DESIGN, DEVELOPMENT, AND TESTING OF THE COMMAND AND DATA HANDLING SUBSYSTEM FOR MICROSATELLITE MISSIONS

by

Ji Eun (Christine) Lee

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Institute for Aerospace Studies
University of Toronto

Abstract

Design, Development, and Testing of the Command and Data Handling Subsystem for Microsatellite Missions

Ji Eun (Christine) Lee

Master of Applied Science

Institute for Aerospace Studies

University of Toronto

2019

A spacecraft is composed of multiple subsystems that work together to accomplish the mission objectives. One of the most vital subsystems is termed the "brain" of the spacecraft, more formally known as the command and data handling subsystem, which is responsible for commanding all other subsystems and managing all data sent and received by the spacecraft. This thesis describes the author's contributions to the various aspects of the command and data handling subsystem for two microsatellite missions, with focus on work that can be extended for use and implemented on future space missions at the Space Flight Laboratory. Particular emphasis is given to the author's work on design of a new serial interface board, optimization of both ground and satellite software related to payload data handling, and development of various application threads on satellite on-board computers and ground control interfaces.

Acknowledgments

First and foremost, I would like to acknowledge and thank Dr. Robert Zee for providing me with the unique and wonderful opportunity to study and work at the Space Flight Laboratory (SFL) of the University of Toronto Institute for Aerospace Studies. I am extremely grateful for all that you have done and continue to do to in order to provide such an exceptional workplace environment for students, like myself, to learn and grow while working on actual space missions. In particular, thank you for the financial support that has allowed me to pursue my Master's, building and maintaining a team of amazing staff that I have had the pleasure of working with and learning from, and the unparalleled opportunity to work on various space missions. In addition, thank you for all the constructive feedback and guidance you have provided for my ongoing work and master's thesis.

Secondly, I would like to express gratitude to all my awesome colleagues and fellow students at SFL. A huge thank you to the managers of the missions I have worked on: Simon Grocott and Brad Cotten for your guidance, challenges, and support. Thank you so much to Paul Choi for being the best mentor anyone could ask for and always taking the time to help out, even with your busy schedule; I honestly could not have made it this far without your selfless support. Thank you to Daniel Kekez for always taking the time to answer my questions, being willing to help out with any tasks I had trouble with, and knowing everything and anything at any time. Thank you to Nicolaas Handojo for all your help and guidance during my time on the NEMO-HD mission. Thank you to Shariann for always being so supportive and making sure all the students are well and healthy.

Last, but certainly not least, I would like to show my upmost gratitude to my friends and family outside of SFL. Thank you in particular to Johnny Wang, Hao Xing, and Christopher Lie Ken Jie for your support and taking the time to read over my thesis. Thank you to my older brother for always providing me with guidance and advice. Thank you to both my mom and dad for everything you have done and continue to do: the sacrifices you have made, teaching me invaluable lessons in life that have made me who I am today, your never-ending support and love, and for staying strong through the toughest of times.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

ACK	**Ack**nowledgement/**Ack**nowledged
ADC	**A**nalog-to-**D**igital **C**onverter
ADCC	**A**ttitude **D**etermination and **C**ontrol **C**omputer
AIS	**A**utomatic **I**dentification **S**ystem
BER	**B**it **E**rror **R**ate
Bps	**B**ytes **P**er **S**econd
bps	**B**its **P**er **S**econd
C&DH	**C**ommand and **D**ata **H**andling
CAN	**C**ontroller **A**rea **N**etwork
CANOE	**C**anadian **A**dvanced **N**anosatellite **O**perating **E**nvironment
CLARA	**C**ompact **L**ightweight **A**bsolute **RA**diometer
COTS	**C**ommercial-**O**ff-**T**he-**S**helf
CRC	**C**yclic **R**edundancy **C**heck
FFI	Norwegian Defence Research Establishment
GUI	**G**raphical **U**ser **I**nterface
HR-HD	**H**igh-**R**esolution – **H**igh-**D**efinition
HRS-PAN	**H**igh-**R**esolution **S**pectral – **Pan**chromatic
HRS-MS	**H**igh-**R**esolution **S**pectral – **M**ulti**s**pectral
I^2C	**I**nter-**I**ntegrated **C**ircuit
ICD	**I**nterface **C**ontrol **D**ocument
IMO	**I**nternational **M**aritime **O**rganization
LFFT	**L**ong **F**orm **F**unction **T**est
LR-HD	**L**ow-**R**esolution – **H**igh-**D**efinition
LTAN	**L**ocal **T**ime **A**scending **N**ode
NACK	**N**on-**ack**nowledgement/**N**ot **Ack**nowledged
NCA	**N**orwegian **C**oastal **A**dministration
NEMO-HD	**N**ext-Generation **E**arth **M**onitoring and **O**bservation – **H**igh **D**efinition

NIR	**N**ear **I**nfra**r**ed
NRD	**N**avigation **R**adar **D**etector
NSC	**N**orweigan **S**pace **C**entre
NSP	**N**ano**s**atellite **P**rotocol
OBC	**O**n-**B**oard **C**omputer
OS	**O**perating **S**ystem
P/F	**P**oll/**F**inal
PER	**P**acket **E**rror **R**ate
PMTP	**P**ayload **M**ass **T**ransfer **P**rogram
POBC	**P**ayload **O**n-**B**oard **C**omputer
PPS	**P**ulse **P**er **S**econd
QMTP	**Q**t-based **M**ass **T**ransfer **P**rogram
SCC	**S**erial **C**ommunications **C**ontroller
SFIC	**S**pacecraft **F**rame **I**nterface **C**ontroller
SFL	**S**pace **F**light **L**aboratory
SIB	**S**erial **I**nterface **B**oard
Space-SI	Slovenian Centre of Excellence for Space Sciences and Technologies
SPI	**S**erial **P**eripheral **I**nterface
SSO	**S**un-**S**ynchronous **O**rbit
TNC	**T**erminal **N**ode **C**ontroller
UART	**U**niversal **A**synchronous **R**eceiver/**T**ransmitter
UHF	**U**ltra **H**igh **F**requency
UTIAS	**U**niversity of **T**oronto **I**nstitute for **A**erospace **S**tudies
VHF	**V**ery **H**igh **F**requency
WOD	**W**hole **O**rbit **D**ata

Chapter 1

Introduction

Today, there are over 1500 operational man-made satellites orbiting the Earth [1]. While people may not constantly think about their impact on society, satellites play an integral role in our daily lives. Every day, these satellites provide information and services to support multiple applications, including, but not limited to, Earth observation, global communications, environment, safety, navigation, and security [2]. One of the many barriers associated with spacecrafts is the high cost and lengthy schedules associated with the build and launch. This is mainly attributable to the conventional and procedure-heavy method of development, which focuses on risk aversion as opposed to managed risk [3]. What if there was a way of carrying out these crucial space missions without the excessive cost and protracted schedules? More satellites could then be built, resulting in more scientific discoveries, more services to society, and more advancement all around.

With the rapid growth of space missions involving nanosatellites and microsatellites in the recent years, it is becoming more widely accepted that smaller satellite missions, with lower costs and faster schedules, can accomplish as much as, if not more than, conventional big space missions in a growing number of areas. Founded in 1998, the Space Flight Laboratory (SFL) at the University of Toronto Institute for Aerospace Studies (UTIAS) is a global leader in nanosatellite and microsatellite missions (satellites under 10 kg and 100 kg, respectively) [4], recognized for its ultra-low cost, yet high performance missions. SFL's mission is to lower the entry barrier to space to allow more productive use of space for the future. SFL applies the Microspace Philosophy to their spacecraft design and development processes to achieve bigger returns with smaller satellites.

1.1 The Microspace Philosophy

The Microspace Philosophy emphasizes the approach of achieving the lowest cost possible to accomplish high-value space objectives without sacrificing quality or introducing excessive risk [5]. This approach is executed using:

- typically modern, high-tech, commercial-off-the-shelf (COTS) components,
- embedded systems with more complex software, which allows reduction in sizes of these systems, and
- a bottom-up approach that informs and moderates the traditional top-down approach, which allows the conventional requirements- and performance-driven design to be viewed with a capabilities- and cost-driven mindset [6].

The benefits of the Microspace approach are as follows:

1) For a given level of functionality, overall spacecraft size is reduced as a result of emphasizing the use of COTS technology and embedded systems with more complex software.
2) Engineers involved in the missions gain experience much more quickly and frequently, as they will likely see their work reach space annually compared to every decade [3].
3) For a given cost, a larger number of smaller and less expensive satellites can be produced, which in turn activates and promotes a virtuous cycle, resulting in an increase in demand and reliability of similar future missions [7].

This philosophy is established on the belief that cheaper and faster missions lead to more accomplishments in space [3]. This helps to increase not only demand, but also confidence in more space missions. Leveraging the latest advances in commercial technologies in order to provide a performance advantage in space for future space-based data users, SFL is at the forefront of employing the Microspace Philosophy in space mission design [5].

1.2 NEMO-HD and NorSat-3 Microsatellites

This thesis describes the author's contributions to the command and data handling subsystem on two microsatellite missions:

- the Next-Generation Earth Monitoring and Observation – High Definition (NEMO-HD) mission, and
- the NorSat-3 mission.

This section provides a general outline of both missions as well as a brief overview of the command and data handling subsystem.

1.2.1 NEMO-HD Mission

The Next-Generation Earth Monitoring and Observation – High Definition (NEMO-HD) satellite is an Earth-observation microsatellite developed for the Slovenian Centre of Excellence for Space Sciences and Technologies (Space-SI). NEMO-HD is Slovenia's first satellite for high-definition imaging and video from Earth orbit, leveraging SFL's NEMO technology to produce a high-performance imaging and video mission. The main objectives of the mission are to provide moderate- to high-resolution Earth imagery in a number of bands and transmit real-time high-definition videos with both wide and narrow fields of view.

1.2.2 NorSat-3 Mission

The NorSat-3 satellite is a maritime monitoring microsatellite developed for the Norwegian Space Centre (NSC), following from the highly successful AISSat-1, -2, and -3 and NorSat-1 and -2 satellites also built by SFL. NorSat-3 is, therefore, the sixth Norwegian satellite built by SFL for the purpose of providing governmental users in Norway with data that are crucial for assuring protection and safety in Norwegian waters. This satellite is designed to capture signals from frequencies allocated for civil navigational radars by the International Maritime Organization (IMO), and demonstrate an experimental payload to augment ship detection capabilities from its Automatic Identification System (AIS) receiver.

1.2.3 Command and Data Handling Subsystem

A typical spacecraft is composed of a payload subsystem – the main subsystem used to fulfill the mission objective(s) – and the bus or platform – which consists of various subsystems that support the payload(s) in its operations. One of these crucial subsystems is what is termed the 'brain' of the spacecraft: the command and data handling (C&DH) subsystem. The main function of this subsystem is to perform on-board operations, process various types of data, and manage all internal communications between the various subsystems on the spacecraft [8]. Simply put, it is responsible for handling all the data sent and received by the spacecraft, as well as commanding the different subsystems accordingly.

Closely associated with the C&DH subsystem, is the ground segment software, which refers to all the ground-based software elements of a spacecraft used by the operators during the mission. The ground segment software allows ground control of the spacecraft as well as management of the spacecraft data [9]. Depending on the role and corresponding functions, ground segment software can fall under the following categories:

- **Control software:** programs used for interacting with the satellite or with any subsystem(s) of the satellite.
- **Planning software:** programs used for generating sets of commands for the satellite.
- **Monitoring software:** programs used for autonomously monitoring specific parameters (i.e. telemetry) of any subsystem(s) of the satellite [10].

While ground segment software is not typically considered a subsystem of a spacecraft, the control-related ground segment software discussed in this thesis is regarded as a part of the overall C&DH subsystem. Although well-established, there is always room for improvement and development in software, and this thesis touches on some of significant aspects of the overall C&DH subsystem that the author has contributed to during her time on the two missions.

1.3 Thesis Objectives and Outline

This section explains the state of work on both the NEMO-HD and NorSat-3 satellites at the time the author joined the respective missions, introduces the resulting thesis objectives for each, and provides an outline of this thesis document.

1.3.1 NEMO-HD: State of Work and Thesis Objectives

From a general mission perspective, at the time the author started working on the NEMO-HD mission, the overall design and satellite build was complete, with the exception of the primary payload integration which was completed in parallel to the author finishing her work on the mission. From a C&DH subsystem perspective, the software for both the satellite computers and ground segment programs were functionally complete. However, the traditional method used at SFL for requesting and downlinking payload data was observed to be insufficient for this particular mission.

Compared to previous SFL missions, NEMO-HD is an exceptionally data-intensive mission, which means large amounts of payload data are expected to be collected and required to be downlinked to ground. Due to a combination of factors, such as the significant difference in the downlink and uplink rates (50 Mbps vs. 4 kbps, respectively), the large amounts of data collected and required for downlink, and the lossy communication that naturally exists between the satellite and its ground station(s), the traditional method used at SFL for requesting and downlinking data – on this particular mission – results in an effective downlink data rate that is heavily reliant on the uplink rate. The effective data rate, which is explained in greater detail in **Section 3.5**, is defined as the total size of files requested for downlinking divided by the time taken to download all the requested files successfully. The main issue motivating the author's work is that the mission is not capable of utilizing the high-speed downlink rate to its full potential; instead, the rate becomes heavily dependent on how fast, or in this case, how slow the requests for the files and their missing segments can be uplinked to the satellite from ground. At such a detailed and advanced phase in the development process, investing in a new uplink system capable of faster rates would have added significant cost and delay to the mission. Therefore, the preferred approach was to improve the overall process via software. This thesis explains the baseline algorithm that was initially developed and implemented to address this issue, then details the author's contributions and corresponding results to improve the effective data rate.

The NEMO-HD portion of this thesis presents the author's work on configuring existing software programs to achieve a target effective data rate of 24 Mbps, specified in **Section 3.5**, for the payload data downlinking portion of the mission in the worst-case scenarios. The high-level thesis objectives for the NEMO-HD portion include:

- locating and fixing causes of delay observed in the ground segment software,
- configuring, as necessary, the algorithm used by the ground segment software program and satellite computers responsible for executing the payload data request and downlink,
- finding the parameter values required to achieve the target effective data rate, and
- performing tests to prove requirement compliance.

Although future missions may employ uplink systems with higher rates than NEMO-HD's, since the ground will always have more power available than a satellite would, it is typical for satellites to have a downlink rate that is faster than the uplink rate. As data becomes more valuable and advances in technology allow satellites to be capable of collecting more and more data, the author's work on the NEMO-HD mission provides an extendable and configurable solution for ensuring that a satellite's effective downlink data rate does not become significantly hindered by the relatively slower uplink rate.

1.3.2 NorSat-3: State of Work and Thesis Objectives

As the NorSat-3 satellite utilizes the heritage platform from its predecessors NorSat-1 and -2, at the time the author started working on this mission, the overall design was complete with the exception of updates required to accommodate the devices unique to NorSat-3. Similarly, from a C&DH subsystem perspective, the approach was to use software that was used on NorSat-1 and -2, incorporating modifications required to accommodate the devices unique to NorSat-3.

The NorSat-3 portion of this thesis presents the author's work on designing, developing, and testing the C&DH subsystem for the satellite, which involves C&DH hardware as well as software for the ground station programs and the satellite's on-board computers. The high-level thesis objectives for the NorSat-3 portion include:

- designing, manufacturing, and testing the serial interface board,
- developing the interface and software for retrieving telemetry values of a new receiver,
- developing the interface and software for the new payload on NorSat-3,
- developing the interface and software for handling commands that cannot be performed using the standard programs and application software.

Although a significant portion of the author's work on NorSat-3 also included developing and performing the long form function test for the overall spacecraft and configuring software for all three on-board computers, this thesis focuses on the more innovative and original contributions that can be extended for use on future missions.,

As mentioned above, the author's contributions to the NorSat-3 mission discussed in this thesis presents work that can be easily configured and used on future SFL missions. New developments, both hardware and software, that can be reused on other missions are extremely valuable in the space industry as they allow more time and money to be allocated for new work. From a hardware perspective, using hardware that has been approved, qualified, and flown on a previous mission significantly reduces the resources spent on review processes and testing for that component. In addition, future missions can improve on the design, if necessary, based on lessons learned from previous missions that have used that hardware. From a software perspective, incorporating approved code allows more time to be allocated for improvements and optimizations based on its previous performance.

