the file be deleted. As the lower level link is a reliable link, as described in Section 3.3, all data transmitted over it will necessarily be delivered to the ground station.

5.8.7 Download response

The download response is the message downloaded to a ground station in response to a download request. The required fields are: the destination ID, query type, file name and file data. Listing 5.9 presents an example of a download response, where the destination ID is the ID of the ground station for which the download is destined. The query type is the download response type, the file name is the name of the file uploaded earlier and the file data contains the file itself. The file downloaded in the

example, is the file uploaded earlier by the upload command in Section 5.8.3. In an effort to ensure that data are transmitted as efficiently as possible, each download file is compressed by the lower level transport protocol.

5.9 Logging

Extensive logging serves an important purpose in any software system. It allows administrators to gauge the performance of the system and it allows developers to quickly and efficiently discover and fix software errors. To this end, the entire SCSS was designed to log all actions taken and all errors encountered. Logging for the SCSS is of greater importance than for a general user-run program. The reason for this is that the SCSS works autonomously, with no direct user input. All actions taken are taken with no user overseeing the system. This is why it is important to use logging to enable administrators to inspect the logs and from these files discover what actions were taken by the system, to ensure that these actions were the correct ones.

Every piece of software logs its execution and the actions it takes. All major and minor system failures are also logged. Major system failures are states that force the SCSS, or a part of the system, to terminate. Minor system failures are failures from which the SCSS can recover. An example of a minor system failure is an unknown station query.

Every week a new log file is created to ensure that a log file does not become too large to transmit. Logs are stored in the runtime logs folder, as discussed in Section 5.5. Stored logs are retrieved by a log download command, after which the logs are deleted, as described in Section 5.8.6.

The format of a log message is shown below, along with an example log message.

```
day-month-year hour:min:sec: (Entity name): (Message)
04-08-2009 23:57:15: SERVER: Received activation acceptance
```

The initial part of the message, including the date and time, is the time stamp. The second part shows the name of the thread or process where the function is located and the third part shows the message itself. The example log entry shows a message generated by the station server to inform the administrator that the SCSS received an activation acceptance message at the specified date and time.

5.10 Conclusion

This chapter described the design of the SCSS that enables the satellite to communicate with ground stations on a file level. Initially, a functional analysis was done on the SCSS to establish what functionality the system should support. A system design comprising of different elements was then proposed, where each element performs a unique function, but all elements work together to fulfil the functional requirements of the satellite system.

The station server was presented as the controller of the software system and the station handler was presented as the the provider of services requested by ground stations. The means of communication was investigated and presented as textual file

formats. The files supporting the communications system were also discussed along with how these files are updated and used by the SCSS. The storage of these files was also discussed.

Mechanisms were presented that enable the SCSS to perform its duties as well as to prevent critical system failure. These mechanisms span across multiple functions and structures to strengthen the communications system and implement safe-guards against the space and embedded environments. Along with the mechanisms presented in the station handler and server sections, the logging strategy was also described. Robust data logging is implemented throughout the system, in all processes across all functions. This enables system maintainability and upgradability.

Throughout the design of the SCSS, the embedded system and space environment were kept in mind. Important factors influencing the overall system design were limited system resources such as memory and CPU power, as well as the real-time nature of the system. These factors influenced the way functions were designed as well as the methods used to ensure system stability over long periods of operation.

Chapter 6

Implementation, Testing and Performance

6.1 Introduction

After the completion of the design phase of the project, all entities described therein were implemented. This chapter chronicles the implementation of the SCSS, while describing the work environments, challenges, testing and performance. The purpose of this section is to show that the goal of minimising resource usage was achieved and that testing, which is of great importance to software systems, was given due importance. Another goal is to present the SCSS from the practical perspective of the support engineer or technician.

Section 6.2 describes the details concerning the specific programming environments used to implement the SCSS, the scheduler and the visibility prediction algorithms. Section 6.3 presents the program flow of the MATLAB script used to perform all the position predictions and visibility calculations described in Chapter 4. Section 6.4 describes methods used to reduce the memory usage of the SCSS. Section 6.5 describes techniques used to reduce the CPU utilisation of the SCSS. Section 6.6 describes the benifits of using multi-threading, but also the challenges faced, when dealing with a multi-threaded system implementing cancellation. Section 6.7 discusses how the scheduler was implemented in software and how it functions from a user perspective. Section 6.8 discusses the details of the SCSS implementation. As with the scheduler implementation, the SCSS implementation is described from the perspective of the user or software support engineer. Section 6.9 describes the functional and non-functional testing methodologies used, to ensure the correct functionality of the SCSS. Section 6.10 gives some performance figures of the SCSS. This shows how well the SCSS achieved the resource utilisation goals set out throughout this work.

6.2 Development environments

Two development environments were used during the development of the scheduler and the SCSS. These are MATLAB and the QNX Momentics development environment (QDE) [75]. MATLAB was used to implement the satellite propagator

and QDE was used to implement the SCSS and scheduler which runs on the OBC, introduced in section 3.4.

The satellite position prediction algorithm makes use of large matrices to predict and manipulate the predicted position. MATLAB was chosen for the ease with which large matrices can be manipulated as well as for the ease with which the matrices can be visualised. The visualisation is important during the development phase of the algorithm, as it allowed for the verification of generated data. Although MATLAB uses more memory and CPU power, these factors are not as important for the scheduler as the CTWs are generated off-line and on the ground. The benifits of easily manipulating and visualising the data are greater than the drawbacks of decreased memory usage and increased CPU utilisation.

QDE is used for all C development performed on-board the OBC. QDE 4.6 is based on Eclipse version 3.3 and contains all the "perspectives" of Eclipse. There are two main differences worth mentioning. Firstly, the Integrated Development Environment (IDE) allows for compiling code natively on-board the OBC and then piping the output back to the IDE. This supports features such as remote graphical step-through debugging, resource monitoring and code profiling. It also greatly decreases development time as executables do not have to be downloaded manually to the OBC via the File Transfer Protocol (FTP). The output of the executing program is also piped directly to the built-in console, to enable the developer to monitoring the status of executing code.

Step-through debugging presents a view of the program output as well as the executing code. An arrow shows which line of code is executing and the developer can then step from one line to the next and monitor all variables, to check for unexpected changes. Resource monitoring logs the number of memory allocated and deallocated during the execution of a program. Code profiling monitors which functions are executed during program execution. This allows for thorough test code to be written, because after a set of tests are performed, the IDE will also present the developer with a report to show which functions executed.

A second important feature of QDE is the built in QNX compiler allowing code to be compiled for a wide range of hardware types. In the case of the satellite OBC, an SH4 processor is used. When creating a new project, an SH4 architecture is selected and all code compiled will execute on an SH4 running QNX. All code compiled using QDE will only execute on platforms running QNX.

All SCSS code were compiled in QDE and tested to run on the OBC as described in section 6.9. The use of IDEs are not required to compile and test C code, but they greatly decrease development time and simplify and support program testing. The decrease in development time is due to library functions that are easily accessible, code completion and support for projects with makefile generators. These makefile generators present the user with a graphical interface where libraries may be added and class paths defined. The generator then generates the required commands for the C compiler to correctly link all library objects with the program and then build the project.

The simplification of testing is achieved through features such as step-through debugging and code profiling. When writing test functions, code profiling can be a powerful tool, which allows the tester to know which parts of a program are being tested and where test functions have to be added in order to test unexecuted code.

6.3 Position prediction and visibility calculation implementation

The theory developed in Chapter 4 is implemented as a MATLAB script. This script uses the position of a ground station and satellite to generate a CTW series for a specific satellite-ground station combination. The steps taken in the script are:

1. Set step time to number of seconds.
2. Define the mean Earth radius, gravitational constant and Earth's sidereal rotation period.
3. Define satellite orbit parameters.
4. Calculate satellite period from orbit parameters.
5. Set starting point of satellite as above the North pole.
6. Define latitude, longitude and height above sea level for the ground station.
7. Convert ground station position to Cartesian coordinates.
8. Calculate communications range of satellite.
9. Calculate angular step size, from the step time for the satellite and ground station.
10. Set the simulation stop time.
11. Calculate number of data points required, using the stop and step times for the satellite and ground station.
12. Create the ground station rotation matrix.
13. Create the satellite rotation matrices.
14. Calculate the satellite position vector.
15. Calculate the ground station position vector.
16. Calculate the ground station-satellite distance vector.
17. Calculate the vertical angle.
18. Project satellite and ground station positions onto the horizontal reference plane.
19. Calculate the horizontal angle.
20. Calculate the left or right plane for the horizontal angle.
21. Calculate CTW times from the distance vector.
22. Compute CTW statistics.