1.3.3 Thesis Outline

This thesis describes the author's contributions to the NEMO-HD and NorSat-3 missions, focusing on the objectives outlined in **Section 1.3.1** and **Section 1.3.2**, and provides appropriate background information to help the reader understand the purpose and motivation behind the author's work .

In terms of the overall outline, following this introduction, **Chapter 2** provides more information on the command and data handling subsystem, specific to SFL, highlighting areas that are pertinent to the author's work presented in this thesis. **Chapter 3** and **Chapter 5** provide overviews of the NEMO-HD and NorSat-3 missions, respectively. In both chapters, mission requirements and mission information relevant to the author's work are presented. **Chapter 4** and **Chapter 6** describe the author's responsibilities on and contributions to the NEMO-HD and NorSat-3 missions, respectively. In both chapters, the author's mission-specific thesis objectives are reiterated in more detail and her mission-specific work and results are presented. **Chapter 7** summarizes the author's objectives and work on the respective missions and presents the author's closing remarks.

Chapter 2

Background: Command and Data Handling

The command and data handling (C&DH) subsystem is responsible for managing all data sent and received by the spacecraft, which comprises payload data, various subsystem data, and commands [11]. From an uplink perspective, where the ground is sending commands and/or data to the satellite, the C&DH subsystem is responsible for receiving that information from the communications subsystem, decoding them, and then either executing them or directing them to the appropriate subsystem(s). From a downlink perspective, where the satellite is sending information to the ground, the C&DH subsystem is responsible for collecting and handling different types of data obtained from the various spacecraft subsystems, including the payload(s), then routing them to the communications subsystem to downlink to Earth. The C&DH subsystem is also responsible for collecting telemetry from all the various devices on the satellite as well as commanding the attitude determination and control subsystem, which is responsible for attitude control functions.

This chapter provides background information on the C&DH subsystem, specific to SFL, predominantly focusing on areas pertinent to the author's work discussed in this thesis.

2.1 Nanosatellite Protocol

The nanosatellite protocol (NSP) is a specially designed protocol used in communication links between:

- ground software,
- ground and spacecraft software,
- computers on the spacecraft, and
- threads on the spacecraft.

NSP was originally designed for use on low-bandwidth radio communication links between a spacecraft and its respective ground station(s), but has since been adapted for use over various wired links and for intra-computer message handling between threads [12]. The devices on both the NEMO-HD and NorSat-3 satellites as well as the threads and terminal programs outlined in this thesis communicate via NSP, unless explicitly specified otherwise.

2.1.1 Nanosatellite Protocol Packet

A NSP packet, shown in **Figure 1**, has a minimum size of 5 bytes, where 3 bytes are allocated for the header, 0 or more byes for packet data, and 2 bytes for cyclic redundancy check (CRC).

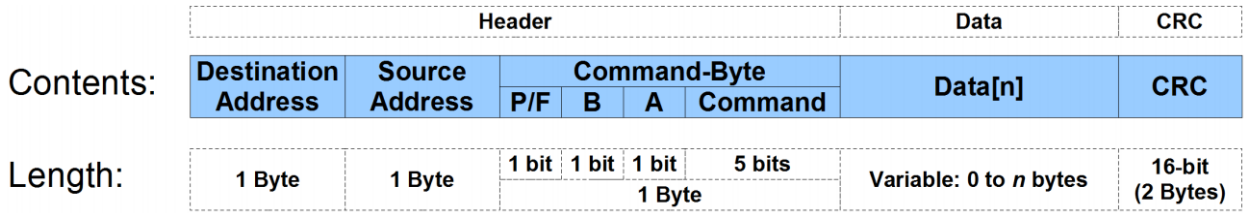

Figure 1: Nanosatellite Protocol (NSP) Packet [12]

The various contents of a NSP packet are as follows:

- **Destination Address:** This single-byte field identifies the destination computer, device, or application thread on a satellite, or a terminal program on the ground station.
- **Source Address:** This single-byte field identifies the source computer, device, or application thread on a satellite, or a terminal program on the ground station.

- **Command-Byte:** This single-byte field is divided into four sub-fields: a bit for poll/final (P/F-bit), a bit for packet identifier (B-bit), a bit for acknowledgement (A-bit), and the remaining 5 bits for specifying the command.
  - **Poll/Final Bit (P/F-bit):** This single-bit field represents a poll-bit in uplink packets and a final-bit in telemetry downlink packets. An uplink packet with P/F-bit set to '1' indicates that a reply is expected from the recipient. A downlink packet with P/F-bit set to '1' indicates that this packet is the last one in a sequence of packets, which occurs in bulk-download. Once a master receives a response packet with the P/F-bit set to '1', it is able to send another command.
  - **B-bit:** This bit alternates for every command sent from the ground. In cases where a command needs to be resent and a reply is received, this bit helps the operators determine whether the received reply was to the original command or to the resent command. This information can then be used for debugging purposes to verify if a problem in the software is the cause of delayed replies.
  - **A-bit:** This bit represents acknowledgement. A response packet with the A-bit set to '1' indicates that the command has been acknowledged (ACK), while an A-bit set to '0' indicates that the respective command has not been acknowledged (NACK).
- **Data:** This variable-length field may contain nothing, additional parameters in command packets, or requested data in response packets. Although there is no hard constraint on the maximum length of this field, the typical maximum length used is 260 bytes. This preferred size mainly stems from bit error rates, explained in **Section 2.4**, associated with radio communications. As the packet size increases, the chance of a bit in the packet being corrupted increases, which means the entire packet becomes corrupted. On the other hand, as the packet size decreases, the larger the percentage of the packet that the packet overhead – the 5 bytes discussed above – expends, which decreases the effective data rate. Previous analyses have concluded that 260 bytes is a good maximum length.
- **CRC:** This 2-byte field is used for error checking, which allows frames with incorrect CRCs to be silently dropped by the receiving device.

For terminology, a frame refers to a packet in its entirety including any CRC fields and delimiters, while a packet refers to the remaining contents of a frame after encapsulation has been removed; a packet is therefore a subset of a frame.

2.1.2 Nanosatellite Protocol for Radio Links

Every spacecraft flown is assigned a unique 16-bit high-level data link control (HDLC) address. Specific to communication between a satellite and its respective ground station(s), radio links make use of HDLC framing – a standard packetization protocol for serial links. Packetization, as shown in **Figure 2**, involves prepending and appending a framing byte (binary "01111110" or hexadecimal 0x7E), 0-bit insertion (bit-stuffing) after five consecutive 1-bits, and appending a 16-bit CRC onto the packet. When using HDLC framing, since the framing bytes used to indicate the start and end of a packet consists of six consecutive 1-bits, bit-stuffing is required to ensure that the actual data within the packet does not contain more than five 1-bits in a row and be accidentally processed as a framing byte. Bit-stuffing ensures that any time five consecutive 1-bits are present in the data being transmitted, a 0-bit is inserted. The receiving device is aware of the bit-stuffing process and, therefore, strips out all the 0-bits that were inserted for this purpose before passing the data to the threads on the satellite computers.

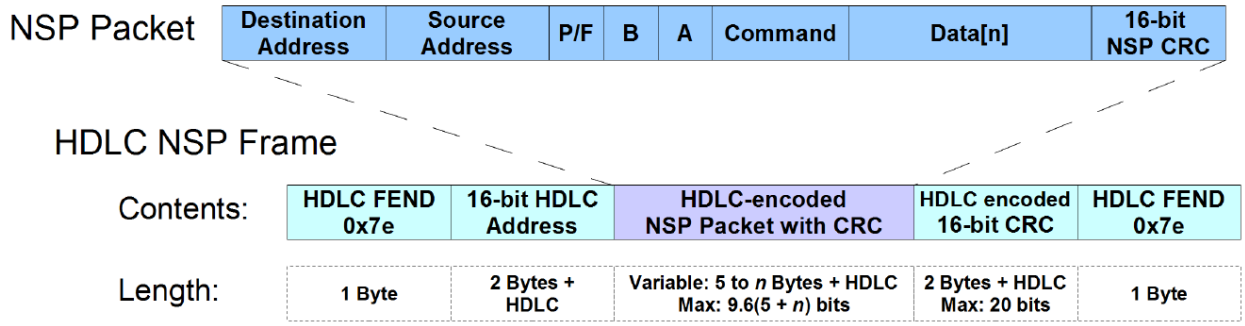

Figure 2: HDLC Framing of NSP Packet

On ground, the HDLC packetization and depacketization is handled by the Spacecraft Frame Interface Controller (SFIC), also referred to as the Terminal Node Controller (TNC) on the NEMO-HD mission, which is described in greater detail in **Section 2.3.4**. On the spacecraft, the HDLC framing and deframing is handled by a Serial Communications Controller (SCC), which is included in the design of the SFL on-board computers.

2.2 On-Board Computer Designs

This section provides high-level information on the on-board computers (OBC) discussed in this thesis, specifically, the standard OBCs used on SFL satellites and the payload OBCs specific to the NEMO-HD mission.

2.2.1 SFL On-Board Computers

At SFL, a satellite's C&DH subsystem is centered around a number of OBCs – all identical from a hardware perspective – that act in concert to provide housekeeping, telemetry collection, attitude determination and control, and payload data handling functionality to perform the functions required to achieve the mission objectives [13]. The main OBCs discussed in this thesis follow a heritage design that has been flown on many SFL missions, shown in **Figure 3**.

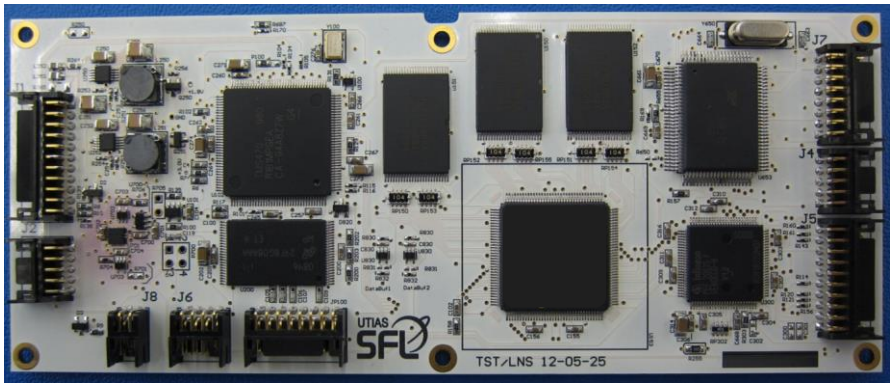

Figure 3: SFL On-board Computer As-Built Hardware

The NorSat-3 satellite consists of three SFL OBCs: housekeeping, attitude determination and control, and payload, which are predominantly differentiated by their application software:

1) **Housekeeping Computer (HKC):** responsible for basic housekeeping tasks including, but not limited to, periodic collection of spacecraft telemetry, execution and logging of time-tagged commands, routing of commands to other OBCs and spacecraft hardware, and relaying packets through the radio links.
2) **Attitude Determination and Control Computer (ADCC):** responsible for the attitude determination and control system functionality of the spacecraft and logging of that data.
3) **Payload On-board Computer (POBC):** responsible for interfacing with the payloads on the spacecraft as well as collecting and handling payload data.

The OBC software comprises a two-level approach: bootloader and application software:

- The **bootloader** software is run after a power-up, exception, or reset, and is identical for all OBCs from a functional perspective. Its main function is to initialize all hardware to enable communication between the spacecraft and ground, thus allowing the ground to upload software and start it. It is also capable of basic safe-hold monitoring functionalities such as access to telemetry and low-level access to the OBC for debugging purposes.
- The **application software** is loaded by ground command and is specific for each OBC. The application software consists of a real-time, multi-tasking operating system (OS) called Canadian Advanced Nanosatellite Operating Environment (CANOE), which provides all the functionality required by the spacecraft. There are multiple application threads on the OS which carry out specific tasks required for the type of OBC.

2.2.1.1 Relevant Communication Interfaces

A list of the various communication channels that the OBC design includes is as follows:

- **Controller Area Network (CAN):** multi-master serial communication protocol. The CAN interface is used to communicate with the satellite's power system and all other OBCs [13].
- **Inter-integrated Circuit (I^2C):** synchronous 2-wire bus, where one wire acts as a data line and the other as a clock line. Each OBC contains two I^2C peripherals, which are typically used for attitude sensors [14].
- **Serial Peripheral Interface (SPI):** single-master serial communication protocol. Each OBC contains two SPI ports used to interface with SPI devices, which do not use NSP for communications.
- **Synchronous Serial:** supplied via the SCC, which, as mentioned in **Section 2.1.2**, is responsible for the implementation of the HDLC protocol required for radio communication [15]. As such, the synchronous serial channels are primarily used for communication with the ground over the satellite's receiver(s) and transmitter(s).
- **Universal Asynchronous Receiver/Transmitter (UART):** serial 2- or 4-line interface, where one line is dedicated for receiving and another for transmitting. Each OBC contains ten UART channels, where two are located on the processor itself and eight are provided by an eight-channel UART device [13].

2.2.2 NEMO-HD Payload On-Board Computers

The NEMO-HD satellite consists of two SFL OBCs, HKC and ADCC, and different POBCs based on COTS single-board computers. The POBCs specific to the NEMO-HD satellite are based off a x86 single-board computer which contains three levels of software: bootloader, application, and recovery [16]. There is one POBC per imaging channel on the NEMO-HD satellite, as explained in **Chapter 3**, summing up to 5 POBCs in total. GRUB (GRand Unified Bootloader), an open source bootloader commonly used to load Linux on x86 platforms, was selected as the bootloader for the NEMO-HD POBCs. Both the recovery and application kernels, described below, are Linux-based.

At a high-level:

- The **bootloader**, which is the first portion of software to execute at a power-up or reset, performs hardware initialization tasks and loads the application Linux kernel.
- The **application** software has been implemented on top of the OS, and includes functions such as configuring the cameras, capturing image from the cameras, and downlinking saved image data to the ground.
- The POBCs are to be booted from the **recovery** kernel when the application kernel requires an update or if a fault develops in the file system. The recovery kernel contains the file transfer, partitioning, and formatting tools necessary to rectify the issue.

2.3 Ground Software and Equipment

There are standard SFL ground software programs used for satellite operations, which are explained briefly in the following subsections and described in greater detail throughout this thesis.

2.3.1 NEMO Control

NEMO Control is a real-time command and control interface for all the devices on the satellite. This is the main program used during mission operations which the operator can use to control all functions related to the satellite's housekeeping, power system, payload, and attitude control. A screenshot of the NEMO Control program for the NorSat-3 mission is shown in **Figure 4**.

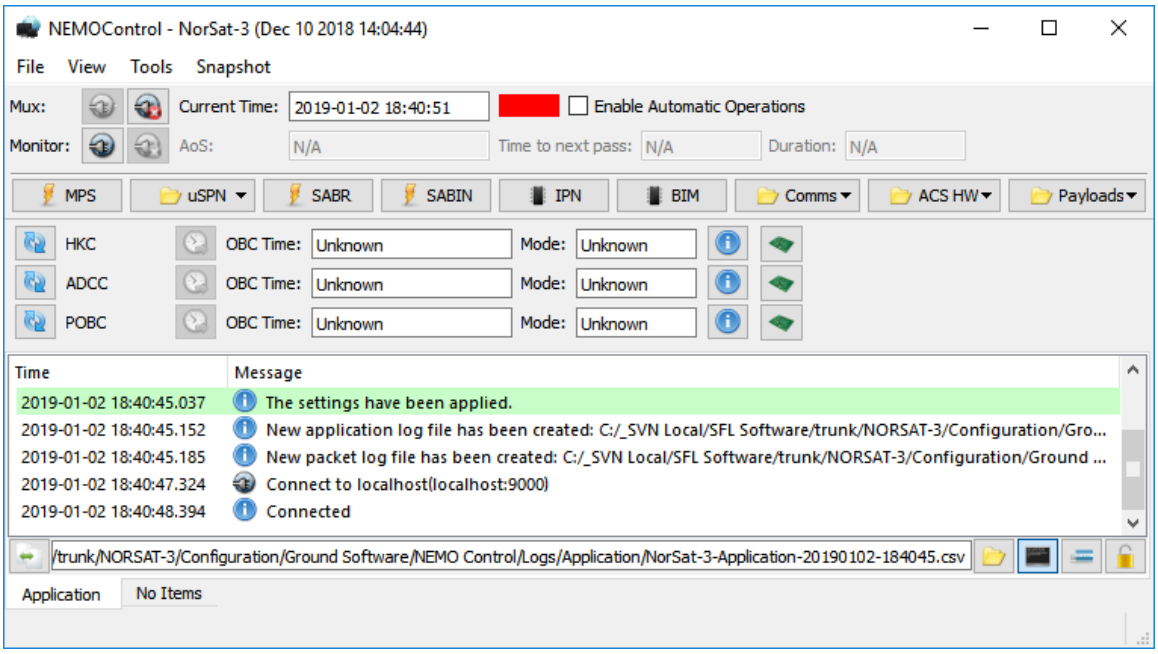

Figure 4: NorSat-3 NEMO Control Interface

2.3.2 Payload/Qt-based Mass Transfer Program (PMTP/QMTP)

Qt-based Mass Transfer Program (QMTP), shown in **Figure 22**, is a real-time interface for the file systems on the OBCs. It allows operators to:

- view the files and their states on each of the OBC file systems,
- upload/download files to/from any of the OBC file systems, and
- delete any files on any of the OBC file systems.