23. Write CTWs to file.

Step 1 defines the resolution of the prediction. The smaller the step time, the more accurate the prediction and longer the calculation. Steps 2 and 3 define the Earth and satellite constants used to later calculate the satellite period in Step 4 and to define the ground station period. Step 5 defines a starting point for the satellite position prediction. Step 6 defines the ground station starting point in latitude and longitude coordinates and then converts the coordinates to Cartesian coordinates in Step 7. Step 8 calculates the range at which communications may occur from the visual range or a 60° antenna turning angle range, as described in Section 3.2 in Step 8. Step 9 calculates the angle by which both the ground station and satellite should be rotated during every step time, to complete their orbits within the calculated periods. Step 10 sets the simulation stop time, which defines the period for which prediction will be performed. Step 11 calculates the total number of data points required for the defined stop time and step time, to enable array sizes to be defined.

The rotation matrices may be created and the positions predicted, as described in Sections 4.5 and 4.6, after all required variables have been calculated. Steps 12 and 13 define the various satellite and ground station rotation matrices. The position matrices are then iteratively calculated in Steps 14 and 15, starting from the initial point earlier defined and applying the calculated step angle for every time step, using the rotation matrices. After the position vectors have been created, the satellite-ground station distance vector is calculated in Step 16, as described in Section 4.7.

The angles are then predicted as discussed in Section 4.8. Step 17 calculates the vertical angle. The horizontal angle is predicted in Steps 18, 19 and 20. The CTW times are then calculated in Step 21, as described in section 4.9. Step 22 computes the CTW statistics, which include minimum, maximum and average communication times, average passes per week and total communications time per week. Finally, Step 23 writes all CTW data to a file for processing by the scheduler described in Section 4.4.

6.4 Designing for memory limited systems

The principal challenge facing embedded systems is resource limitations. The SH4 board has a total of approximately 2 MB of memory available after all device drivers and interface modules have been loaded. The 2 MB of memory, not only has to house the SCSS, but also the transport protocol, the first sub-layer of the TM data link protocol and the antenna control software. This severely limits the amount of memory available to the SCSS.

There is no one solution to this problem, but a methodology that has to be followed during the complete development life cycle of the SCSS. The methodology is to always write code that uses as little required memory as possible, without placing unnecessary strain on the CPU, which also has a limited capacity. Usually when implementing software, there is a trade-off between storage and processing power. Using less memory, usually requires more CPU time to perform the same task and vice versa. A balance has to be found between memory usage and CPU utilisation.

One method used to reduce memory usage, is to only allocate the required memory for any array structure in C and then reallocate the memory dynamically as the array structure grows. A memory reallocation is CPU intensive as a contiguous block of memory might not be available. In this case, the realloc function creates a new larger block of memory and copies the data of the old block into the new block and then frees the old block's memory [76]. In an attempt to obtain a balance between processing and memory usage, memory is allocated in blocks of increasing size. The next allocated block of memory is double the currently allocated block of memory. Although this approach may waste half the allocated memory in the worst case, it decreases CPU utilisation.

Memory usage is also reduced by the schedule loading mechanism. The schedule file may be of any length, depending on the total time length of the schedule as well as the length of the individual requested communications times of each ground station. Memory is wasted if the SCSS loads the complete schedule. The total amount of possible memory usage is also difficult to specify, as the usage is dependant on the schedule size, which in turn adds a size constraint to the schedule.

The SCSS is not required to be aware of the complete schedule at any given time. A single schedule item is handled at a time and so a single schedule item need be loaded at a time. To reduce memory usage, a ribbon loading approach is taken when loading the schedule. The schedule file is opened by the station server, but only the current item is loaded into memory at any given time. After an item completes, the schedule item is unloaded and the next item is loaded. This allows for a schedule of virtually unlimited length.

6.5 Designing for CPU cycle limited systems

The effort to reduce CPU utilisation having the single largest impact on the program design, was the decision to never make use of polling. Polling is a method to enable one entity to check whether information is available for it to process. The information source is constantly asked for information, until information becomes available. The information is then processed and the source is polled for new information. An application that constantly polls, uses CPU cycles even when no work is being performed and no reply is received from the object being polled.

An approach that significantly reduces CPU utilisation, is to have a process block until the required information becomes available. When the information becomes available, the entity which produced the information then signals the blocked process, which unblocks and processes the information. Blocked processes are not scheduled by the operating system, and therefore require no CPU cycles.

When designing a software system for a resource limited platform, it is important that all processes block when not performing work. The simplest method by which to block the running process or thread is to use the C "sleep" function. This function sleeps for a specified number of seconds. This method may be used when the block time is known, but this might not always be the case.

For example, a system waiting for information is not aware of how long it will take for the information to become available. The solution of sleeping for a small amount of time and then polling for information, while being simple to implement,

Listing 6.1: Wait for signal

```
//Global variables
int running;
pthread_mutex_t running_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t running_cv = PTHREAD_COND_INITIALIZER;

...

//Wait for the other thread to cease execution
pthread_mutex_lock(&running_mutex);
if (running)
        pthread_cond_wait(&running_cv, &running_mutex);
pthread_mutex_unlock(&running_mutex);
```

Listing 6.2: Signal waiting thread

```
pthread_mutex_lock(&running_mutex);

running = 0;
pthread_cond_signal(&running_cv);

pthread_mutex_unlock(&running_mutex);
```

is also not an optimal one as it still implements a polling design. Listings 6.1 and 6.2 show a solution that has the thread waiting for information, block, and when information becomes available, unblock.

Listing 6.1 shows the process waiting for the information. The information is the value of the *running* integer. Initially, the running integer is declared and a mutex lock (`running_mutex`) and a condition variable (`running_cv`) is created and initialised. A mutex lock prevents more than one process from concurrently changing the value of a single variable. The condition variable is the signalling device used.

When information is required, the process checks whether the running variable has been set. If this is not the case, the condition wait function is called. This function blocks the process and waits for a signal via the condition variable. When the function returns, information will have been made available. There is also a timed wait function, which allows the calling process to time-out if no information is received within a certain time frame.

Listing 6.2 shows the information producing process. When the information is made available, which in this case is achieved by setting the running variable to zero, the mutex is firstly locked. This prevents the waiting process from reading or writing to the variable. After the variable has been altered, the signalling process sends a signal via the condition variable and again unlocks the mutex. This mechanism wastes no CPU cycles in polling, but rather uses an efficient lock-step approach to solve the problem of waiting on information from other threads.

An example of blocking by using the standard C sleep function in the SCSS is found in the station server, where the server has to wait for the allowed station

communications time to expire. After the station server creates the station handler thread, it blocks (sleeps) for the allowed communications times. When it unblocks, is cancels the station handler if it is still executing.

An example of the lock step notification mechanism found in the SCSS is the cancellation lock-step mechanism, described in Section 5.6.2. The station server blocks until the station handler has cancelled, before scheduling the next handler. When the station handler is ready to cancel, it notifies the station server to unblock and cancels.

6.6 Multi-threaded systems with cancellation

There are many benifits to using multi-threaded software which mostly stem from the modularisation and concurrency of multi-threaded code. Threads assist a designer in writing discrete work entities that work together to achieve a common goal. This approach simplifies the coding process and also allows for compartmentalisation of code. This also increases maintainability and improves manageability. Multi-threaded code can also execute concurrently on multi-core systems, which greatly improves execution time.

The only drawback with multi-threaded code is the increased complexity in the design. Some of the factors that have to be taken into account are that multiple threads may not access the same variable concurrently. This can lead to the variable being in an undetermined state. One, therefore, has to take this into account when manipulating data visible to all threads. The solution to this problem is to use mutual exclusion (mutex) locks, whenever a variable is used that is accessed by multiple threads.

A mechanism adding significant complexity to a multi-threaded design, is cancellation. Cancellation is used in the SCSS to limit the time each ground station may communicate for. When the communications time of a ground station expires, the station handler for that ground station is cancelled if the ground station has not already closed the connection. To solve the thread cancellation issue, various mechanisms are implemented as described in sections 5.6.2 and 5.7.2.

With the thread cancellation issue solved, the multi-threading implementation of the SCSS allows for great flexibility in design as the station server is implemented as a separate thread from the station handler.

The decision to use multiple threads to multiple processes should also be discussed. Both an advantage and disadvantage of using multiple processes to multiple threads, is that separate processes execute in separate memory spaces. Separate memory spaces improve the overall robustness of the SCSS. If the station handler function tries to access an illegal block of memory, the process will be destroyed by the operating system. If that process executed in a separate memory space, the station server that spawned it will not be destroyed. This will allow the station server to monitor the created process and take corrective action if an illegal action is performed.

The disadvantage of a separate memory space is that memory that could have been shared can now no longer be. This increases memory usage and also complicates the communication mechanism. Inter-process communication requires kernel calls to

send a message from one process to another, while inter-thread communication only involves accessing a shared block of memory. Shared memory communications are also available for the QNX operating system, but not natively supported in the kernel. The process manager has to run, which decreases the efficiency of message passing [37]. The multi-threading approach was taken to remove IPC overhead and save memory.

6.7 Scheduler implementation and testing

The scheduler program is written in C and uses CTW files to produce a communications schedule. The program requires two types of files, the first type is the station information file, from which it retrieves the station names, required communication times and priorities. The second type is the CTW files.

When the program starts, the station information file is read and all required station information is stored in memory. After all station information has been stored, the scheduler reads all ground station CTW files listed in the station information file. A file name is constructed by using the station name and appending "`.ctw`" to it. The scheduling algorithm illustrated in Figure 4.2 is executed after all CTWs have been loaded. The algorithm generates a schedule written to a schedule file.

The program uses command line input to execute, and takes the station information file as input argument. It then expects all CTW files to be present and terminates with an error if all files are not present. No CTW file, other than those specified in the station information list will be loaded.

The schedule file generated, can by uploaded to the satellite with a schedule upload command as described in Section 5.6.1. A new schedule should be regularly uploaded to the satellite so as to avoid the activation of the emergency communication mechanism as described in section 5.6.3

To test the schedule, 64 ground stations were placed approximately 90 km apart in a rectangular area stretching from Cape Town to Nelspruit. This created a system with ground stations entering visual satellite range less than 10 seconds apart. All ground stations were assigned a required communications time of 45 seconds. This allows for 105 KB of data transferred per ground station, for a 19200 bits per second communications link.

During this test, all ground stations were allowed to communicate in a three day period at least once. What was noticed is that after day one, all ground stations could not communicate, because of a lack of capacity of the system. Those ground stations that could not communicate on day one, were prioritised the following two days. This shows how the scheduling algorithm equalises all ground station communication times to avoid starvation.

This also becomes clear when looking at the number of times ground stations were allowed to communicate for. All ground stations were allowed to communicate with a mean of 1,33 times and a variance of 0,26 times. This again shows the little variance between different ground stations.

This test showed that the algorithm performed adequately, to be implemented on the SCSS. The size of the system matches the expected size of the communications system, and the amount of data that could be transmitted was twice the required

amount.

6.8 Satellite Software Communications System implementation

The SCSS is written C, to execute on the QNX Neutrino operating system described in Section 3.5. The SCSS is designed to be executed by a bash script, which should in turn execute as a cron job when the satellite communications payload is active.

The communications payload is deactivated during the time when the satellite has passed the last ground station and the time before the satellite is within communications range of the first ground station of the next pass. When the payload is activated, the SCSS should be initiated, along with the rest of the protocol stack. The station server then uses the system clock to check where in the schedule it is and schedules the next ground station for communication.

The SCSS requires no parameters as input arguments as all parameters are set in the system parameters file described in Section 5.4.4. When the program is initiated it executes as discussed in Chapter 5.

An example of the log output is shown in Appendix A. Initially it attempts to initiate a connection to a ground station. No response is received from that ground station for the duration of its communications period. The software then attempts to contact another ground station, which is successfully connected through use of the activation offer and activation acceptance handshake mechanism.

The ground station then elects to perform some tasks. These tasks are, in order: a mailbox download request, a log download request and a new schedule upload. These requests also prompt all required handler responses. When the communications time expires, the connection is closed and the next ground station is contacted. The next ground station performs a file upload, which is stored in the destination station's mailbox. After the upload, the timer expires and the connection is closed.

6.9 Testing

In order to validate a software design, it is imperative that adequate testing be performed. Testing verifies whether a product functions correctly and gives a measure of the quality of the product. Functional as well a non-functional testing techniques exist. Non-functional testing technique test aspects of the system, which the user is unaware of. Memory usage efficiency is not directly experienced by the user, except when the complete system fails, due to a memory error or overrun.

Functional testing techniques test the functionality of the system. These tests are developed from the user specification of the system and test whether all use case scenarios were implemented correctly. Two types of functional testing were performed: unit testing and integration testing, described in Section 6.9.1 and 6.9.2 respectively. Memory testing described in Section 6.9.3 is a non-functional testing technique used.

6.9.1 Unit testing

Unit tests separately test every software unit of the SCSS as described in Chapter 5 of [77]. Unit tests should be developed in parallel with the system or before the actual development of the system. For the SCSS, unit tests were developed in parallel with the design of the system. These tests were manual tests, which implies that no automated testing environment was used. Although more time is required to perform manual unit testing, time is saved on developing an integrated testing environment. Designing such an environment was deemed unnecessary, due to the code only having a length of around two thousand lines.

For this length of code, it is still feasible to design and perform unit testing manually, whereas the available testing environments for C are quite complex and more difficult to use than their object orientated counterparts. The testing suite investigated is called CUnit [78]. The time that would have been required to set up such an environment was deemed not to warrant the time that would be gained by automatic testing.

Unit testing is performed in a test function, which calls the function being tested and prints the output to a log. All the required information is printed to enable the tester to ensure the software is functioning correctly. The debug output is then verified and the next function in the list is tested. Unit testing is performed for every function written and performed again when any function changes.

6.9.2 Integration testing

The aim of integration testing is to test the SCSS as a whole. When all units are completed, the functionality of the overall system is tested. Integration testing was done by stimulating the SCSS with multiple simulated ground station connections and messages.

To perform this testing, simulated messages have to be received from a ground station. A synthetic schedule is generated with multiple ground stations. Ground station communications are simulated by making use of the `ARQ_read` function call. The actual purpose of this function is to initiate a connection with the underlying transport layer and receive the names of received files. A test function was written to not connect to another protocol, but to report a file name from a static list of file names. The file name list can be altered and expanded to increase code coverage.

The file names specified all correspond to ground station messages placed in the incoming folder of the file store. This simulates a stage where some station requests have been received from the transport layer and should be serviced. The synthetic file list reported by ARQ is also able to simulate a connection set-up.

Figure 6.1 shows the code coverage of the station server. The only functions with low code coverage are two deprecated utility functions. Code coverage is used to verify whether a significant portion of the code is tested.

The SCSS is then tested with variations of all possible messages that can be received as requests. The tests are also set up in such a way that no one test will effect the outcome of the other tests. This is achieved by each request having a different name and all file names used during testing, being unique. This is a form of white box testing, which is similar to black box testing. For a description of

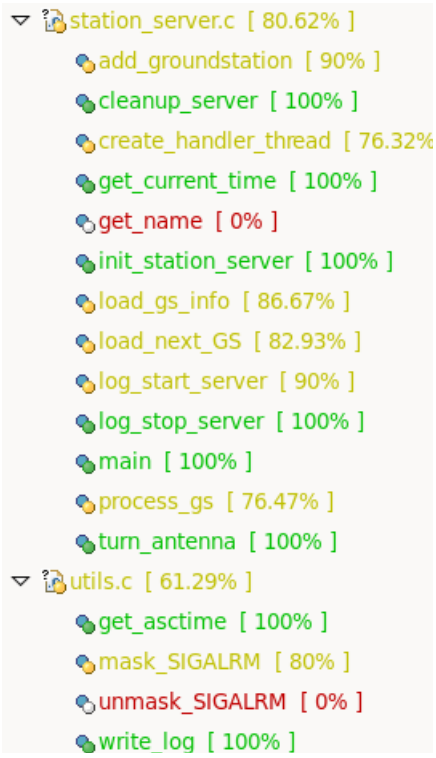


Figure 6.1: The code coverage achieved during testing of the station server and utilities files.

white box testing, see Chapter 4 of [77]. The difference between white box and black box testing is that the contents of the program, including the code itself, are taken into account. If the designer keeps the code driving the system in mind, more meaningful tests may be produced. The station messages were created by specifying the functionality of each request and then creating different forms of each request to test different failure and success scenarios.