A common use of this program is to automatically download data from the OBCs during a pass. The types of data include, but are not limited to, whole orbit data (WOD) logs, which contain periodically collected telemetry values of all devices on the spacecraft, and payload data logs, which contain data collected by the payload(s). For the NorSat-3 mission, QMTP is used for interfacing with the file system on all three OBCs: HKC, ADCC, and POBC.

As mentioned in **Section 2.2.2**, the NEMO-HD POBCs use different hardware than the typical OBCs on SFL satellites. As a result, a program called Payload Mass Transfer Program (PMTP), based off of QMTP, exists specifically for the NEMO-HD mission. PMTP is a real-time interface, shown in **Figure 5**, for the five POBCs on NEMO-HD, which allows the operator to:

- view the files and their states on each of the POBC file systems,
- upload/download files to/from each of the POBC file systems, and
- delete any files on any of the POBC file systems.

Therefore, for the NEMO-HD mission, QMTP is used for interfacing with the file system on the HKC and ADCC, while PMTP is used for interfacing with the file system on each of the 5 POBCs.

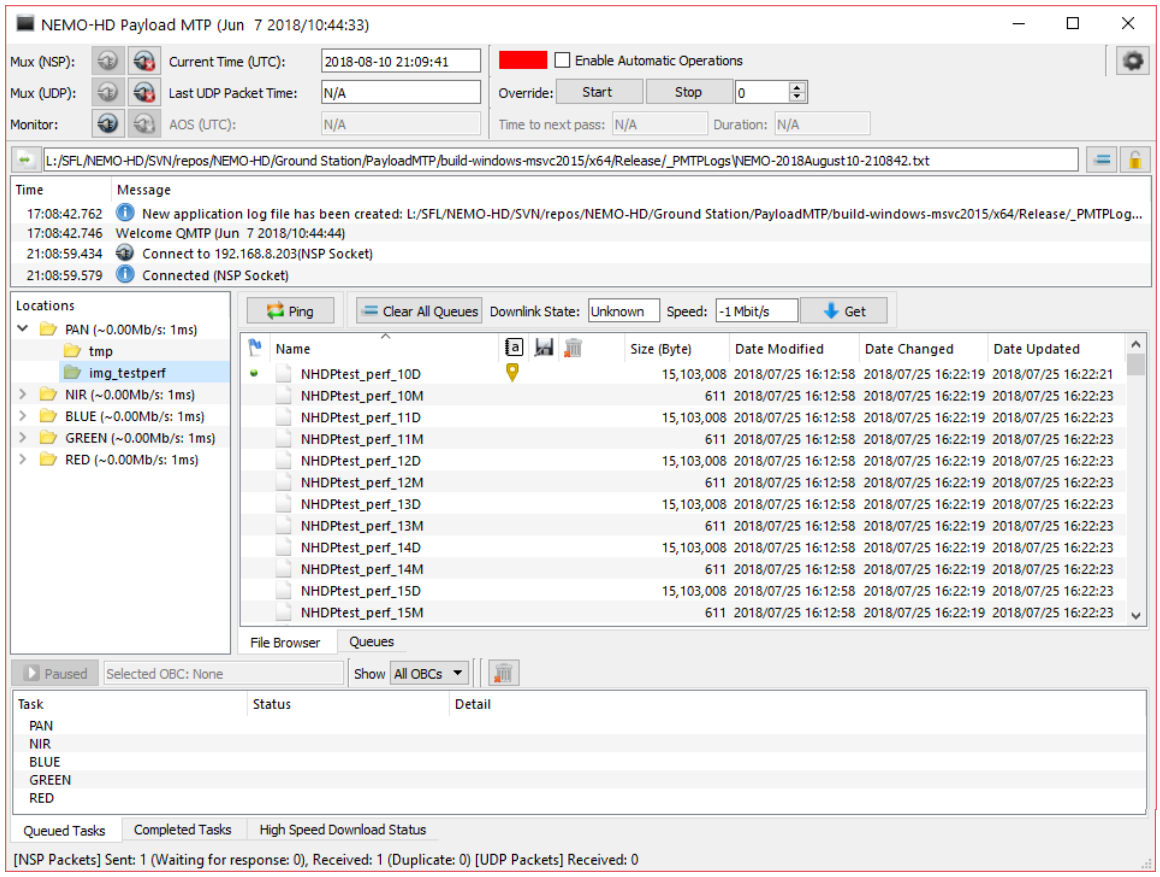

Figure 5: PMTP Interface for NEMO-HD

2.3.3 Mux Programs

There are three Mux programs called: MuxMaster, MuxStation, and MuxControl. The MuxMaster and MuxStation programs, shown in **Figure 10**, work in pairs to forward and route packets between the terminal programs on ground (NEMO Control, QMTP, etc.) and the satellite. The MuxControl program allows the operator to select the type of communications they wish to use for the downlink and uplink.

2.3.4 Spacecraft Frame Interface Controller

The Spacecraft Frame Interface Controller (SFIC), referred to as the Terminal Node Controller (TNC) on the NEMO-HD mission, is a device which provides an interface between the ground signal chain and the Mux programs. At a high-level, the SFIC is responsible for the routing and packetization/depacketization of packets sent and received via radio, i.e., HDLC-framed packets. Each satellite is assigned a dedicated SFIC and a unique HDLC address. As such, only packets with the appropriate HDLC framing and CRC values received by the SFIC are forwarded to the MuxStation.

2.4 Bit Error Rate

Bit error rate (BER) is a key parameter used to assess the full end-to-end performance for digital communications, including the transmitter, receiver, and the medium. BER is defined as the number of bits that are incorrectly received (number of bits with errors) divided by the total number of transmitted bits [17]. The rate is typically expressed as 10 to the negative power. The equation used to calculate BER is as follows:

$$\mathbf{BER} = \frac{\boldsymbol{b_e}}{\boldsymbol{b_t}} \tag{1}$$

where b_e is the number of bits with error (incorrectly received) and b_t is the total number of bits transmitted. For instance, a BER of 10^{-5} indicates that out of 100,000 bits transmitted, one bit was incorrectly received (in error).

Chapter 3

NEMO-HD: Background

This chapter provides an overview of the NEMO-HD mission, focusing on areas of specific relevance to the author's thesis scope. This includes mission requirements applicable to the author's work, details on the payload subsystem, a high-level look at the mission's observation concept, and the main challenge the author contributed to for the NEMO-HD portion of the thesis.

3.1 NEMO-HD Mission Overview

Next-Generation Earth Monitoring and Observation – High Definition (NEMO-HD) is a high performance, multispectral, Earth-observation microsatellite developed for the Slovenian Centre of Excellence for Space Sciences and Technologies (Space-SI). SFL is responsible for the designs of both the spacecraft bus and the payload subsystem, while Space-SI is responsible for the experimental payload and data processing. NEMO-HD is expected to be launched in 2019 by Arianespace on the Vega launch vehicle as part of the Small Spacecraft Mission Service Proof of Concept flight [18]. It will operate in a sun-synchronous orbit (SSO) with a 10:30 local time ascending node (LTAN) and altitude between 600 – 700 km [19].

As Slovenia's first satellite for high-definition imaging and video from Earth orbit, NEMO-HD represents a breakthrough in next-generation microsatellite missions [20]. NEMO-HD leverages SFL's NEMO technology in order to produce a high-performance imaging and video mission – something that was previously only possible on much larger platforms – in a small platform and

at a fraction of the conventional cost. This microsatellite is designed to provide high-definition, multispectral imaging capabilities with real-time interactive command and control. The resulting main objectives of the mission are to provide moderate- to high-resolution Earth imagery in a number of bands and transmit real-time high-definition videos with both wide and narrow fields of view. To achieve these objectives, the NEMO-HD payload comprises two instruments: a primary payload for narrow-field and high-resolution imaging and video, and a secondary payload for wide-field and low-resolution imaging and video. The spacecraft weighs approximately 70 kg and takes the form of an octagonal prism, as shown in **Figure 6** and **Figure 7**, with an envelope of 37.9 cm x 57.3 cm x 57.3 cm (not including the ultra high frequency (UHF) antennas).

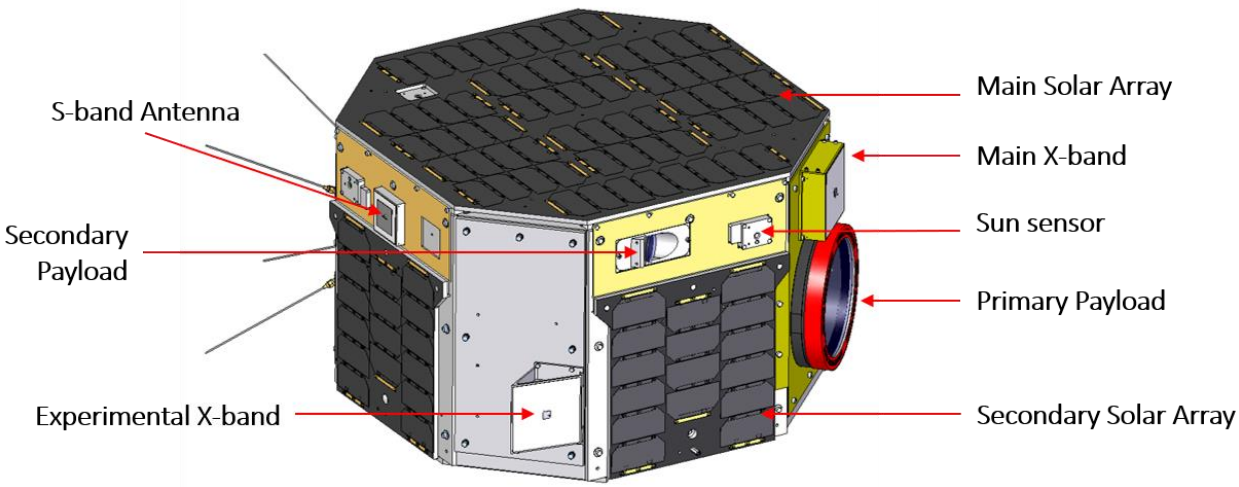

Figure 6: NEMO-HD System (Front View) [21]

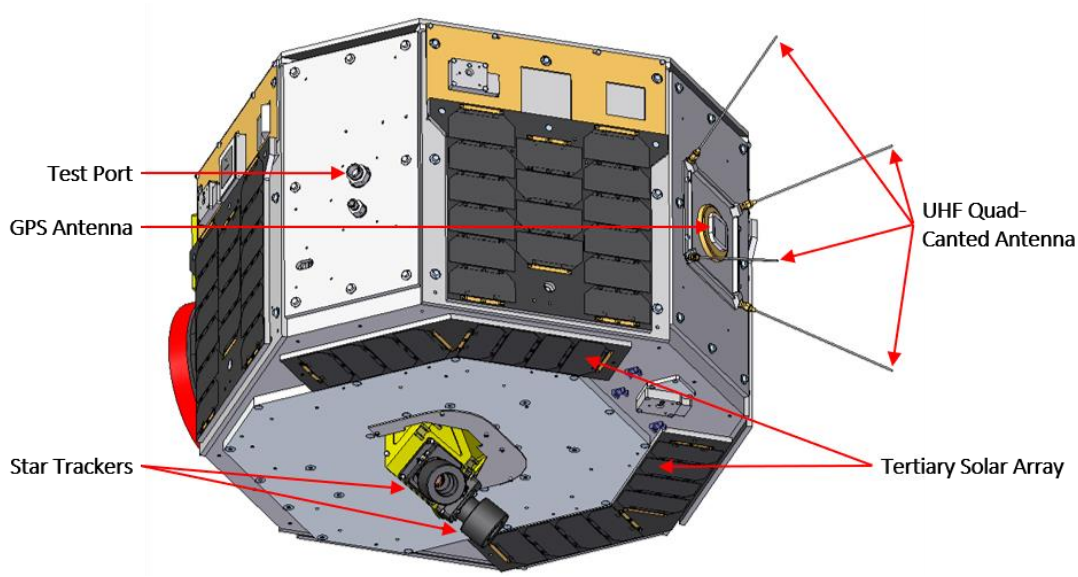

Figure 7: NEMO-HD System (Back View) [21]

3.2 Relevant Mission Requirements

The NEMO-HD requirements relevant to the author's work on the mission are listed in **Table 1**.

Table 1: Relevant NEMO-HD Requirements

REQUIREMENT NUMBER	REQUIREMENT
SFL-NHD-PAY-R001-R48	The instrument shall capture level 0 images [1]of no less than one standard scene, as defined in the NASA 2006 Earth Science Reference Handbook, from the HRS-PAN and HRS-MS imagers. These images **shall** be **downlinked and processed within no more than 10 minutes after image acquisition**. [22]
SFL-NHD-OPT-R01-1.2.11	The instrument **shall** achieve the **following photometric signal to noise ratio (SNR)** for each of the channel: **panchromatic SNR of 75**, **blue SNR of 75**, **green SNR of 75**, **red SNR of 75**, and **near-infrared SNR of 75**. The photometric SNR calculation shall be provided. [23]
SFL-NHD-COM-R001	NEMO-HD shall have separate communication frequency bands for Telemetry and Commanding and for Data Download. **Command uplink shall** be in **UHF band**. **Telemetry downlink shall** be in **S-band** and **data downlink shall** be in **X-band**. [24]
SFL-NHD-COM-R102	The **command uplink data rate shall** be **4 kbps**. [24]
SFL-NHD-COM-R301	**Downlink payload data rate shall** be no less than **50 Mbps**. [24]

SFL-NHD-PAY-R001-R48, derived from a SPACE-SI requirement, states that all images in one standard scene (described in **Section 3.4.3**) shall be downlinked and processed within 10 minutes after acquisition. This requirement is imperative in setting the target effective data rate (explained **Section 3.5**) for the main goal of the NEMO-HD portion of the thesis. *SFL-NHD-OPT-R01-1.2.11* specifies the signal-to-noise ratio (SNR) required for each imaging channel. This requirement is the main driver in the calculated stack depth required for each observation (discussed in **Section 3.4.3**). The last three requirements state that the satellite is required to have separate communication frequency bands for the uplink and downlink, specifically:

- UHF for the command uplink at a rate of 4 kbps, which is constrained by the SFL hardware capabilities on this particular mission, and
- X-band for the payload data downlink at a rate of no less than 50 Mbps, which is set by a customer requirement.

[1] **Level 0 images**, as per the NASA 2006 Earth Science Reference Handbook [35], are defined as reconstructed, unprocessed, full-resolution payload data.

3.3 Payload Design

To achieve its high-performance imaging and video objectives, NEMO-HD's payload design comprises two separate telescopes (two different sets of optics), referred to as the primary payload and the secondary payload. These two telescopes are connected to a low-resolution detector and a cluster of high-resolution detectors in the range of 400 – 960 nm [25].

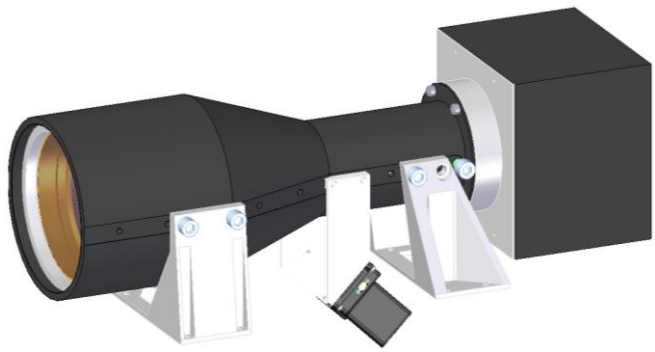

Figure 8: NEMO-HD Primary Payload

The NEMO-HD payload consists of 7 imaging and video channels in total. The primary payload, shown in **Figure 8**, is for narrow-field and high-resolution imaging and video, comprising 1 high-resolution spectral – panchromatic (HRS-PAN) camera, 4 high-resolution spectral – multispectral (HRS-MS) cameras, and 1 high-resolution – high-definition (HR-HD) video camera. The secondary payload is for wide-field and low-resolution imaging and video, comprising 1 low-resolution – high-definition (LR-HD) video camera. In terms of the payload, this thesis focuses on the imaging channels, specifically the HRS-PAN and 4 HRS-MS channels in the primary payload.

3.3.1 Primary Payload

The primary payload contains a beam splitter which splits the output from the telescope into the various channels [26]. The bandwidths for the different channels on the primary payload are:

- **HR-HD:** 400 – 900 nm (Wideband),
- **HRS-PAN:** 400 – 900 nm (Wideband),
- **HRS-MS1:** 420 – 520 nm (Blue),
- **HRS-MS2:** 535 – 607 nm (Green),
- **HRS-MS3:** 634 – 686 nm (Red), and
- **HRS-MS4:** 750 – 960 nm (Near Infrared (NIR)),

and each HRS channel has a corresponding POBC: POBC-PAN, POBC-BLUE, POBC-GREEN, POBC-RED, and POBC-NIR, where data taken using the cameras can be stored.

The idea behind various imaging channels on the primary payload is to employ a pan-sharpening scheme. Pan-sharpening – shorthand for panchromatic sharpening – is a method that uses a panchromatic image to increase the spatial resolution of a multispectral image [27]. A multispectral image contains higher spectral resolution than a panchromatic image, while a panchromatic image contains higher spatial resolution than a multispectral image. A pan-sharpened image represents a sensor fusion between the two types, resulting in the best of both worlds: an image with high spectral solution and high spatial resolution.

3.4 Observation Concept

This section provides a high-level look at the observation concept pertinent to the author's work on this mission. This includes relevant terms and definitions, a description of the spacecraft's downlink mode, as well as details on the two observation scenarios of interest on the NEMO-HD mission.

3.4.1 Terms and Definitions

In this thesis, SI prefixes are used to represent data rates in powers of 10 and IEC 60027-2 prefixes are used to represent data quantities in powers of 2. For instance, 1 Mbps is equivalent to 1×10^6 bits per second, while 1 MiB is equivalent to 1×2^{20} bytes.

The relevant terms used throughout the following sections to describe the observation concept for NEMO-HD are defined as follows:

- **Frame:** image generated from a single exposure by a single imaging sensor.
- **Scene:** single rectangular region of interest on the ground, which is independent of the imaging sensors and contains the full spectral information, spatial resolution, and SNR.
- **Swath:** larger region of interest composed of one or more contiguous scenes.
- **Stack:** collection of frames added to generate a scene and/or a swath.
- **Stack depth:** number of overlapping frames that must to be added in order to achieve the required SNR.

3.4.2 Real-time Downlink Mode

NEMO-HD has a total of five operating modes: Safe Hold Mode, Detumble Mode, Imaging (Earth Observation) Mode, Sun Pointing Mode, and Downlink Mode [21]. Of the five operating modes, the one of relevance to this thesis is the Downlink Mode, used for downloading observation data from the satellite [28]. In this context, observation data includes imagery, recorded video, payload logs, and/or telemetry files stored on the POBCs. The Downlink Mode itself has two different types of modes, differentiated by which ground station(s) are in view during the downlink pass:

- **Remote Downlink Mode:** case where only the high-speed downlink ground station[2] is in view of the spacecraft during the downlink pass (i.e. the satellite can only downlink data). In this scenario, the satellite needs to be pre-programmed, during a pass in which the full telecommand ground station[3] is in view, with a list of desired files and/or file segments along with a time-tagged script to ensure the spacecraft is commanded to begin transmission at the appropriate time.
- **Real-time Downlink Mode:** case where both the telecommand ground station and high-speed downlink ground station are in view of the spacecraft during the downlink pass. In this scenario, the ground station can request data, receive the requested data, query contents of the on-board storage, and/or re-request missing segments within the same pass.

As elaborated in **3.5**, the mode of interest for this thesis is the Real-time Downlink Mode.

3.4.3 Observation Scenarios of Interest

The two observation scenarios of interest on NEMO-HD are:

1) **single 10 km scene:** primary payload operating in snapshot mode, and
2) **single 100 km swath:** nominal imaging mode passing over Slovenia,

with full stacking [29]. The number of frames and the corresponding amount of data required to be downlinked for each scenario can be found in **Table 2**.

---

[2] **High-speed Downlink Ground Station:** X-band downlink

[3] **Telecommand Downlink Ground Station:** UHF uplink and S-band downlink

Table 2: Data Sizes of Observation Scenarios of Interest

SWATH LENGTH	1 SCENE (10KM)				10 SCENE SWATH (100KM)			
STACK DEPTH	HRS-PAN: 1 HRS-MS: 1		HRS-PAN: 5 HRS-MS: 12		HRS-PAN: 1 HRS-MS: 1		HRS-PAN: 5 HRS-MS: 12	
	FRAMES	DATA (MIB)	FRAMES	DATA (MIB)	FRAMES	DATA (MIB)	FRAMES	DATA (MIB)
HRS-PAN	2	28.81	13	187.24	16	230.45	87	1253.09
HRS-MS	1	7.18	23	165.11	8	57.43	109	782.50
TOTAL	**6**	**57.53**	**105**	**847.68**	**48**	**460.17**	**523**	**4383.09**

The two different stack depths presented in **Table 2** represent single stacking and full stacking. Due to constraints on the spacecraft volume and mass, the size of the optical aperture of the payload is limited. This limitation affects the amount of light that can reach the detectors, and for NEMO-HD, a single frame is not sufficient to meet the SNR requirement specified in *SFL-NHD-OPT-R01-1.2.11* in **Table 1**. In order to comply with the specified requirement, NEMO-HD relies on the method of stacking successive overlapping images, which boosts the amount of collected light, thus improving overall SNR. Radiometric analysis has shown that the required SNR of 75, as per the requirement, can be achieved with a stack depth of 5 for the HRS-PAN channel and a stack depth of 12 for the HRS-MS channels [29].

The standard scene in requirement *SFL-NHD-PAY-R001-R48* refers to the single 10 km scene with full stacking, which refers to the second column in **Table 2**, totaling up to just under 850 MiB in size.

3.5 Challenge: Effective Data Rate

Typically, having a downlink rate that is significantly faster than the uplink rate does not cause substantial limitations. This is because the data being requested for downlinking are usually much larger in size than the commands requesting that data. For instance, in the case of a satellite-ground link where the data being downlinked is a file captured by the payload, the data being uplinked to request for that file would be a single command. The uplink, in this case, would be a few bytes in comparison to the thousands of bytes that the file to be downlinked could be. For an exceptionally data-intensive mission, like NEMO-HD, the significant difference in the downlink and uplink rates (50 Mbps vs. 4 kbps) results in an unforeseen challenge, specifically during the Real-time Downlink Mode defined in **Section 3.4.2**.

The issue that was identified on NEMO-HD is the overall rate of the request and downlink process for the payload data is heavily affected by the uplink rate, which means the high-speed downlink rate cannot be utilized to its full potential. This problem is due to a combination of:

- expected lossy communication between the satellite and its ground station(s),
- significant difference in the downlink and uplink rates (50 Mbps vs. 4 kbps, respectively),
- large amount of payload data required for download, and
- each hole[4] request occupying 4 bytes of an uplink packet.

During a pass, NEMO-HD can expect a worst-case BER of 10^{-5}, which translates to a downlink packet error rate (PER) of 8%, as per the optimal downlink packet size analysis [30]. This means for each requested file that is downlinked, up to 8% of the total packets can be incorrect, and thus discarded and labelled as missing. In Real-time Downlink Mode, these missing packets, termed as "holes", are to be requested via the UHF uplink. The problem with requesting holes under these conditions is that the uplink packets for requesting the missing segments cannot be uplinked to the satellite fast enough to utilize the full 50 Mbps downlink rate. In other words, the downlink process in Real-time Downlink Mode is limited by the uplink rate, as the satellite can only downlink requested packets as fast, or in this case as slow, as the commands can be uplinked. Therefore, the value that is important on NEMO-HD is the effective data rate, which is defined as follows:

$$r_{eff} = \frac{n_{Mb}}{t_{downlink}} \tag{2}$$

where r_{eff} is the effective data rate in Mbps, n_{Mb} is the total size, in megabits (Mb), of files requested/queued for downlinking, and $t_{downlink}$ is the total time, in seconds, taken to downlink all the files fully and correctly.

Explained in **Section 3.2** and **Section 3.4.3**, *SFL-NHD-PAY-R001-R48* states that a single 10 km scene with full stacking shall be downlinked and processed within 10 minutes after image acquisition. In setting a target effective data rate for this thesis, it was decided that half of that time, 5 minutes, be allocated for the downlinking portion, leaving ample time, the remaining 5 minutes, for "processing" purposes.

---

[4] **Hole:** segment (one or more consecutive packets) of downlinked file that was dropped, and is, therefore, missing due to lossy communication between satellite and ground station

As detailed in **Table 2**, the total amount of data to downlink for a standard scene, as per the requirement, is 847.68 MiB. Using the conversion from MiB to bits, where $1\,\text{MiB} = 1024 \times 1024 \times 8\,\text{bits}$, the total amount of data to downlink in Mb is:

$$847.68 \times 1024 \times 1024 \times 8 = 7{,}110{,}855{,}229.44\,\text{bits} = \mathbf{7{,}110.86\ Mb}$$

Using $n_{Mb} = 7110.86\,\text{Mb}$ and $t_{downlink} = 5 \times 60 = 300\,\text{seconds}$ with (2), the target effective data rate is calculated as follows:

$$r_{eff} = \frac{7{,}110.86\,\text{Mb}}{300\,\text{s}} = \mathbf{23.70\ Mbps}$$

It is also important to note that the uplink channel is not purely reserved for payload data downlink purposes. Even during Real-time Downlink Mode, other commands unrelated to payload data requests may be uplinked. Therefore, the full uplink rate of 4 kbps was not used to justify the compliance of achieving the specified target effective data rate. Based on data and experience from previous SFL missions, it was decided that a more realistic uplink rate to assume for payload data downlinking activities was 3.5 kbps.

Therefore, the main goal of the NEMO-HD portion of this thesis was to configure the existing software programs to achieve consistent effective data rates of at least 24 Mbps for the two scenarios of interest in the worst-case scenario (i.e. highest expected BER with full stacking) using an uplink rate of 3.5 kbps.

Chapter 4

NEMO-HD: Achieving the Target Effective Data Rate

This chapter presents the author's contributions and corresponding results in achieving target effective data rates of greater than 24 Mbps for the two scenarios of interest on the NEMO-HD mission. Specifically, this chapter describes the test setup and the various software programs involved, explains the baseline algorithm used in the payload data downlink process, identifying its limitations, defines the author's goal and resulting objectives for the NEMO-HD portion of this thesis, details the author's work and contributions to the mission, and presents their results.

4.1 Test Setup

All testing was done using the NEMO-HD flatsat, which is a functional and electrical representation of the satellite itself. All results presented in this thesis were obtained by simulating radio communications over CAN as opposed to directly using radio communications, mainly due to the fact that the hardware required for radio communications was being used by the actual satellite for other testing purposes during this period. Before obtaining the results, multiple tests were executed and required modifications (explained in **Section 4.5**) were made to ensure that the simulation using CAN was accurately representative of communicating over radio. **Figure 9**, provides a high-level look at the test setup that was used, and **Table 3** provides descriptions of various software programs and units involved in this setup.

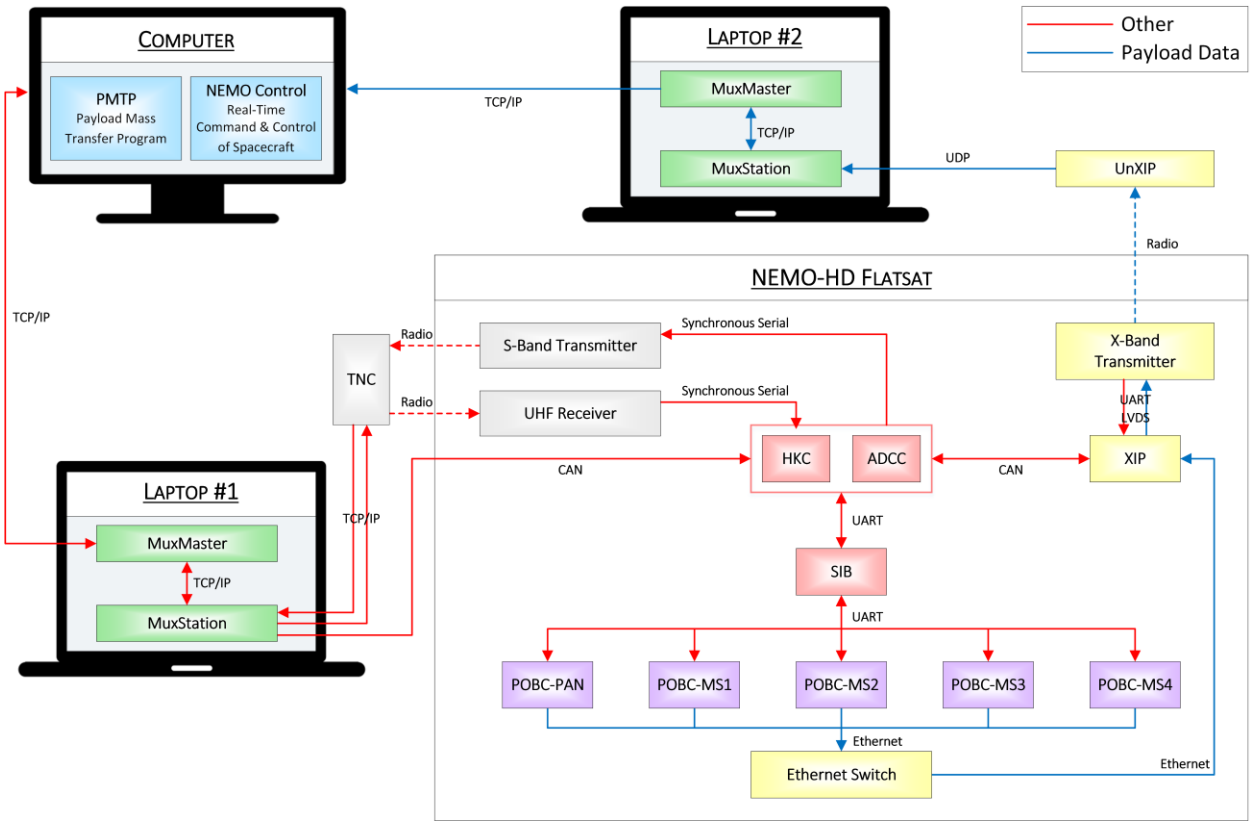

Figure 9: NEMO-HD Test Setup

Table 3: Description of Software Programs and Units in NEMO-HD Test Setup

SOFTWARE PROGRAM / UNIT	DESCRIPTION
NEMO Control	Real-time command and control interface of satellite/flatsat, including all housekeeping, power system, payload, and attitude control functions.
Payload Mass Transfer Program (PMTP)	Real-time interface of all 5 POBCs on satellite/flatsat, responsible for the request and downlink process of payload data stored on the POBCs.
MuxMaster and MuxStation pair	Responsible for forwarding and routing packets between the terminal programs on the computer and the satellite/flatsat.
Terminal Node Controller (TNC)	Radio network device used to send and receive packets over radio.
Ethernet Switch	Responsible for multiplexing high-speed data links from the 7 imaging and video channels to the XIP.
X-band Interface with Payload (XIP)	Responsible for mapping ethernet data from POBCs to the X-band transmitter.
UnXIP	Responsible for mapping X-band receiver data at the ground station back to Ethernet to send to ground station computer.