6.9.3 Memory testing

An important non-functional testing technique, especially for embedded systems, is memory testing. This technique ensures that no memory leaks exist and that the system does not perform any memory reads or writes in unallocated or illegal memory space. The program used to perform the memory testing is the "Memcheck" tool of the Valgrind tool suite [79].

Memcheck functions by inserting extra code into a program being tested, to track memory allocations and assignments. To execute the Memcheck tool, the Valgrind tool suite must be installed and then the command:

```
valgrind --tool=memcheck program_bin
```

can be typed into the command line. This will execute the binary and test memory validity during run-time. After the program has been executed, Memcheck displays a summary of all memory lost and all illegal reads and writes as shown below.

```
==28450== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 18 from 3)
```

```
==28450== malloc/free: in use at exit: 916 bytes in 5 blocks.
==28450== malloc/free: 1,412 allocs, 1,407 frees, 134,327 bytes allocated.
==28450== For counts of detected errors, rerun with: -v
==28450== searching for pointers to 5 not-freed blocks.
==28450== checked 76,520 bytes.
==28450==
==28450== 19 bytes in 1 blocks are still reachable in loss record 1 of 5
==28450== at 0x4026FDE: malloc (vg_replace_malloc.c:207)
==28450== by 0x400A251: (within /lib/ld-2.9.so)
==28450== by 0x4005CF2: (within /lib/ld-2.9.so)
==28450== by 0x400766C: (within /lib/ld-2.9.so)
==28450== by 0x4012216: (within /lib/ld-2.9.so)
==28450== by 0x400E035: (within /lib/ld-2.9.so)
==28450== by 0x4011C1D: (within /lib/ld-2.9.so)
==28450== by 0x419F901: (within /lib/tls/i686/cmov/libc-2.9.so)
==28450== by 0x400E035: (within /lib/ld-2.9.so)
==28450== by 0x419FAC4: __libc_dlopen_mode (in /lib/tls/i686/cmov/libc-2.9.so)
==28450== by 0x4073E16: pthread_cancel_init (in /lib/tls/i686/cmov/libpthread-2.9.so)
==28450== by 0x4073F40: _Unwind_ForcedUnwind (in /lib/tls/i686/cmov/libpthread-2.9.so)
==28450==
```

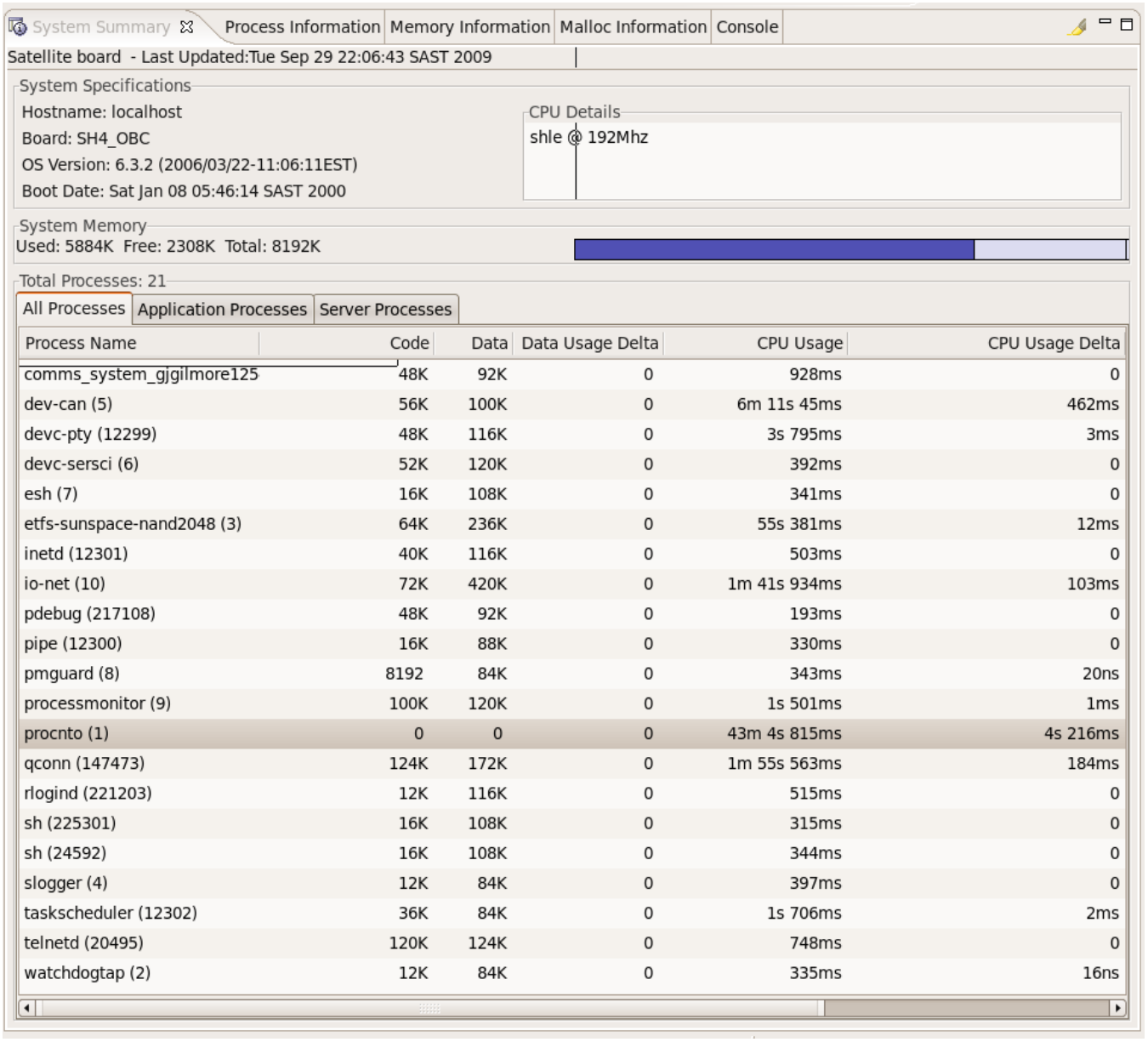
A difference is shown in the memory allocated and memory deallocated (1412 vs. 1407). The amount of memory in use at exit is 916 bytes. The block partially shows where the memory is used. Further blocks exist, but all contain the same information, where only the number of bytes lost differ. The stack trace shows that the issue originates in the `pthread_cancel_init` function. This is the function responsible for thread cancellation. If a thread is cancelled, a number of bytes still remain to enable the thread to return a value to a joining thread. The memory still in use shows the size of the memory region used to return a value to the station server.

The testing methodology used was to develop memory unit tests. These tests were implemented in parallel with the unit tests described in Section 6.9.1. Performing memory testing is very important in languages such as C, as C does not possess a garbage collector to automatically free allocated memory. For an embedded system, where only 2 MB of memory is available for the complete protocol stack and antenna controller, it is a high priority to identify and fix all memory errors.

6.10 Performance

The goals set out throughout this work revolved around implementing the SCSS that uses a sufficiently small amount of resources to enable implementation on an embedded satellite system. The maximum amount of allowable memory usage is set at a quarter of the available megabyte of memory. This figure is calculated from the assumption that there are three other processes that must also run on the system. These are the TM data link layer protocol, the ARQ transport protocol and the antenna control software.

Figure 6.2 shows the summary of all processes executing on the OBC when the snapshot was taken. The memory usage statistics reported by QDE for the SCSS is 320 kB of memory in total, of which 64 kB is code segment and 256 kB is data segment memory. This is during a file download, where the complete file must first be loaded into memory. The total memory usage of 320 KB is well under the maximum allowed usage of $2048/4 = 512$ KB of memory.