Regarding the test setup, the author's computer was used for running the two terminal programs, NEMO Control and PMTP, responsible for interfacing with the flatsat and various POBCs, respectively. The two laptops connected to the flatsat, referred to as Laptop #1 and Laptop #2, both ran instances of a MuxMaster and MuxStation pair. The pair running on Laptop #1 was responsible for routing packets between the terminal programs and the flatsat, which included packets from PMTP and NEMO Control as well as all packets from the flatsat unrelated to payload data. The pair running on Laptop #2 was responsible for routing all packets related to payload data from the POBCs on the flatsat to the terminal programs on the computer. It is also important to note that the MuxStation program allows simulated BERs for both the uplink and downlink, and the MuxMaster program allows simulated uplink rates, as shown in **Figure 10**.

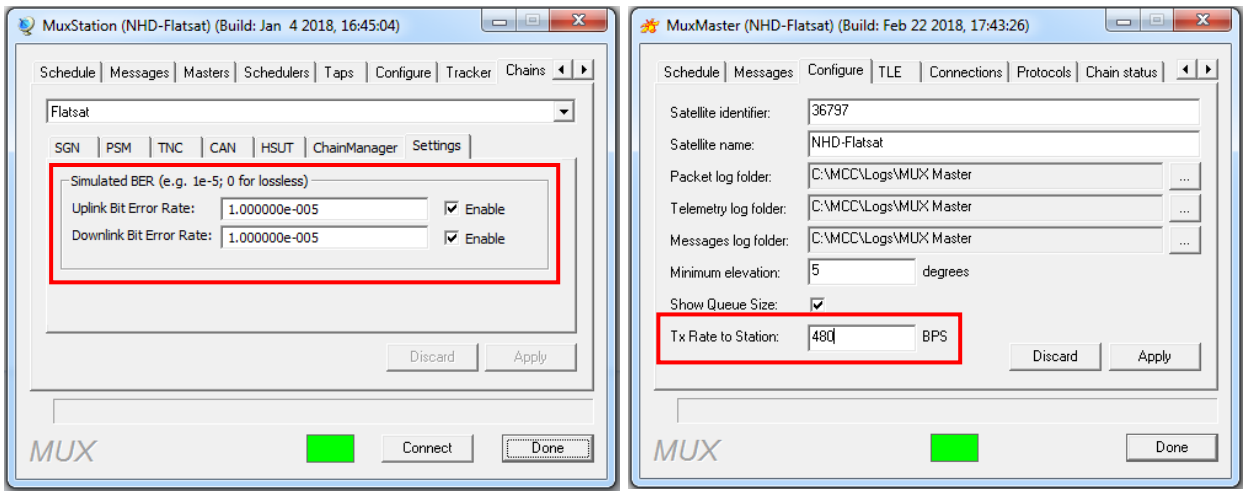

Figure 10: MuxStation and MuxMaster Programs – Simulated BERs and Uplink Rate

Since all 7 payload channels use Ethernet for interconnect, an Ethernet switch was selected to connect all the channels and multiplex the high-speed data links to the XIP. The XIP encodes and translates all Ethernet packets into synchronous serial data before forwarding them to the X-band transmitter, which sends them over radio to the receiver on ground. The UnXIP accepts the low-voltage differential signaling data from the receiver, descrambles the data, decapsulates the HDLC framing, and converts the received packets to Ethernet before forwarding them to Laptop #2. The Mux pair on Laptop #2 forwards the downlinked payload data packets to the PMTP on the computer. After receiving the requested data, the PMTP software locates any missing segments and automatically uplinks request packets to Laptop #1 to be forwarded to the POBCs on the flatsat. This process is repeated until the requested files are fully downloaded.

4.2 Baseline Algorithm

The baseline algorithm refers to the initial hole-request algorithm that had been implemented on the PMTP software prior to the author's allocation on the project. In short, when a requested file is received, the baseline algorithm locates all the missing packets (holes) of that file and constructs the appropriate uplink packets to request for those holes.

The picture shown in **Figure 11** provides a visual representation of the packets as they are downloaded. Specifically, this picture shows how many packets were received correctly in the first download attempt of a 7.5 MB file at 10^{-5} BER, where the green blocks represent the successfully received packets and the white blocks represent the incorrectly received, and therefore, missing packets. For every requested file received, the baseline algorithm determines which of the packets are missing, and therefore, need to be re-requested.

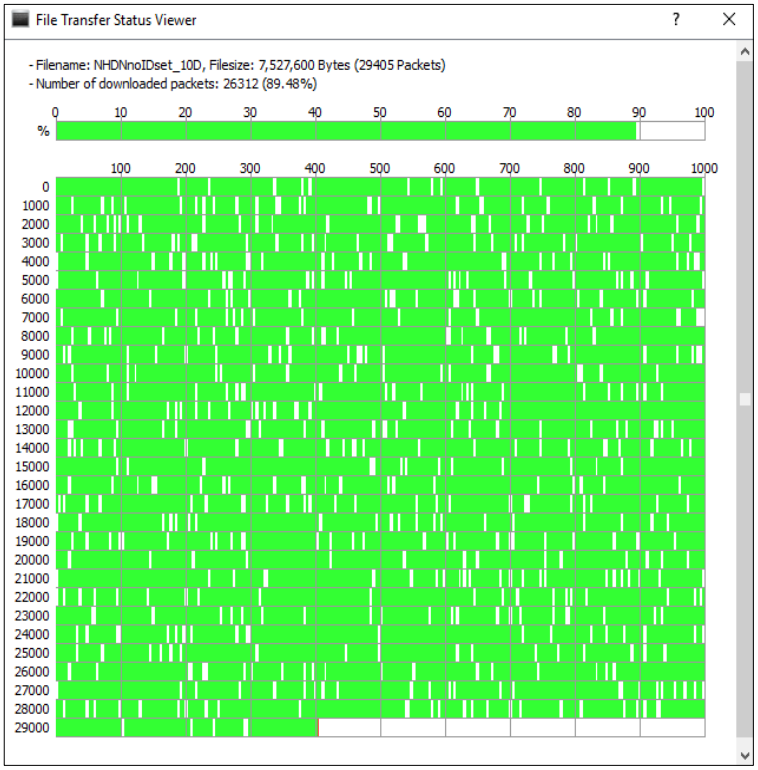

Figure 11: Visual Representation of Packets Downloaded (PMTP)
(First Download Attempt of 7.5 MB file at 10^{-5} BER)

This algorithm creates requests for these holes (one or more consecutive missing packets) by specifying their offset and length. A hole's offset refers to the location of the first packet in that hole, while a hole's length refers to the number of missing packets that constitute that hole. The hole requests are 4 bytes in size, where 2 bytes are used for specifying the offset and the other 2 bytes are used for specifying the length.

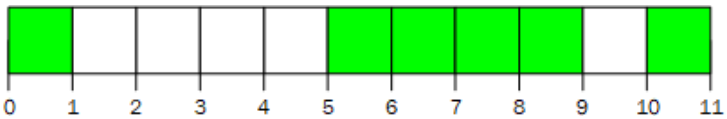

Figure 12: Zoomed-in Look at Missing Packets

Looking at a hypothetical span of 11 packets shown in **Figure 12**, the baseline algorithm will specify two holes. The first hole request will be identified as: offset = `0x0001` with length = `0x0004`, while the second hole request will be identified as: offset = `0x0009` with length = `0x0001`, which together would occupy 8 bytes of an uplink packet. Additionally, this algorithm is designed to group certain holes together based on the number of correct packets they are apart, specifically, less than 5 correct packets. Therefore, once all the holes are located, the algorithm groups any nearby holes together. In this case, **Figure 12** results in a single hole request of offset = `0x0001` and length = `0x0009`, occupying 4 bytes of an uplink packet instead of the 8 prior to the algorithm grouping the holes together. Although this equates to re-requesting packets that have already been correctly received, it is understood and explained in the later sections that re-downlinking correctly received packets – to a certain extent – is much more favourable than expending bytes of an uplink packet for additional requests.

Although this algorithm has been and is being successfully used on previous/current SFL missions, the results outlined in **Table 5** in **Section 4.6** show that it is not sufficient in obtaining the target effective data rate of 24 Mbps under the conditions specific to the NEMO-HD mission. This can be attributed to the following major limitations that were identified in the PMTP software:

- unusually long data processing delays experienced on PMTP, and
- not accounting for varying sizes of files.

The data processing delay represents the time it takes the thread responsible for handling the hole requests to process a received packet. The acceptable and expected range of this delay is on the order of milliseconds; however, PMTP was seeing delays up to a minute during the downlink process. The second limitation alludes to the fact that the payload data files to be downlinked can vary significantly in size. For instance, the files making up the single 10 km scene range from 500 bytes, 7.5 MB, or 15 MB in size. It is important to note that the single 10 km scene contains the least amount of size variety; therefore, if the algorithm struggles in the least variable case, it will likely perform worse in other cases. Specifically, video files captured using the HR-HD or LR-HD channels can be as large as 50 MB. Naturally, for a given BER, there will be more holes to request for files of larger size, meaning more uplink packets will be required. As a result, a problem that was observed with the baseline algorithm on the PMTP software is it disproportionately allocates more hole requests for the larger files compared to the smaller files, which saturates the uplink channel very quickly, significantly slowing down the overall process.

4.3 NEMO-HD Thesis Goal and Objectives

As stated in **Section 3.5**, the main goal for the NEMO-HD portion of this thesis was to configure the existing software programs to achieve effective data rates of at least 24 Mbps for the two scenarios of interest in the worst-case scenario (i.e. highest expected BER with full stacking) using an uplink rate of 3.5 kbps. Taking into consideration the limitations with the PMTP software identified in **Section 4.2**, this goal can be broken up into the following objectives:

- locate and fix the cause of the data processing delay seen in PMTP,
- configure the PMTP software and/or POBC software as necessary,
- find the optimal parameters required to achieve the target effective data rate under the specified conditions, and
- perform tests to prove requirement compliance.

4.4 PMTP Software Improvements

The author's work on the existing PMTP software aimed to resolve the two limitations identified in **Section 4.2**. Specifically, this section presents the author's modifications to the software to significantly reduce the data processing delay as well as contributions to the baseline algorithm, termed the upgraded algorithm, to account for the various file sizes and increase the overall effective data rate.

4.4.1 Data Processing Delay

The data processing delay represents the time it takes the thread responsible for handling the file hole requests to process a received packet, with the acceptable range being on the order of 0 – 2 ms. PMTP, however, was seeing delays up to a minute during the downlink process. Even if any changes were made to the algorithm itself, the improvements would not be useful or even apparent with such large delays in place. Therefore, before any modifications to the algorithm could be implemented, this data processing delay issue had to be resolved.

As the source of the data processing delay was completely unknown, the overall debugging process included investigating all the different threads and functions, in the PMTP software, related to the

data downlink process. Originally surmised to have been due to a bigger component, the principal cause of the long data processing delay was actually discovered to be a poor choice in initial construction of a list.

A global list, **`m_hs_download_file_list`**, is a list which contains all the names and information of the files requested from PMTP for high-speed downloading (i.e. download via X-band). Every time a packet related to high-speed downloading is received, this list is searched to obtain relevant information regarding that file(s). The search through this list was the major component in the long data processing delay, attributed to its initial construction. When files were requested for high-speed downloading via PMTP, they were being appended to the list. Due to the order of the downlink process, the file being searched always ended up being located at the end of this list. This meant as more files were requested for high-speed downlinking, the longer the data processing delay became.

Therefore, the solution was to modify the initial list construction, specifically, having the files prepended to the list as opposed to appended. Once this change was implemented, the data processing delays were seen to be in the range of 0 – 2 ms as expected.

4.4.2 Brute Force Algorithm

One method that was considered when the author first approached this challenge was the idea of using a brute force algorithm. Specifically, requesting and downlinking entire files multiple times until all segments of the files are all fully received, as opposed to locating and requesting each of the missing segments. The reasoning behind this was that since the brute force algorithm will require significantly less uplink packets, the issue of saturating the uplink channel, as observed when using the baseline algorithm, is nonexistent. This method had never been deemed feasible as no previous SFL mission has had such a high downlink rate as NEMO-HD. However, we see from the results, presented in **Table 5** in **Section 4.6**, that although the downlink rate is significantly faster, it is not sufficient to justify downloading all the required files multiple times. Since the files for downloading on this mission are not only large in quantity, but in size as well, solely eliminating the saturation on the uplink side does not necessarily equal to an increase in the overall effective data rate. It can be observed that saturation on the downlink channel is possible as well, which also impacts the effective data rate, and is, therefore, something that needs to be considered.

4.4.3 Additional Parameters and Upgraded Algorithm

As mentioned in **Section 4.2**, the baseline algorithm alone is not capable of meeting the target effective data rate under the specified conditions. The modifications to the existing software programs in order to meet the target effective data rate include: an addition of two parameters in the POBC software, and an additional parameter and upgraded algorithm in the PMTP software.

4.4.3.1 POBC Parameters: Retransmit Threshold and Retransmit Number

During the time the author was configuring the PMTP software, two parameters were added in parallel to the POBC software by Nicolaas Handojo at SFL. The parameters, **`multi_retransmit_thresh`** and **`multi_retransmit_num`**, are implemented in the POBC thread responsible for handling the downlinking of payload data. The values of these parameters are user-configurable via NEMO Control, and together they specify the number of times (**`multi_retransmit_num`**) that the POBCs shall downlink hole requests smaller than the assigned threshold size (**`multi_retransmit_thresh`**). For instance, parameters **`multi_retransmit_num`** and **`multi_retransmit_thresh`** with values set to 2 and 15, respectively, will result in the POBCs downlinking all hole requests, with a size less than or equal to 15 packets, twice at the time of that request. These parameters help to lower the probability of having to re-request small holes. This is preferable because reserving uplink packet space for bigger hole requests means less saturation in the uplink channel and, therefore, higher overall effective data rates.

4.4.3.2 PMTP Parameter: Maximum Number of Hole Requests per MB of File

An additional parameter, called **`max_num_holes_per_MB`**, has been implemented in the PMTP software as part of the upgraded algorithm explained in **Section 4.4.3.3**. Combined with an existing **`max_holes_per_request`** parameter, this additional parameter allows users to set a limit on the number of uplink packets as per the following equation,

$$n_{up} = \text{ceil}\left(\frac{n_{h,MB} \times s_{file}}{n_{h,req}}\right) \tag{3}$$

where n_{up} is the maximum number of uplink packets allowed to be used for a file's hole requests per turn, $n_{h,MB}$ is the **`max_num_holes_per_MB`**, s_{file} is the size of the file in MB, and $n_{h,req}$ is the **`max_holes_per_request`**.

For instance, parameter **`max_num_holes_per_MB`** with value set to 13 for a 7.5 MB file would limit the maximum number of hole requests this file can send per turn to $13 \times 7.5 \approx 98$. Then if parameter **`max_holes_per_request`** is set to 50, this would limit the uplink to a maximum of two packets to be used by that particular file, where one packet contains the first 50 hole requests and the other the remaining 48.

4.4.3.3 PMTP Upgraded Algorithm

The upgraded algorithm uses information gathered by the baseline algorithm in conjunction with the additional parameter described in **Section 4.4.3.2**. A high-level flow of the overall request process for each file, with the upgraded algorithm, is laid out in **Figure 13**.

In the PMTP request and downlink process, the first step is to send the requests, whether that be requests for an entire file or for specific holes of a received file. Once PMTP receives the data that it is expecting, the baseline algorithm locates any missing segments of the file of relevance.

If it determines that there are no more missing segments, that particular file has been successfully received and no additional requests need to be sent.

On the other hand, if there are missing segments detected, the baseline algorithm generates a list of hole offsets and lengths as explained in **Section 4.2**. After that, the upgraded algorithm determines the maximum number of hole requests allowed for that file using (3). Once that is determined, the upgraded algorithm uses the list of hole requests for the file generated by the baseline algorithm to determine if the number of hole requests in that list is greater than the maximum number of hole requests allowed for that file.