Process Name	Code	Data	Data Usage Delta	CPU Usage	CPU Usage Delta
comms_system_gjgilmore125	48K	92K	0	928ms	0
dev-can (5)	56K	100K	0	6m 11s 45ms	462ms
devc-pty (12299)	48K	116K	0	3s 795ms	3ms
devc-sersci (6)	52K	120K	0	392ms	0
esh (7)	16K	108K	0	341ms	0
etfs-sunspace-nand2048 (3)	64K	236K	0	55s 381ms	12ms
inetd (12301)	40K	116K	0	503ms	0
io-net (10)	72K	420K	0	1m 41s 934ms	103ms
pdebug (217108)	48K	92K	0	193ms	0
pipe (12300)	16K	88K	0	330ms	0
pmguard (8)	8192	84K	0	343ms	20ns
processmonitor (9)	100K	120K	0	1s 501ms	1ms
procnto (1)	0	0	0	43m 4s 815ms	4s 216ms
qconn (147473)	124K	172K	0	1m 55s 563ms	184ms
rlogind (221203)	12K	116K	0	515ms	0
sh (225301)	16K	108K	0	315ms	0
sh (24592)	16K	108K	0	344ms	0
slogger (4)	12K	84K	0	397ms	0
taskscheduler (12302)	36K	84K	0	1s 706ms	2ms
telnetd (20495)	120K	124K	0	748ms	0
watchdogtap (2)	12K	84K	0	335ms	16ns

Figure 6.2: Snapshot of the system summary, showing all running processes, their resource usage statistics and system information.

Figure 6.2 also shows extra information of the board, including the QNX version (6.3.2), clock speed (192 MHz) and memory statistics (5,75 MB used, 2,25 MB free and 8 MB total). For each process running, the number of code and data memory, CPU usage, and memory and CPU usage delta is provided. The delta gives an indication of change in memory and CPU usage. In other words, how the requirements of the program has changed over time.

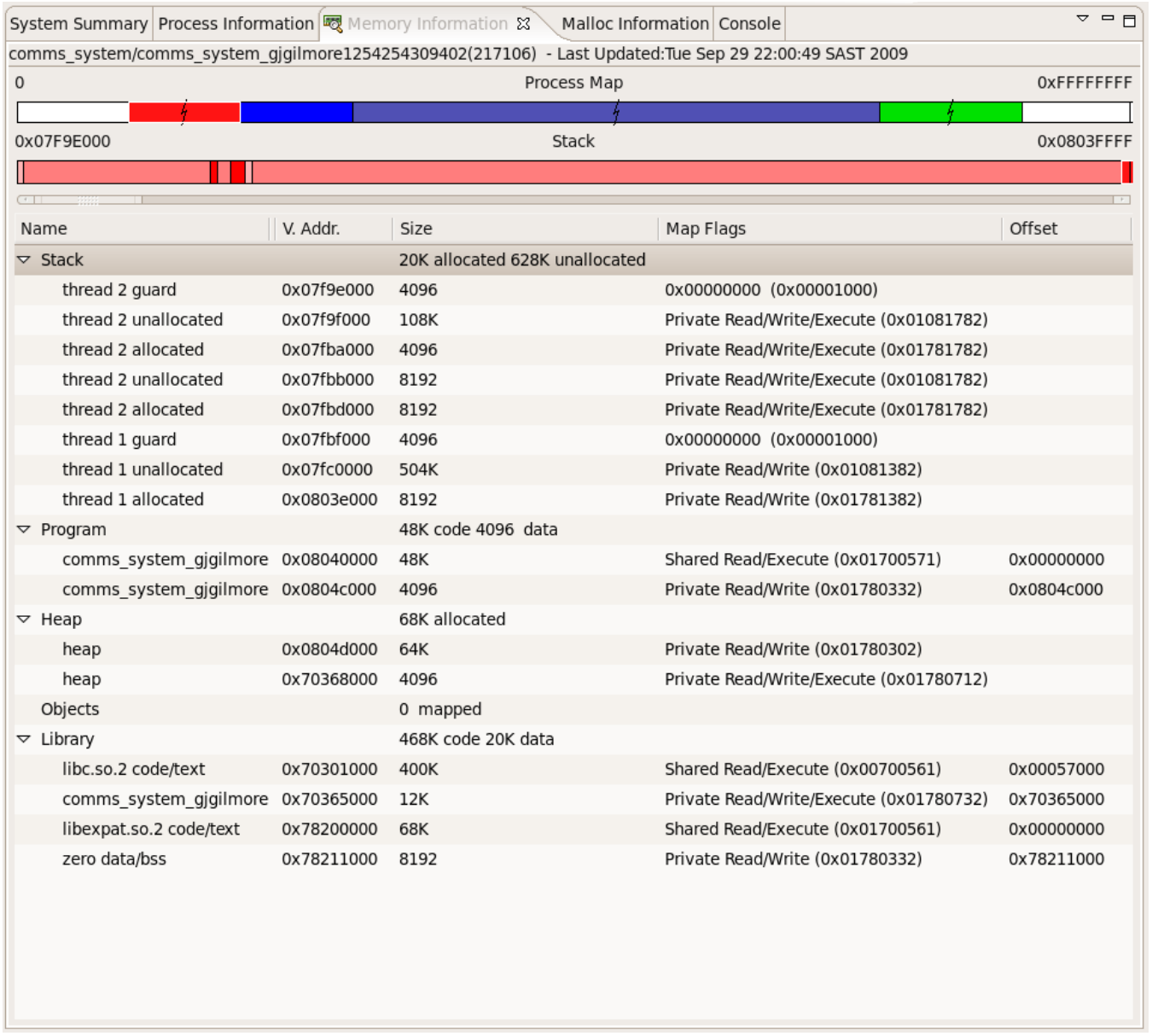


Figure 6.3: Memory information of the SCSS during runtime, showing stack, program, heap and library memory used.

Figure 6.3 presents more detailed information on the memory usage of the SCSS.

The information shows that the program has two threads running. These are the station handler and station server threads. It also shows how much memory is available to each thread and how much memory has been allocated to each thread. Memory guard regions are used to protect threads from other threads, using more than their assigned memory amount. The information also shows the amount of heap memory allocated, where dynamic memory allocation is done. From this, one can clearly see the extensive use of dynamic memory allocations. The heap size is more than three times larger than the stack size, when comparing allocated memory. It can also be seen that most of the memory used by the program is used by libraries. The largest library is the standard C library itself, followed by the expat library, used for XML parsing.

6.11 Conclusion

This section described the different environments in which development work was performed. It described the challenges faced with CPU limited and memory limited systems and also with multi-threaded systems. A practical perspective was also presented of the scheduler and the SCSS. Testing was described and the three different testing techniques used, were discussed. It is important that the initial design goals for an embedded system were achieved. The performance section showed that the software performed within allowable boundaries.

This section also marks the conclusion of the design cycle for the SCSS. All that remains is integration with the lower layer protocols and the antenna steering interface. After the integration, field tests should be performed to ensure that the SCSS performs optimally.

Chapter 7

Conclusions and Recommendations

7.1 Communication strategy

When designing the SCSS, various methods were investigated for a means to efficiently establish the satellite-ground station connection. A few major factors specified the design of the protocol. These were:

1. the satellite mounted steerable antenna,
2. the need to produce low cost ground stations and
3. the single available communications channel on the satellite.

With the satellite mounted steerable antenna, a sufficient link margin was achieved, to not require steerable antennas on the ground stations. This did, however, create the requirement that the satellite, and not the ground stations, should initiate connections, as described in Section 2.5. Two methods of tracking were also investigated: program tracking and tracking techniques, making use of satellite beacon transmissions, as presented in Section 2.3. Program tracking was chosen, as this requires no transmission of beacon packets, which frees up the communications channel for data.

A broadcast method may be used for a single channel, where the satellite broadcasts until a ground station connection has been established, processes all ground stations requests, and then broadcasts again to establish a new connection. This method is inefficient, as no knowledge of the link quality or available communications time is taken into account.

If the system possesses knowledge of link quality and communications duration, it can intelligently schedule ground station connections, to optimise the volumetric data throughput of the system. Since the satellite has a known orbit track and the ground station also moves predictably with the movement of the Earth, the distance between these two entities may be predicted. The distance prediction produces a measure of link quality and, because time can also be linked to the movements, the communication duration may also be predicted.