If it is not greater, PMTP uses the original list created by the baseline algorithm to uplink the hole requests. Conversely, if it is greater, the upgraded algorithm regroups the holes into bigger segments such that the number of holes is within the maximum allowed. It does this by grouping the holes that have the least number of correct packets between them, to ensure that the least possible amount of correctly received packets is requested in this process. PMTP then uses the list condensed by the upgraded algorithm to uplink the hole requests.

This process is repeated until the specified file is fully downloaded.

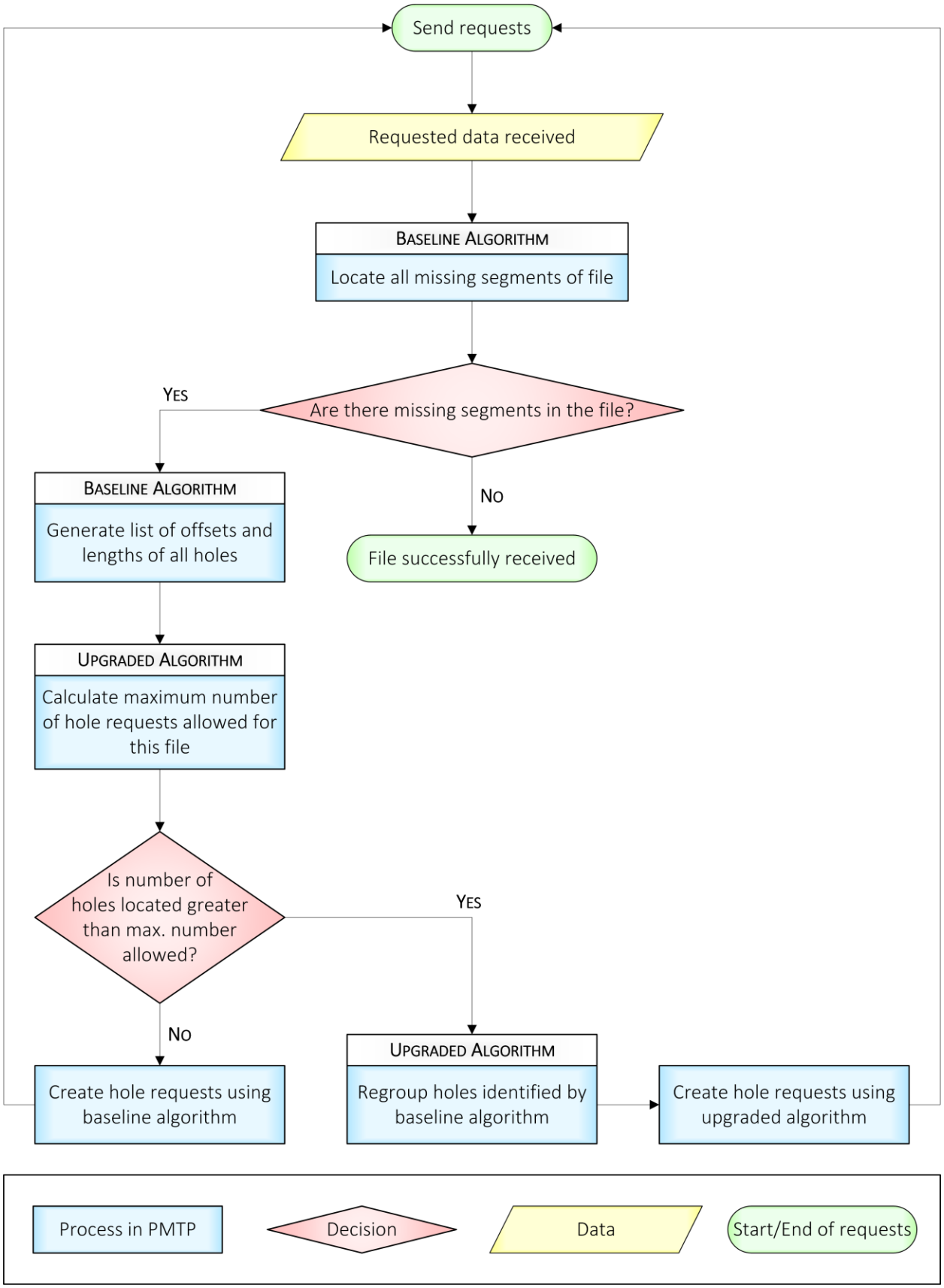

Figure 13: PMTP Request and Downlink Process Flow

4.5 Unexpected Hindrances

To ensure that simulating radio communications via CAN was truly representative of actual radio communications between the satellite and ground, tests were simulated using radio to confirm that there were no significant differences in the performance. During the initial tests using radio, the effective data rates observed were significantly lower at less than half of what were observed when testing over CAN. From analyzing the data, it was observed that the uplink packets were not reaching the flatsat at the expected uplink rate of 4 kbps. This was believed to be due to either or a combination of – later discovered to be both –the following two speculations:

- the flow control algorithm on the MUXMaster program, which simulates the uplink rate by controlling the dispatch rate of uplink packets, was not executing as expected, and/or
- there was a problem with the software on the TNC.

4.5.1 MuxMaster Flow Control

The MuxMaster program, responsible for directing the uplink packets to the flatsat, contains a flow control algorithm which regulates the dispatch rate of uplink packets based on the desired uplink rate. The program allows operators to set the desired rate in bytes per second (Bps), as shown in **Figure 10**, where 8 bits is equal to 1 byte. Therefore, to simulate the 4 kbps uplink performance, the parameter was originally set to 500 Bps.

In order to pinpoint any issues with the flow control algorithm, timed tests were conducted where full-sized packets were continuously transmitted over radio. Using a custom script, random full-sized NSP packets were generated and sent to MuxMaster to dispatch to MuxStation at the set uplink rate. MuxStation then routed these to the TNC, which applied the HDLC framing and transmitted them over radio. As specified in **Section 2.1**, a full NSP packet for the NEMO-HD mission is 265 bytes (overhead included) and packets are HDLC framed when transmitting over radio. Specifically, HDLC framing adds 6 bytes of overhead to a NSP packet as well as another estimated 1.61% for bit-stuffing, which brings a full NSP packet to approximately 275 bytes [31]. This means for a 4 kbps, or equivalently 500 Bps, uplink rate, MuxMaster is designed to dispatch a full packet every 550 ms. Through the timed tests, it was observed that the estimated size of a HDLC-framed packet calculated in the flow control algorithm could sometimes be lower than the

actual size it left the TNC at, meaning it could be dispatched from the MuxMaster faster than it would be dispatched from the TNC. Since the bit-stuffing happens on the TNC, the exact size of a packet being transmitted from the TNC is unknown on the MuxMaster end. In addition, unlike the MuxMaster, the TNC is not capable of queuing up packets. In other words, any packet that is sent to the TNC before the current one has been transmitted is dropped. Although the flow control algorithm used on MuxMaster could have been implemented on the TNC, due to time constraints, the approach was to use a slightly lower rate (i.e. 480 Bps) that would closely represent a 4 kbps uplink rate, but be slow enough such that no packets would be dropped by the TNC.

4.5.2 Terminal Node Controller Software

To determine if there was an issue on the TNC side, timed tests were performed for the TNC as well. Similar to the tests described in **Section 4.5.1** for observing potential issues with the MuxMaster flow control, series of full-sized packets were transmitted via the TNC to locate any unexpected delays in the process. Through these tests, gaps of 160 ms were observed every 1.25 seconds, which were deemed to be considerable enough in impacting the flow of uplink packets. These gaps were mainly due to a default process in the TNC application software, which updates the TNC's LCD display every 1.25 seconds. The function responsible for updating the LCD display has always been present in the TNC application software, specifically ones that have been used on previous missions; however, since no mission prior to NEMO-HD required full use of the uplink rate, this delay never caused any issues. The TNC application software for NEMO-HD has, therefore, been updated to eliminate this LCD display update, which eradicated the unnecessary delays. It was then confirmed that after these updates, the TNC had no issues transmitting packets at the specified 4 kbps uplink rate.

4.6 Modified PMTP Software Results

Once it was verified that simulating over CAN is representative of the radio communications and the PMTP software was confirmed to run smoothly, performance tests were conducted using the setup explained in **Section 4.1**. This section discusses the parameter values the author deemed sufficient and reasonable for achieving the target effective data rate under the specified conditions. It also presents the obtained resulting effective data rates for the various scenarios on this mission.

4.6.1 Recommended Parameter Values

This section describes the different parameters involved in the PMTP request and downlink process. **Table 4** lists those parameters, their descriptions, the values used to obtain the effective data rates presented in **Section 4.6.2**, and the rationale behind it.

Table 4: Relevant Parameters in PMTP Request and Downlink Process

PARAMETER NAME	DESCRIPTION	SET VALUE	RATIONALE
Max. Bytes per Request	Maximum size of file in bytes to request via PMTP	65,011,712 bytes	62 MiB is currently the maximum size allowed on PMTP: $62\ \text{MiB} = 62 \times 1024 \times 1024 = 65{,}011{,}712\ \text{bytes}$
Number of Bytes per Block	Number of bytes of file per downlink packet	1000 bytes	Per previous analysis, it was determined that 1000 bytes is the optimal value [31]
Max. Holes per Request	Maximum number of hole requests allowed per uplink packet.	50	As mentioned in **Section 2.1.1**, the data field of an uplink packet is 260 bytes. Included in that data field is the filename which can vary from 12 – 30 bytes. Each hole request is 4 bytes; therefore, a safe allocation of 200 bytes for hole requests was selected.
Max. Gap Size for Hole Bridging	Used in baseline algorithm. Maximum number of correct packets between two holes to combine into a single hole request.	5000 bytes	Per previous analysis, it was determined that 5 – 10 packets was the optimal setting. Since each downlink packet is set to1000 bytes, this equates to 5000 – 10000 bytes. Since the upgraded algorithm will group the holes as well, preference was to use the lower value of 5000 bytes.
Max. Holes per MB of File	Maximum number of hole requests allowed per uplink turn per file.	13	Most of the files for POBC-MS consist of 7.5 MB files while for POBC-PAN consists of 15 MB files. Combined with the max. holes per request parameter, this parameter limits the number of uplink packets for uplink turn to 2 and 3 for POBC-MS and POBC-PAN, respectively.
Retransmit Threshold	These two parameters, as described in **Section 4.4.3.1**, define the number of times POBCs should downlink a hole that is smaller than a specified length.	15 (all POBCs or MS only) 5 (PAN only)	The values were determined via series of tests. It was determined that downlinking hole requests two times is enough to ensure that all small holes will be correctly received over those two occurrences. Anything greater than the values listed resulted in too much redundant behaviour, which ended up saturating the downlink channel more quickly. Anything less than the values listed did not contribute much in helping to increase the effective data rate.
Number of Retransmits		2	

4.6.2 Effective Data Rates Obtained

This section presents the effective data rates obtained through testing. Specifically, it summarizes and compares the effective data rates obtained for the single 10 km scene with full stacking, downloading from one POBC-MS, one POBC-PAN, and all 5 POBCs with uplink BER of 0, 10^{-6}, and 10^{-5} and downlink BER of 10^{-6} and 10^{-5} via:

- the upgraded algorithm at 4 kbps uplink rate,
- the upgraded algorithm at 3.5 kbps uplink rate,
- the baseline algorithm at 3.5 kbps uplink rate, and
- the brute force algorithm at 3.5 kbps uplink rate.

Table 5: Effective Data Rates for Standard Scene under Various Conditions

# OF POBCS	# OF FILES	TOTAL SIZE (MB)	EFFECTIVE DATA RATE (MBPS)					
			10^{-5} DOWN BER			10^{-6} DOWN BER		
			0 UP BER	10^{-6} UP BER	10^{-5} UP BER	0 UP BER	10^{-6} UP BER	10^{-5} UP BER
UPGRADED ALGORITHM @ 4 KBPS UPLINK								
1 (MS)	46	173	~ 27	~ 27	~ 27	~ 43	~ 43	~ 43
1 (PAN)	26	196	~27	~ 27	~ 27	~ 43	~ 43	~ 43
5 (all)	210	889	~ 27	~ 27	~ 26.5	~ 44	~ 44	~ 44
UPGRADED ALGORITHM @ 3.5 KBPS UPLINK								
1 (MS)	46	173	~ 26.5	~ 26.5	~ 26.5	~ 43	~ 43	~ 43
1 (PAN)	26	196	~ 26.5	~ 26.5	~ 26.5	~ 42.5	~ 42.5	~ 42.5
5 (all)	210	889	~ 26	~ 26	~ 25.5	~ 43.5	~ 43.5	~ 43.5
BASELINE ALGORITHM @ 3.5 KBPS UPLINK								
1 (MS)	46	173	~ 10	~ 10	~ 9	~ 41	~ 41	~ 40.5
1 (PAN)	26	196	~ 9	~ 9	~ 9	~ 40	~ 36.5	~ 36
5 (all)	210	889	~ 9	~ 9	~ 9	~ 43.5	~ 43.5	~ 42
BRUTE FORCE ALGORITHM @ 3.5 KBPS UPLINK								
1 (MS)	46	173	~ 10	~ 10	~ 10	~ 19	~ 18	~ 18
1 (PAN)	26	196	~ 9	~ 9	~ 9	~ 15	~ 15	~ 15
5 (all)	210	889	~ 10	~ 10	~ 10	~ 17	~ 17	~ 17

Using the single 10 km scene with full stacking as the comparison scenario, the effectiveness of the upgraded algorithm can be observed from the results presented in **Table 5**. The effective data

rates of particular interest are the rates listed in the 3rd column, which represent the effective data rates in the worst-case scenario (i.e. worst expected BER on both uplink and downlink). From the results, the impact that the upgraded algorithm has on the effective data rates is clearly significant, especially in the worst-case BER scenarios. Comparing the worst-case scenario rates of the upgraded algorithm to the baseline algorithm, an effective data rate increase of up to 194% is achieved. It is also evident that although the baseline algorithm alone may satisfy the requirement in the 10^{-6} downlink BER case, it is not capable of doing so in the 10^{-5} downlink BER case.

To ensure that the performance achieved for the single 10 km scene can be maintained for the second scenario of interest, tests for the single 100 km swath at 3.5 kbps uplink rate were conducted as well. The effective data rates obtained using the upgraded algorithm at 3.5 kbps for both scenarios of interest on the NEMO-HD mission can be found in **Table 6** below.

Table 6: Effective Data Rates for NEMO-HD Scenarios of Interest at 3.5 kbps Uplink Rate

# OF POBCS	# OF FILES	TOTAL SIZE (MB)	EFFECTIVE DATA RATE (MBPS)					
			10^{-5} DOWN BER			10^{-6} DOWN BER		
			0 UP BER	10^{-6} UP BER	10^{-5} UP BER	0 UP BER	10^{-6} UP BER	10^{-5} UP BER
UPGRADED ALGORITHM @ 3.5 KBPS UPLINK (10 KM SCENE)								
1 (MS)	46	173	~ 26.5	~ 26.5	~ 26.5	~ 43	~ 43	~ 43
1 (PAN)	26	196	~ 26.5	~ 26.5	~ 26.5	~ 42.5	~ 42.5	~ 42.5
5 (all)	210	889	~ 26	~ 26	~ 25.5	~ 43.5	~ 43.5	~ 43.5
UPGRADED ALGORITHM @ 3.5 KBPS UPLINK (100 KM SWATH)								
1 (MS)	218	821	~ 27	~ 27	~ 27	~ 43	~ 43	~ 43
1 (PAN)	174	1314	~ 28	~ 28	~ 28	~ 44	~ 44	~ 44
5 (all)	1046	4596	~ 26	~ 26	~ 26	~ 43.5	~ 43	~ 43

From the results presented, it can be concluded that with the modifications and additions described in **Section 4.4.3**, PMTP is capable of meeting the target effective data rate of 24 Mbps for the two scenarios of interest in the worst-case BER conditions at an uplink rate of 3.5 kbps.

Chapter 5

NorSat-3: Background

This chapter provides an overview of the Norwegian Smallsat Program, an overview of the NorSat-3 mission, list of mission requirements applicable to the author's work, details on relevant devices on the NorSat-3 satellite, and high-level descriptions of the software programs used.