The predictions can then be used to schedule ground stations in such a way as to maximise the total communication time of the system and minimise the number

of ground stations, not allowed to communicate. The schedule can then be uploaded to the satellite, from where connections are made. When the schedule is created, the total amount of communications time is also equalised among ground stations, which produces a fair schedule. These two techniques produce a more efficient method of link acquisition, than a system with no past or future knowledge of the established links.

In scenarios where the link margin is not sufficient for communications, directional ground station antennas may be used. These antennas have to be pointed in some specific direction and only when the satellite flies through the antenna beam, will it be able to communicate. Angle predictions were performed to aid in calculating the optimal direction, in which the antenna should point, to communicate with the satellite for the maximum amount of time.

7.2 Satellite Communications Software System

With the communications strategy devised, the SCSS was designed. The design is based on the client-server model of communications, where the server runs and creates handler threads for clients requiring service. From this model, the schedule was incorporated into the server.

A deviation from the classic client-server model is present in the connection setup phase of the SCSS. In the classic model, clients initiate connections to the server, but because of the schedule controlled communication times, the server must initiate connections to ground stations. This difference does not, however, change the basic functionality of the SCSS from that of a classic client server model.

Multi-threading also allowed for the modularisation of the the SCSS. This simplified development, decreased the level of coupling and promoted encapsulation of data and functions. The multi-threaded approach also improved the robustness of the SCSS. If an unexpected scenario occurs, from which the SCSS cannot correct itself, only the current ground station connection is lost and further ground station may still be handled by other station handlers.

During development, resource limitations were identified as a potential development risk. To manage this risk, various mechanisms were implemented to reduce memory and CPU usage. This led to an efficient design, well suited for an embedded implementation.

The designed SCSS performed well under testing and well within resource limits set by the hardware. The thread cancellation mechanism used, requires involved memory management techniques, but the mechanism also increases the responsiveness of the SCSS. The cancellation mechanism ensures that no ground station can use more than its assigned communications time.

The schedule driven design proved to be a robust and efficient design, while implementing all features required by other sub-systems and ground stations. Features required by other sub-systems include providing a pointing target for the antenna control software to point to and providing the lower lever protocol layers with data to transmit. Features required by ground stations are those enabling the ground stations to implement a store-and-forward system over the satellite link for remote monitoring applications. These features include: allowing ground stations to com-

municate with the satellite on a file level, allowing ground stations to upload and download data and allowing ground stations to send data to other ground stations.

7.3 Contributions

The three main contributions of this work, are:

1. the development of a communications strategy,
2. design and implementation of the SCSS
3. and a centralised control approach.

The main contributions of the SCSS are:

- A resource efficient and multi-threaded high level communications control system with a high level of responsiveness.
- The scheduler, which determines the communication times of ground stations.
- The definition of all messages and their formats, which will be used for satellite-ground station communications.
- The Station Server commanding the antenna control software and thereby supporting the satellite mounted steerable antenna.

The main contributions of the communications strategy are:

- The possible maximisation of volumetric data throughput of the satellite system.
- This is achieved by efficiently scheduling ground station communications.
- The calculation of the satellite position as a function of time, using orbital elements and orbital mechanics, enable the calculation of the communications schedule.
- Fair assignment of communication times are also ensured by the scheduling algorithm.
- Position prediction also allows for the calculation of the communications statistics by making use of visibility predictions. These predictions were compared to the STK software package and found to match closely with the values predicted by the STK.
- Predicted total communication times enable system designers to calculate the overall system capacity as well as the capacity of each ground station, according to its geographic location.

- Angle predictions also enable system designers to optimally point static directional ground station antennas. Directional antennas may be required in areas where the link margin is insufficient. The angle predictions provide an optimal elevation angle, as well as an optimal horizontal angle, measured from true North.

The main contributions of the implemented centralised control are:

- The promotion of on-board processing techniques as well as satellite autonomy.
- Ease of administration and coordination.
- Improved response time because of the link acquisition strategy as explained in Section 2.6.
- By using satellite controlled link acquisition, together with a satellite mounted steerable antenna, steerable antennas can be removed from ground stations. This drastically decreases the costs of ground stations.

7.4 Further work

Firstly, the SCSS should be integrated with the rest of the communications protocol stack. This could not be done during the implementation of the SCSS, as the other projects were not at a point in their development where they could be integrated. After integration, integration testing should be performed on the communications system as a whole. This should include sending a file over an RF link and checking whether the file was received. Smaller tests between the SCSS and the antenna control software should also be performed.

Further required testing includes protocol tests, protocol efficiency evaluation and a comparison of these evaluations against other communications protocols. Protocol efficiency evaluation includes calculating the protocol overhead during a typical transmissions.

For the position prediction scheme, the schedule generator should be automated to download the two-line-elements of the satellite from Spacetrack. This will enable the schedule generator to always use up-to-date orbital elements to achieve a maximum accuracy of orbit prediction.

For the orbital prediction itself, a more accurate orbit propagator can be used, for example the SGP4 algorithm or an improved variant thereof. This will also improve the accuracy of the visibility prediction and take into account orbit perturbations. A more accurate orbit propagator requires less frequent two-line-element updates. The corollary of this is that a less accurate orbit propagator may be used, when two-line-elements are updated more frequently.

For the schedule generator, the performance of more scheduling schemes should be compared. Different schemes should be investigated to determine whether other schemes exist, that further reduce the number of ground stations not able to communicate. The *least number of exclusions* (LNE) scheme, proposed in this work, should also be compared to other schemes. The LNE scheme can also be further extended by making the scheme n levels deep, instead of one.

Currently, the scheduler looks ahead one schedule time, to determine how many stations will be excluded. But this scheme can be extended, to evaluate all possibilities and find the scheme with the total minimum number of exclusions. This scheme will most likely have a very high order complexity.

It could also be of worth further investigating the scheduling problem from a purely mathematical perspective. An optimal scheme may be proposed for the defined scheduling problem.

Another algorithm should be developed to optimise the overall communications system, when directed antennas are taken into account. The predicted angle positions of every ground station in the system should be predicted and overlaid, with the CTWs. After all CTWs are scheduled, the optimal angle should be used from every ground station. This angle would preclude other possible communications times for that ground station. The scheduling algorithm will have to be rerun. Times when a scheduled ground station could communicate, may now no longer be valid. This will give rise to different angles that should be optimised. An algorithm should be developed, enabling this calculation to converge to a system where the total amount of communications time is maximised.

Quality of service can also be implemented in the SCSS if more finely grained priority is used. The priority can then be used as a weight during the scheduling algorithm, together with the total allocated communications time, to create a schedule where some ground station receive proportionally more communications time than others.

Appendices

Appendix A

Communications software system log

```
08-01-2000 04:32:57: SERVER: Starting Station server
08-01-2000 04:32:57: SERVER: Version: 0.5
08-01-2000 04:32:57: SERVER: Date: 06-2009
08-01-2000 04:32:57: SERVER: 01-01-1970 00:00:50
08-01-2000 04:32:57: SERVER: Station loaded: pta (1)
08-01-2000 04:32:57: SERVER: Station loaded: cpt (2)
08-01-2000 04:32:57: SERVER: Station loaded: kbl (3)
08-01-2000 04:32:57: SERVER: Station loaded: bft (4)
08-01-2000 04:32:57: SERVER: Station loaded: pmb (5)
08-01-2000 04:32:57: SERVER: Station loaded: bho (6)
08-01-2000 04:32:57: SERVER: Station loaded: npt (7)
08-01-2000 04:32:57: SERVER: Station loaded: mfk (8)
08-01-2000 04:32:57: SERVER: Station loaded: stb (9)
08-01-2000 04:32:57: SERVER: Station loaded: jhb (10)
08-01-2000 04:32:59: HANDLER: Starting Station handler
08-01-2000 04:32:59: HANDLER: Version: 0.5
08-01-2000 04:32:59: HANDLER: Date: 06-2009
08-01-2000 04:32:59: HANDLER: Station: npt
08-01-2000 04:32:59: HANDLER: Start time: 01-01-1970 00:00:02
08-01-2000 04:32:59: HANDLER: Stop time: 01-01-1970 00:00:07
08-01-2000 04:33:00: HANDLER: Activation offer successfully sent for 5 after 0 retries
08-01-2000 04:33:01: HANDLER: Query source address (5), does not match current client address (7)
08-01-2000 04:33:01: HANDLER: Activation offer successfully sent for 3 after 0 retries
08-01-2000 04:33:02: HANDLER: Query source address (5), does not match current client address (7)
08-01-2000 04:33:02: HANDLER: Activation offer successfully sent for 2 after 0 retries
08-01-2000 04:33:03: HANDLER: Query source address (5), does not match current client address (7)
08-01-2000 04:33:03: HANDLER: Activation offer successfully sent for 1 after 0 retries
08-01-2000 04:33:04: HANDLER: Query source address (5), does not match current client address (7)
08-01-2000 04:33:04: HANDLER: Station handler finished execution
08-01-2000 04:33:04: HANDLER: 4 requests handled
08-01-2000 04:33:04: HANDLER: 0 requests successful
08-01-2000 04:33:04: HANDLER: 4 requests failed
08-01-2000 04:33:04: SERVER: The station handler failed to handle station npt
08-01-2000 04:33:04: HANDLER: Starting Station handler
08-01-2000 04:33:04: HANDLER: Version: 0.5
08-01-2000 04:33:04: HANDLER: Date: 06-2009
08-01-2000 04:33:04: HANDLER: Station: pmb
08-01-2000 04:33:04: HANDLER: Start time: 01-01-1970 00:00:07
08-01-2000 04:33:04: HANDLER: Stop time: 01-01-1970 00:00:12
08-01-2000 04:33:05: HANDLER: Activation offer successfully sent for 5 after 0 retries
08-01-2000 04:33:06: HANDLER: Received activation acceptance
08-01-2000 04:33:06: HANDLER: src:5, Contents: ACCEPT
08-01-2000 04:33:06: HANDLER: Activation offer accepted
08-01-2000 04:33:07: HANDLER: Received mail download request from 5
08-01-2000 04:33:07: HANDLER: Download request query successfully handled
```

```
08-01-2000 04:33:07: HANDLER: Command acknowledge successfully sent
08-01-2000 04:33:08: HANDLER: Received log download command from 5
08-01-2000 04:33:08: HANDLER: Log download command successfully handled
08-01-2000 04:33:08: HANDLER: Command acknowledge successfully sent
08-01-2000 04:33:09: HANDLER: Received configuration upload command
08-01-2000 04:33:09: HANDLER: src:5, file name: new_schedule.csv
08-01-2000 04:33:09: HANDLER: Configuration upload query successfully handled
08-01-2000 04:33:09: HANDLER: Command acknowledge successfully sent
08-01-2000 04:33:09: HANDLER: Station handler finished execution
08-01-2000 04:33:09: HANDLER: 4 requests handled
08-01-2000 04:33:09: HANDLER: 4 requests successful
08-01-2000 04:33:09: HANDLER: 0 requests failed
08-01-2000 04:33:09: HANDLER: Starting Station handler
08-01-2000 04:33:09: HANDLER: Version: 0.5
08-01-2000 04:33:09: HANDLER: Date: 06-2009
08-01-2000 04:33:09: HANDLER: Station: cpt
08-01-2000 04:33:09: HANDLER: Start time: 01-01-1970 00:00:11
08-01-2000 04:33:10: HANDLER: Stop time: 01-01-1970 00:00:22
08-01-2000 04:33:10: HANDLER: Activation offer successfully sent for 9 after 0 retries
08-01-2000 04:33:11: HANDLER: Received activation acceptance
08-01-2000 04:33:11: HANDLER: src:2, Contents: ACCEPT
08-01-2000 04:33:11: HANDLER: Activation offer accepted
08-01-2000 04:33:12: HANDLER: Received upload query
08-01-2000 04:33:12: HANDLER: src:2,dest:1, file name: satellite_icon.gif
08-01-2000 04:33:12: HANDLER: Upload query successfully handled
```

Bibliography

[1] K. Eguchi, "Tracking array antenna system," U.S. Patent 5 594 460, January, 1997. [Online]. Available: http://www.freepatentsonline.com/5594460.html

[2] S. Tirró, *Satellite communication systems design.* Springer, 1993, ch. 7.3.

[3] A. C. Clarke, "Extra-terrestrial relays - can rocket stations give worldwide radio coverage?" *Wireless World*, vol. 51, pp. 305–308, October 1945.

[4] R. E. Sheriff and Y. Fun Hu, *Mobile satellite communication networks*, ser. Probability and Statistics. John Wiley and Sons, 2001, ch. 2.2.

[5] O. Montenbruck and E. Gill, *Satellite Orbits: Models, Methods and Applications*, 3rd ed. Springer, 2005, ch. 1.1.

[6] D. Roddy, *Satellite communications*, 4th ed. McGraw-Hill Professional, 2006, ch. 2.4.

[7] P. W. Fortescue, J. Stark, and G. Swinerd, *Spacecraft systems engineering.* John Wiley & Sons, 2003, ch. 5.4.

[8] R. J. Boain, "A-B-Cs of sun-synchronous orbit mission design," in *AAS/AIAA Space Flight Mechanics Conference*, 2004.

[9] G. Maral and M. Bousquet, *Satellite communications systems*, 3rd ed., ser. Communication and distributed systems. Wiley, 1998, ch. 5.1.

[10] P. W. Fortescue, J. Stark, and G. Swinerd, *Spacecraft systems engineering.* John Wiley & Sons, 2003, ch. 12.2.9.

[11] F.-X. B. Ruiz, M. Lopriore, and L. Bella, "Procedure for controlling a scanning antenna," U.S. Patent 5 648 784, July, 1997. [Online]. Available: http://www.freepatentsonline.com/5648784.html

[12] M. A. Sturza, "Dielectric lens focused scanning beam antenna for satellite communication system," U.S. Patent 5 548 294, August, 1996. [Online]. Available: http://www.freepatentsonline.com/5548294.html

[13] T. Kitsuregawa, *Advanced Technology in Satellite Communication Antennas: Electrical & Mechanical Design.* Artech House, 1990, ch. 3.2.

[14] S. D. Ilčev, *Global mobile satellite sommunications for maritime, land and aeronautical applications.* Birkhäuser, 2005, ch. 8.

[15] F. Ghazvinian, M. A. Sturza, S. M. Hinedi, S. S. Sarrafan, and B. N. Shah, "Low-earth orbit satellite acquisition and synchronization system using a beacon signal," U.S. Patent 6 127 967, October, 2000. [Online]. Available: http://www.freepatentsonline.com/6127967.html

[16] *Space Engineering: Ranging and Doppler tracking*, European Cooperation for Space Standardization Std. ECSS-E-ST-50-02C, July 2008.

[17] G. Maral and M. Bousquet, *Satellite communications systems*, 3rd ed., ser. Communication and distributed systems. Wiley, 1998, ch. 6.

[18] G. Maral and M. Bousquet, *Satellite communications systems*, 3rd ed., ser. Communication and distributed systems. Wiley, 1998, ch. 10.5.5.

[19] P. W. Fortescue, J. Stark, and G. Swinerd, *Spacecraft systems engineering*. John Wiley & Sons, 2003, ch. 13.

[20] P. Zetocha and L. Self, "An overview of agent technology for satellite autonomy," in *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 1999, pp. 64–68.

[21] W. J. Larson and J. R. Wertz, Eds., *Space mission analysis and design*, 3rd ed., ser. Space Technology. Microcosm Press and Kluwer Academic Publishers, 1999, ch. 13.1.2.

[22] D. G. Lockie and M. Thomson, "Spacecraft antennas and beam steering methods for satellite communciation system," U.S. Patent 5 642 122, June, 1997. [Online]. Available: http://www.freepatentsonline.com/5642122.html

[23] W. M. Mularie, "Department of defense world geodetic system 1984, its definition and relationships with local geodetic systems," National Geospatial-Intelligence Agency, Tech. Rep., 2000.

[24] K. D. Froome, "A New Determination of the Free-Space Velocity of Electromagnetic Waves," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 247, no. 1248, pp. 109–122, 1958.

[25] M. Allman, V. Paxson, and C. W. Stevens. (1999, April) TCP congestion control. RFC 2581. Network working group. [Online]. Available: http://tools.ietf.org/html/rfc2581

[26] M. Allman, S. Floyd, and C. Partridge. (2002, October) Increasing TCP's initial window. RFC 3390. Network working group. [Online]. Available: http://tools.ietf.org/html/rfc3390

[27] S. Oueslati-boulahia, A. Serhrouchni, S. Tohm, S. Baier, and M. Berrada, "TCP over satellite links: Problems and solutions," *Telecommun. Syst*, vol. 13, pp. 199–212, 2000.