5.1 Norwegian Smallsat Program

In Norway, there exists a Smallsat Program for maritime surveillance funded by the Norwegian Coastal Administration (NCA). The purpose of this program is to provide governmental users, ranging from Norwegian Maritime Directorate to armed forces, with data that are crucial for assuring protection and safety in Norwegian waters [32]. This program began with the successful collection of Automatic Identification System (AIS) signals from space with the AISSat-1 satellite, launched in 2010 [33]. Currently, five satellites (AISSat-1, -2, and -3 and NorSat-1 and -2) have been built and developed by SFL as part of this program. These satellites provide essential information on vessel position, identity, and navigation data to the various governmental agencies, allowing near real-time maritime surveillance. The main goal of this program is to ensure continuity with launches every 2 – 3 years and improvements in performance through:

- new AIS receivers with improved sensitivity and receiving capacity,
- expansion to high density regions, and
- complementary sensor technology to detect and track more vessels [34].

5.1.1 AISSat-1

Norway is a country with a long history in shipping, fisheries, and oil and gas exploitation at sea. With over 2 million square kilometres of ocean regions, maritime traffic surveillance has always posed a challenge. In 2006, The Norwegian Space Centre (NSC) and the Norwegian Defence Research Establishment (FFI) began the development of a small satellite that would collect AIS signals from ships, which later became known as AISSat-1 that successfully launched in July of 2010 [33]. The satellite was built and developed by SFL while the payload – AIS receiver – was developed by Kongsberg Seatex. Not too long after launch, its data quickly became crucial for maritime authorities; so much so, that the NCA managed to secure long-term funding, for what is now referred to as the Norwegian Smallsat program, to ensure continuity of these satellites [34].

5.1.2 AISSat-2

The second satellite developed and built for the Smallsat program was AISSat-2, launched in July of 2014 [35]. This satellite is identical to its predecessor AISSat-1, but with all the improvements made to the first satellite following its launch. Its main objective was to increase coverage, shorten revisit times, and provide enhanced quality and reliability of AIS data through an upgraded AIS receiver. AISSat-1 and -2 quickly demonstrated to Norwegian authorities how cost-efficient and sensible small satellites were.

5.1.3 AISSat-3

The third satellite in this series was AISSat-3 which was similar to AISSat-2 but incorporated an upgraded AIS receiver for improved ship detection. AISSat-3 was initiated such that it could work, in tandem, with its predecessors AISSat-1 and -2 to increase coverage, shorten revisit times, and enable redundancy for space-based AIS observation under Norway's direct control [36]. AISSat-3 was launched on November 28, 2017, but unfortunately did not reach orbit due to failure of the launch vehicle.

Motivated by the success of the AISSat series, the NorSat series, which uses platforms with increased capacity and capability, was initiated to carry more and/or bigger payloads.

5.1.4 NorSat-1

The fourth satellite built for this program was NorSat-1, which was launched in July of 2017 [37]. NorSat-1 is Norway's first multi-payload microsatellite with two scientific payloads in addition to an upgraded AIS receiver. The two scientific payloads are the Compact Lightweight Absolute Radiometer (CLARA) for measuring total solar irradiance and the multi-needle Langmuir Probe for studying ambient space plasma characteristics. The main mission objectives for NorSat-1 were to provide maritime monitoring of AIS for the NCA, investigate total solar irradiance, and investigate the effects of space weather on the upper ionosphere.

5.1.5 NorSat-2

The fifth satellite built for this program was NorSat-2, which was launched together with NorSat-1 in July of 2017 [38]. NorSat-2 includes a very high frequency (VHF) data exchange system transceiver which, coupled with its deployable Yagi antenna, makes up the VDE-SAT payload. It also includes the same advanced AIS receiver as the one on NorSat-1. The main mission objectives for NorSat-2 were to detect and track maritime traffic in Norwegian and international waters, and allow bi-directional exchanges with ground assets equipped with VHF transceivers.

5.2 NorSat-3 Mission Overview

With AISSat-1 and -2 currently operating far beyond their nominal life, the NSC in cooperation with FFI initiated the development of NorSat-3 in order to ensure the continuity and operational capacity as promised by the Smallsat program. NorSat-3 is, therefore, the sixth Norwegian satellite built by SFL, following the successful AISSat-1 and -2, and NorSat-1 and -2 satellites.

NorSat-3 is a maritime monitoring microsatellite with a planned launch in 2020. It is designed to operate in a SSO with an altitude between 550 – 650 km and no restrictions on the LTAN [39]. NorSat-3 will use the heritage platform of NorSat-1 and -2 and include the same version of the AIS receiver as one of its payloads. As this mission will add another satellite to Norway's assets in space [40], NorSat-3 represents another barrier-breaking advance in small satellites, utilizing SFL's NEMO technology. This microsatellite is designed to capture signals from frequencies allocated for civil navigational radars by the International Maritime Organization (IMO), and to

demonstrate an experimental payload that may potentially provide better maritime awareness when combined with the AIS receiver. The resulting main objectives of NorSat-3 are to:

- demonstrate an experimental Navigation Radar Detector (NRD) payload for space-based maritime surveillance of non-cooperative targets, and
- provide continuity and extension to the Norwegian Smallsat program.

To achieve these objectives, NorSat-3 is equipped with two payloads: an AIS receiver and a NRD payload. The spacecraft weighs approximately 16.9 kg with dimensions of 62 cm x 56 cm x 33 cm. The visual of the satellite system is shown in **Figure 14**, where the NRD antenna elements are located on the bottom, but not shown due to confidentiality reasons.

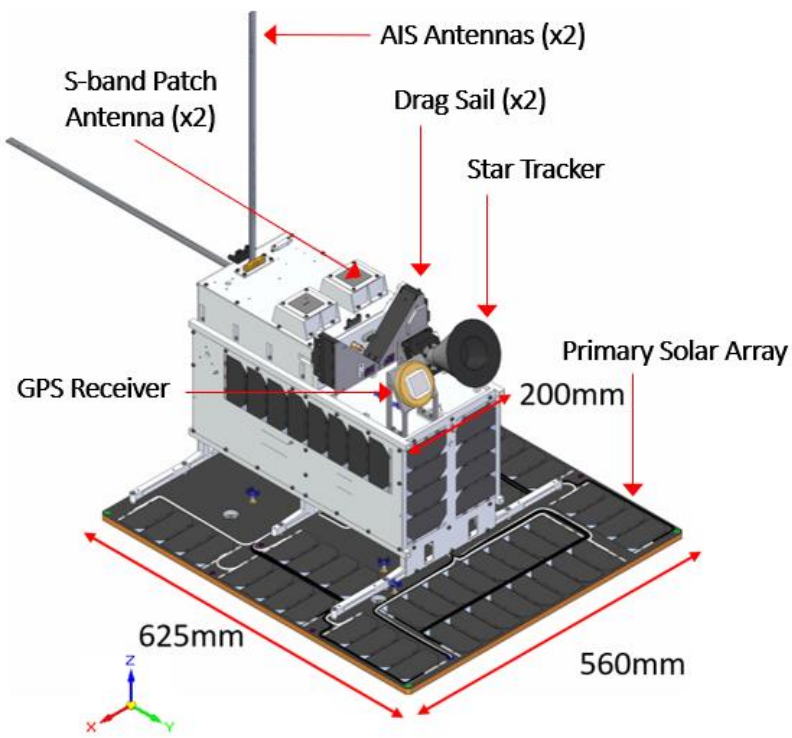

Figure 14: NorSat-3 System

5.3 Relevant Mission Requirements

The requirements that are specific to the author's work on the mission are listed in **Table 7**.

Table 7: Relevant NorSat-3 Requirements [41]

REQUIREMENT NUMBER	REQUIREMENT
NS3-ACS-R018	The **GPS shall** deliver a **pulse per second (PPS) with an accuracy of ±20 ns to** the C&DH subsystem and the **NRD payload**.
NS3-CDH-R007	**Communication between the platform and the payload shall be RS-422 or RS-485** as required by each payload.
NS3-CDH-R024	The **C&DH shall deliver a pulse per second (PPS) signal to the AIS payload**.
NS3-COM-R006	The **uplink data rate shall** support **uplink of 20 MB over 24h** using Vardϕ ground station, given **nominal operating conditions**.
NS3-SYS-R006	The spacecraft **shall** be designed to **accommodate an AIS payload as per the mutually agreed upon interface control document (ICD)**.
NS3-SYS-R007	The spacecraft **shall** be designed to **accommodate a NRD payload as per the mutually agreed upon interface control document (ICD)**.
NS3-CDH-R008	The **default baud rate shall** be **921.6 kbit/s for the NRD payload** and **230.4 kbit/s for the AIS payload.**
NS3-SYS-R009	The spacecraft **shall** be capable of **transmitting real-time payload data** whenever a **ground station is in view**.
NS3-SYS-R010	The spacecraft **shall** be capable of **transmitting stored payload data** whenever a **ground station is in view**.

The first three requirements – *NS3-ACS-R018*, *NS3-CDH-R007*, and *NS3-CDH-R024* – all cater to interface requirements between the payloads and other devices, namely the POBC and the GPS receiver. Explained in greater detail in **Section 6.3**, these requirements not only demand the need for a serial interface board, but also a novel design for this specific mission.

The fourth requirement, *NS3-COM-R006*, imposes a specific uplink data rate, which is expected to be higher than the nominal 4 kbps the typical SFL receivers are capable of. This requirement is to ensure that the expected payload software updates can be easily executed without hindering nominal operations. This requires a receiver, in addition to the nominal 4 kbps UHF receiver, capable of higher data rates as discussed in **Section 5.4** and **Section 6.4**.

The fifth and sixth requirements – *NS3-SYS-R006* and *NS3-SYS-R007* – refer to the interfaces for the NorSat-3 payloads: the AIS receiver and the NRD payload. Although these requirements are

more concerned with the physical interfaces of the payloads to the satellite, the interfaces from a software perspective are equally important as well. The payload interfaces from a software standpoint mainly refer to the NEMO Control interfaces and the payload application threads on the POBC software, which collectively ensure that all commands specified in the respective ICDs are implemented for the operator to use and all outputs, such as telemetry values and responses, from the payloads are parsed and handled correctly. As mentioned in **Section 5.2**, the version of the AIS receiver on NorSat-3 is the same as the one used on NorSat-1 and -2. Since the NEMO Control interface and application thread for the AIS receiver have mostly been developed for the NorSat-1 and -2 missions, this thesis will mainly focus on the NEMO Control interface and application thread developed for the new NRD payload. The major contributions to the interface and thread for the AIS receiver revolve around image control, specifically the uploading and downloading of payload application images, which are discussed in greater detail in **Section 6.6.1**,

The last three requirements – *NS3-CDH*-R008, *NS3-SYS-R009*, and *NS3-SYS-R010* – impose a lower bound on the data transmission rate from the payload to the POBC and vice versa, as well as how the POBC needs to handle the data received from either payload. As such, each payload requires a dedicated interface for data transmission, NEMO Control interface, and application thread that is, at the very least, capable of forwarding payload data to ground during a pass as well as storing payload data in a file on the POBC file system and transmitting those files at later times.

5.4 High-speed S-band Receiver

As per *NS3-COM-R006*, NorSat-3 requires a receiver with a data rate faster than the nominal 4 kbps used on previous SFL missions. Therefore, NorSat-3 includes a receiver with a data rate of 32 kbps, which provides 8 times the nominal rate of 4 kbps. The main purpose of this receiver is to facilitate payload software updates which would otherwise take too long using the 4 kbps receiver.

This receiver, from a telemetry interface perspective, is one of the few devices on NorSat-3 which does not use NSP to communicate. Therefore, retrieving telemetry values from this device requires not only a new interface on NEMO Control, but also a new application thread in the POBC software to handle the required command and routing of it to the receiver as well as the decoding and routing of the received response back to NEMO Control.

5.5 Payloads

One of the two payloads on NorSat-3 is an automatic identification system receiver, identical to the one included on both the NorSat-1 and NorSat-2 spacecraft. The other payload is a navigation radar detector payload, which will be used for experimental and demonstration purposes.

5.5.1 Automatic Identification System Receiver

The automatic identification system (AIS) is a self-organizing radio communications system used for identifying and geolocating maritime vessels. As per regulation 19 of Safety of Life at Sea [42], the IMO requires AIS transponders to be fitted aboard:

- all ships not engaged on international voyages over 300 gross tonnage,
- all cargo ships not engaged on international voyages over 500 gross tonnage, and
- all passenger ships regardless of size.

The AIS automatically provides information about the ship that it is fitted on to coastal authorities and other ships. This information includes the identity, type, position, course, speed, navigational status, and other safety-related data, which provides countries with maritime surveillance to prevent ship collision and increase shipping safety. Land-based AIS systems are limited to tracking ships within 40 nautical miles, or 75 km, off coast, which leaves large regions of water outside the monitoring ranges [34]. Space-based AIS overcomes this limitation by allowing monitoring of much larger areas, particularly in areas that are difficult to monitor through traditional means, and has been successfully demonstrated via the Smallsat program.

Although space-based AIS has been proven to be more advantageous than ground-based, experience indicates that AIS messages alone may not provide a complete depiction of the maritime traffic. This is due to both unintentional error – as AIS message collisions can occur, resulting in the receiver being unable to correctly decode the received messages – and intentional manipulation – as the AIS may be turned off in certain areas and/or fake messages may be transmitted (spoofing). This problem of missing and/or manipulated AIS messages can only be addressed with complementary sensing technology, which NorSat-3 plans to demonstrate in order to achieve more accurate ship detection and identification.

5.5.2 Navigation Radar Detector Payload

The complementary sensing technology that NorSat-3 will employ is the navigation radar detector (NRD) payload, which will augment ship detection capabilities from the AIS receiver by detecting ship navigation radar signals from space. This payload is capable of locating ships that are either broadcasting incorrect or no AIS data, as well as verifying the AIS messages received. The NRD would make a good complementary sensing technology, because navigation radars are much less likely than AIS transponders to be turned off. Navigation radars are crucial for collision prevention as well as overall ship safety, and turning them off would potentially draw uncomfortable attention due to use of radar required by law [43]. The NRD payload will receive pulses from navigation radars and determine the source location for those radar pulses based on the measured phase offset in the signal between the different antenna elements and information about the spacecraft's position and attitude.

Combining the use of the AIS and NRD payloads will allow the satellite to not only provide more complete and accurate maritime awareness, but also enable detection of ships with erroneous or missing AIS information. Therefore, both payloads will ultimately be used for tracking maritime vessels, but as the AIS is limited to cooperative targets, the NRD will ensure inclusion of non-cooperative targets as well. The NorSat-3 mission aims to show the validity of using this combination of payloads as well as demonstrate the overall NRD technology.

5.6 NEMO Control Interface

Described briefly in **Section 2.3.1** and **Section 4.1**, NEMO Control is the real-time command and control interface used for operating the satellite. It is uniquely configured for each mission and contains interfaces for all the devices on the satellite. Majority of the interfaces within NEMO Control are based off those used for the NorSat-1 and -2 satellites. Therefore, the interfaces of interest in this thesis are ones that are unique to NorSat-3:

- the S-band Rx interface, which is discussed in greater detail in **Section 6.4**, and
- the NRD payload interface, which is discussed in greater detail in **Section 6.5**.

5.7 On-board Computer Application Threads

As mentioned in **Section 2.2.1**, the application software on each OBC uses CANOE, a real-time, multi-threaded OS developed at SFL [44]. The threads running on CANOE, referred to as application threads, are defined as pieces of a program in execution. Each thread is task-specific and is assigned a unique 8-bit address, which allows thread-specific commands/responses to be sent/received. Each thread has a Process Control Block (PCB), where the thread's properties and state information are stored and updated continually. Additionally, each thread is assigned a mailbox, where messages using NSP can be sent and received. CANOE contains a pre-emptive scheduler, which allocates each thread an equal amount of time to execute before moving to the next available thread. Although threads are processed one at a time, the execution of the threads appear simultaneous due to the sufficiently small time slice allocation. For context in the NorSat-3 work presented in this thesis, a function refers to any named section, within a thread, which performs a specific task.

As all OBCs on NorSat-3 are identical from a hardware perspective, each OBC's role as a HKC, ADCC, or POBC is mainly defined by their application threads. The list of application threads that are common across all NorSat-3 OBCs can be found in **Table 8**, while the list of application threads unique to the NorSat-3 POBC can be found in **Table 9**. The author's work discussed in this thesis focuses specifically on:

- the S-band Rx Thread, which is discussed in greater detail in **Section 6.4**, and
- the NRD Thread, which is discussed in greater detail in **Section 6.6**.