[28] H. Zimmermann, "OSI reference model–The ISO model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, Apr 1980.

[29] D. Roddy, *Satellite communications*, 4th ed. McGraw-Hill Professional, 2006, ch. 10.6.

[30] G. Fairhurst and L. Wood. (2002, August) Advice to link designers on link Automatic Repeat reQuest (ARQ). RFC 3366. [Online]. Available: http://www.ietf.org/rfc/rfc3366.txt

[31] *TM synchronization and channel coding*, Consultative Committee for Space Data Systems Std. CCSDS 131.0-B-1, September 2003. [Online]. Available: http://public.ccsds.org/publications/archive/131x0b1.pdf

[32] *TM Space Data Link Protocol*, Consultative Committee for Space Data Systems Std. CCSDS 132.0-B-1, September 2003. [Online]. Available: http://public.ccsds.org/publications/archive/132x0b1.pdf

[33] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, August 1996.

[34] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[35] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, 2001.

[36] D. J. C. MacKay, "Gallager codes – recent results," in *Coding, Communications and Broadcasting*, P. Farrell, M. Darnell, and B. Honary, Eds. Research Studies Press, 2000, pp. 139–150.

[37] *QNX Neutrino RTOS: System Architecture*, QNX Software Systems, September 2007. [Online]. Available: http://www.qnx.com/download/download/16854/sys_arch.pdf

[38] *Portable Operating System Interface (POSIX)*, IEEE Std. 1003.1-2008, 2008. [Online]. Available: http://www.opengroup.org/onlinepubs/9699919799

[39] P. D. W. Aerts and G. A. E. Vandenbosch, "Conceptual study of analog baseband beam forming: Design and measurement of an eight-by-eight phased array," *IEEE Transactions on Antennas and Propagation*, vol. 57, pp. 1667–1672, Jun 2009.

[40] J. S. Gilmore and R. Wolhuter, "A multichannel satellite scheduling algorithm," in *Southern African Telecommunication Networks and Applications Conference (SATNAC)*, ser. Protocols, 2008.

[41] J. S. Gilmore and R. Wolhuter, "Predicting low earth orbit satellite communications quality and visibility over time," in *Southern African Telecommunication Networks and Applications Conference (SATNAC)*, ser. Access Networks, 2009.

[42] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, pp. 287–326, 1979.

[43] P. Brucker, *Scheduling Algorithms*, 5th ed. Springer-Verlag Berlin, 2007.

[44] V. T'Kindt, J.-C. Billaut, and H. Scott, *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer, 2006, ch. 1.9.1.

[45] J. Y.-T. Leung, Ed., *Handbook of Scheduling: Algorithms, Models, and Performance analysis*. Chapman & Hall/CRC, 2004, ch. 15.

[46] O. Montenbruck and E. Gill, *Satellite Orbits: Models, Methods and Applications*, ser. Physics and astronomy. Springer, 2000, ch. 3.

[47] F. H. Ronald and R. L. Roehrich, "Spacetrack report no. 3 models for propagation of norad element sets," Proc, Tech. Rep., 1980.

[48] C. I. Coombs, *Spacetrack, Watchdog of the Skies*. William Morrow & Company, Inc., 1969.

[49] D. M. Bourg, *Physics for game developers*. O'Reilly, 2002, ch. 14.

[50] S. Aoki, H. Kinoshita, B. Guinot, G. H. Kaplan, D. D. McCarthy, and P. K. Seidelmann, "The new definition of universal time," *Astronomy and Astrophysics*, vol. 105, pp. 359–361, Jan. 1982.

[51] B. Hofmann-Wellenhof and H. Moritz, *Physical geodesy*. Birkhäuser, 2005, ch. 5.6.

[52] Sebastian Stoff. Orbitron - satellite tracking system. [Online]. Available: http://www.stoff.pl/

[53] K. Eriksson, D. J. Estep, and C. Johnson, *Derivatives and Geometry in IR 3*. Springer, 2004, ch. 21.17.

[54] World Wide Web Consortium, *Extensible Markup Language (XML) 1.1*, Std., Aug. 2006. [Online]. Available: http://www.w3.org/TR/2006/REC-xml11-20060816/

[55] D. Crockford. (2006) The application/json media type for JavaScript Object Notation (JSON). RFC 4627. JSON.org. [Online]. Available: http://tools.ietf.org/html/rfc4627

[56] Amazon.com. (2006) Amazon web services. [Online]. Available: http://aws.amazon.com/

[57] FIX Protocol, Ltd. (1992) Financial Information eXchange (FIX) protocol. [Online]. Available: http://www.fixprotocol.org/

[58] topografix.com. (2002) GPS eXchange Format (GPX). [Online]. Available: http://www.topografix.com/gpx.asp

[59] Adobe Systems. (2004) Adobe Flex. [Online]. Available: http://www.adobe.com/products/flex/

[60] RSS Advisory Board, *Really Simple Syndication (RSS) 2.0*, Harvard Std., 2009. [Online]. Available: http://www.rssboard.org/rss-specification

[61] World Wide Web Consortium, *Scalable Vector Graphics (SVG) 1.1*, Std., Jan. 2003. [Online]. Available: http://www.w3.org/Graphics/SVG/

[62] P. Saint-Andre. (2004, October) Extensible messaging and presence protocol (XMPP): Core. RFC 3920. Jabber Software Foundation. [Online]. Available: http://tools.ietf.org/html/rfc3920

[63] Yahoo! (2005) Using JSON (JavaScript Object Notation) with Yahoo! web services. [Online]. Available: http://developer.yahoo.com/common/json.html

[64] Google. (2006) Using JSON with Google data APIs. [Online]. Available: http://code.google.com/apis/gdata/json.html

[65] J. J. Garrett. (2005) Ajax: A new approach to web applications. [Online]. Available: http://www.adaptivepath.com/ideas/essays/archives/000385.php

[66] E. S. Raymond, *The Art of UNIX Programming*, ser. Professional computing. Addison-Wesley, 2003, ch. 5.

[67] N. Freed and N. Borenstein. (1996) Multipurpose internet mail extensions (mime) part one: Format of internet message bodies. RFC 2045. Innosoft and First Virtual. [Online]. Available: http://tools.ietf.org/html/rfc2045

[68] S. Josefsson. (2006) The base16, base32, and base64 data encodings. RFC 4648. SJD. [Online]. Available: http://tools.ietf.org/html/rfc4648

[69] D. Veillard. (1999) The XML C parser and toolkit of Gnome: libxml. [Online]. Available: http://xmlsoft.org/

[70] J. Clark, C. Cooper, and F. Drake. (1998) The Expat XML parser. [Online]. Available: http://expat.sourceforge.net/

[71] L. Hilaiel. (2007) Yet Another JSON Library (YAJL). [Online]. Available: http://lloyd.github.com/yajl/

[72] B. Nichols, D. Buttlar, and J. P. Farrell, *Pthreads programming.* O'Reilly, 1996, pp. 128–143.

[73] E. S. Raymond, *The Art of UNIX Programming*, ser. Professional computing. Addison-Wesley, 2003, ch. 1.6.17.

[74] I. Ali, P. G. Bonanni, N. Al-Dhahir, and J. E. Hershey, *Doppler applications in LEO satellite communication systems.* Springer, 2002, ch. 4.1.

[75] *QNX Momentics Tool Suite: IDE User's Guide*, QNX Software Systems, May 2009. [Online]. Available: http://photon.qnx.com/download/download/19493/ide_user_guide.pdf

[76] *QNX Neutrino Realtime Operating System Library Reference*, QNX Software Systems, September 2007. [Online]. Available: http://photon.qnx.com/download/download/16852/neutrino_lib_ref.pdf

[77] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, 2nd ed., ser. Business Data Processing, T. Badgett, T. M. Thomas, and C. Sandler, Eds. John Wiley and Sons, 2004, vol. 28.

[78] A. Kumar and J. St.Clair. (2005) Cunit: A unit testing framework for c. [Online]. Available: http://cunit.sourceforge.net

[79] J. Seward and N. Nethercote, "Using valgrind to detect undefined value errors with bit-precision," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 2–2.