Table 8: Common Application Threads on NorSat-3 OBCs

THREAD NAME	THREAD DESCRIPTION
Command Decoding and Execution Thread	Provides interface for querying and setting configuration and status parameters via NEMO Control interface.
Communications Thread	Responsible for routing messages between spacecraft interfaces and the various application threads.
Memory Management Thread	Provide read/write access to static RAM, flash file system, and memory-mapped hardware registers.
Persistent Flash File System Thread	Responsible for the ground interface with the flash file system on the OBC by handling commands related to OBC's file listings, creation of new files, uploading/downloading of files, and/or deletion of files.
Compression Thread	Used for reducing sizes of files generated by the OBC to be downlinked to ground and, conversely, for decompressing files that have ben compressed prior to being uplinked to the OBC.
Time Thread	Responsible for managing the system time in the CANOE OS, where the time can be set from either the ground or from a fine source.
Device Threads	CAN Thread: manages interface between on-board software and the CAN bus. I^2C Thread: provides communication interface for devices connected to processor's I^2C communication lines. SPI Thread: provides communication interface for devices connected via SPI. SCC Thread: manages interface for the SCC on the OBCs, which handles satellite communications with ground over the satellite's receiver(s) and transmitter(s). UART Thread: provides communication interface for devices connected via UART and is capable of handling both NSP and non-NSP communications.
GPS Interface Thread	Provides an interface between ground and the GPS receiver, as the GPS receiver does not communicate using NSP as NEMO Control does.

Table 9: Application Threads Unique to NorSat-3's POBC

THREAD NAME	THREAD DESCRIPTION
Payload Power Thread	Provides an interface between power system and payloads. This thread contains extra safety parameters that will automatically switch off a payload when required and/or prevent the operator from manually turning on a payload when not deemed safe.
S-band Rx Thread	Provides an interface between NEMO Control and the S-band receiver, as the telemetry system on the S-band receiver does not communicate using NSP.
AIS Receiver Thread	Responsible for handling various types of data products outputted by the AIS receiver, forwarding and/or logging that data, controlling the redundancy control line, and executing processes related to uploading/downloading of payload images.
NRD Thread	Responsible for handling various types of data products outputted by the NRD payload, forwarding and/or logging that data, and executing processes related to uploading/downloading of payload images.

Chapter 6

NorSat-3: Command and Data Handling

This chapter defines the author's goal and objectives for the NorSat-3 portion of this thesis and presents the author's contributions to the command and data handling subsystem on NorSat-3, focusing on the work that can be implemented and utilized on future SFL missions. Specifically, these new developments include the design of the new serial interface board, the application thread and NEMO Control interface for the high-speed S-band receiver telemetry, the NEMO Control interface for the NRD payload, and the application thread responsible for handling commands not supported by NEMO Control as well as handling various data products output by the payload.

6.1 NorSat-3 Ground Station Software

The work presented in this chapter refers to a variety of ground station equipment and software described in **Section 2.3**, **Section 5.6**, and **Section 5.7**. **Figure 15** on the next page provides a visual and high-level overview of how all these different software programs and equipment interface with one another for the NorSat-3 mission. Relevant software programs that have been presented in previous sections are: NEMO Control, QMTP, MuxStation, MuxMaster, and MuxControl.

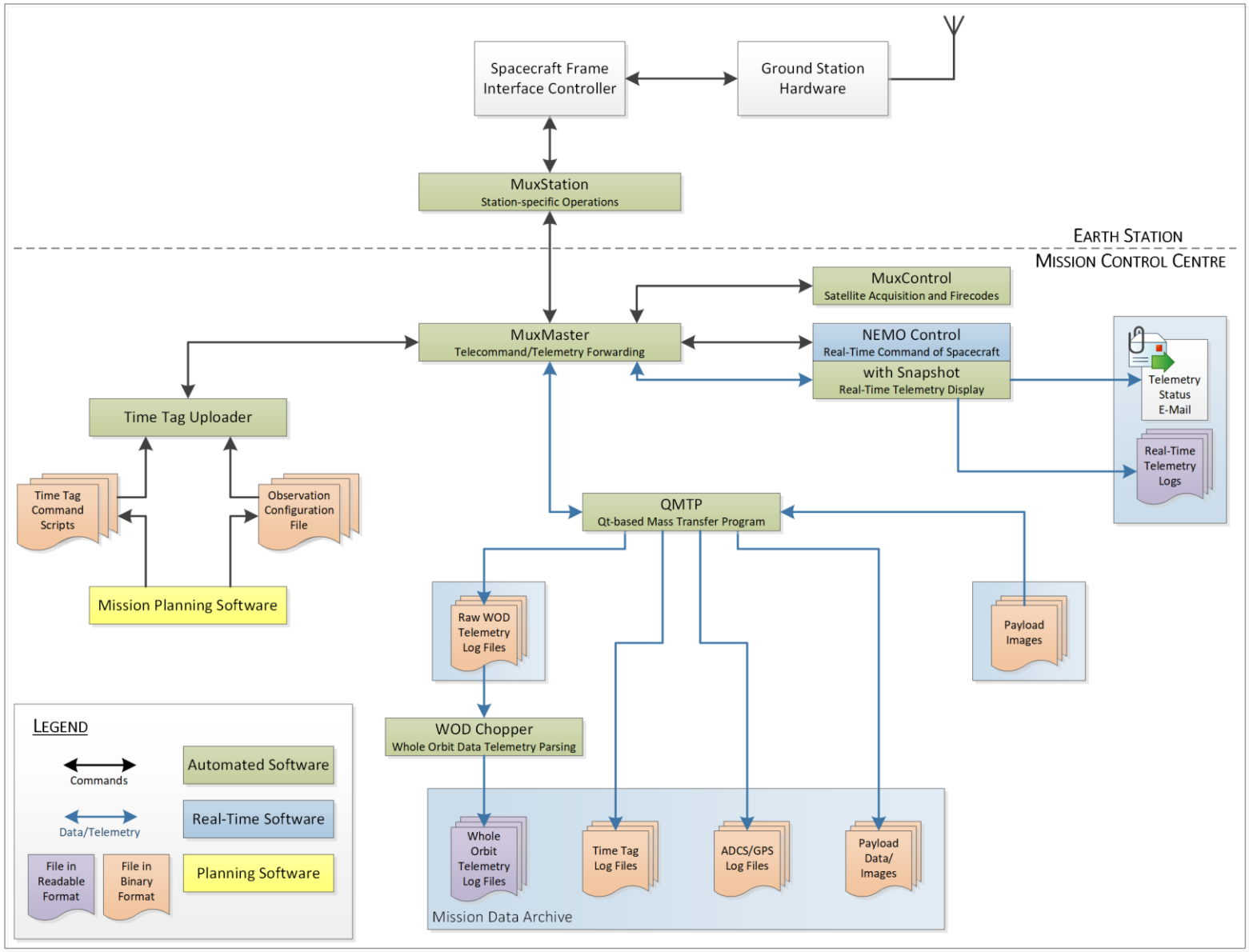

Figure 15: NorSat-3 Ground Station Software

6.2 NorSat-3 Thesis Goal and Objectives

The main goal for the NorSat-3 portion of this thesis was to design, develop, and test the overall C&DH subsystem for the mission. This goal was broken up into the following objectives, where the ones covered in this thesis are not italicized:

- design, manufacture, and test the serial interface board (SIB),
- *configure the application software for the HKC, ADCC, and POBC*,
- *configure the ground station software programs for NorSat-3 operations*,
- develop the application thread and ground control interface for retrieving high-speed S-band receiver telemetry,
- develop the ground control interface for the new NRD payload,
- develop the application thread for handling any payload commands not supported by NEMO Control as well as for handling data products output by the payload, and
- *develop the long form function test scripts and procedures and perform validation with the flatsat*.

6.3 Serial Interface Board

One of the main NorSat-3 objectives for this thesis, as listed in **Section 6.2**, was designing, manufacturing, and testing a new serial interface board (SIB) for the mission. Although the majority of the hardware components on NorSat-3 are identical to those on its predecessors, NorSat-1 and -2, a new SIB design was needed to accommodate the requirements imposed by the new NRD payload.

A SIB translates 3V CMOS signals to RS-485 differential signals. Specifically, it connects the serial and general-purpose input/output (GPIO) connections from the OBCs to the payloads, providing an interface between them. At a high-level, the main differences between the SIB designed for NorSat-3 and the SIB used on NorSat-1/-2 are:

- different PPS outputs as required by the new NRD payload,
- 4-layer printed circuit board (PCB) compared to 2-layer used on previous designs, and
- flexible design for use on future SFL missions.

A pulse per second (PPS) is an electrical signal with an abruptly rising or falling edge that accurately repeats every second. PPS signals are usually output by devices such as radio beacons, precision oscillators, and global positioning system (GPS) receivers. As per *NS3-ACS-R018* in **Table 7**, the customer requires the PPS for the NRD payload to come directly from the GPS receiver to ensure better PPS stability. On the other hand, the PPS for the AIS payload is required to come from the OBCs to ensure an uninterrupted signal. The SIB used on NorSat-1 and -2 allows the PPS to be output from the OBCs, but not directly from the GPS receiver; therefore, one of the drivers for the new SIB design was to allow output of a PPS signal from the GPS receiver as well as from the OBCs.

As a result of a new board design, new PCBs had to be manufactured. Therefore, another decision, which differentiates this SIB from previous SIBs, was the number of layers on the PCB. The previous designs were manufactured on 2-layer PCBs, mainly due to the significant cost difference of adding more layers at the time of manufacture. However, now, the cost difference between a 4-layer PCB and 2-layer PCB is minimal, especially considering the benefits provided by the extra layers. This SIB design, in particular, is very small (10 cm × 10 cm) and, therefore, densely designed in terms of traces and parts. Using a 4-layer PCB, where one layer is entirely dedicated for a power plane and another for ground, allows higher levels of signal integrity and reduces interference.

To further optimize on cost and resources, design flexibility – in terms of compatibility with future missions – was another big aspect of the overall design process. This SIB design, from NorSat-3's perspective, comprises:

- three input-output channels, where each channel connects all three OBCs to the receive/transmit lines on the AIS receiver, the command receive/transmit lines on the NRD payload, and the data receive/transmit lines on the NRD payload.
- four general output channels, where each channel takes a general output from any of the three OBCs to provide to any of the two payloads, for instance, a PPS signal from the OBC to the AIS.
- two PPS output channels, where each channel takes the PPS signal from the GPS receiver to provide to any of the two payloads, in this case, the NRD payload.

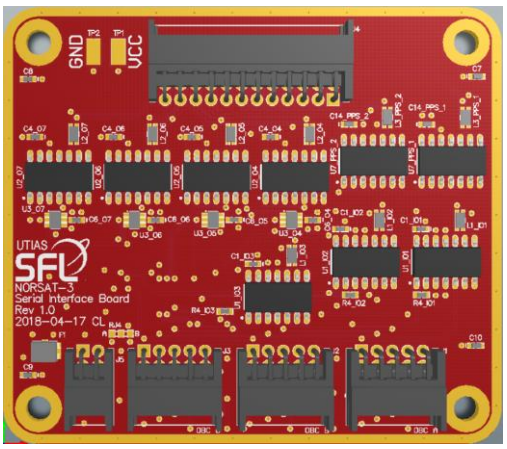

Figure 16: NorSat-3 SIB (Front View)

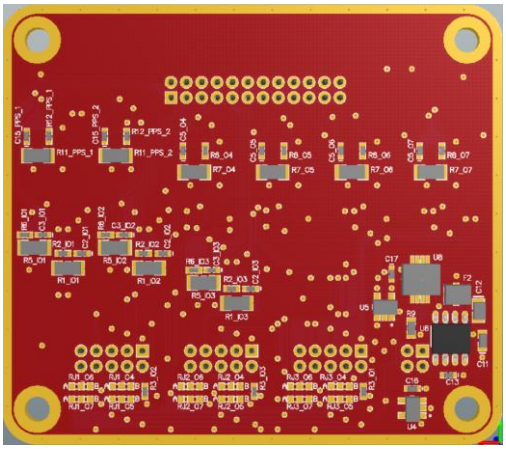

Figure 17: NorSat-3 SIB (Back View)

This design, therefore, allows up to three different payloads, each with one receive and one transmit line, to be connected to up to three OBCs each; up to four general outputs, such as a PPS signal from an OBC, to be fed to any of the payloads from up to three OBCs each; and a PPS signal output, directly from a GPS receiver, to any two payloads.

6.4 High-Speed S-band Receiver Telemetry

The telemetry interface on the high-speed S-band receiver is a simple UART-SPI bridge to an on-board multi-channel ADC, which the POBC interfaces with via UART. As mentioned in **Section 5.4**, the telemetry interface on this receiver does not communicate over NSP, which NEMO Control is designed to do. Therefore, the implementation of retrieving telemetry from this device involves more than a simple configuration on NEMO Control.

The method proposed and selected was to implement a new application thread, on the POBC software, capable of sending the required command, receiving the telemetry values output by the receiver, and packaging it in NSP format before sending back to the NEMO Control interface to receive, parse, and display. The main idea is to use the POBC as the interface between NEMO Control and the S-band receiver.

As shown in **Figure 18** on the next page, an interface was set up in NEMO Control for requesting and displaying the S-band receiver telemetry. The telemetry is requested via the ‘Get Telemetry’ button and the received values are parsed and displayed in the window.

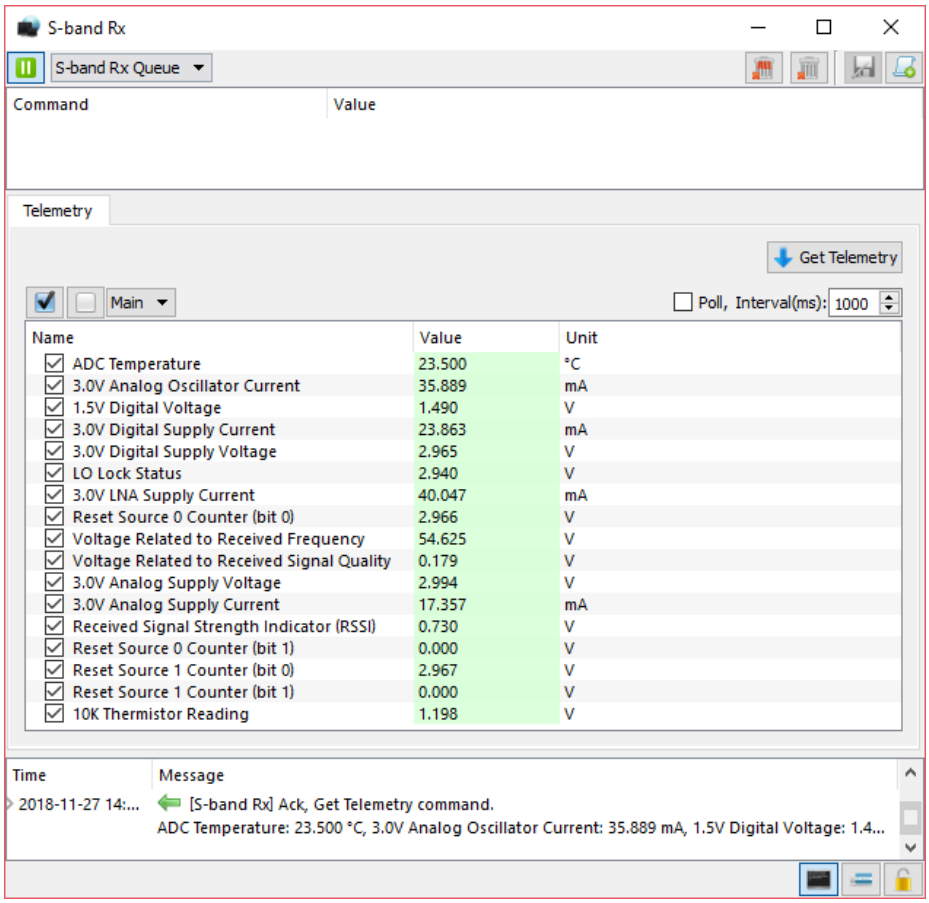

Figure 18: NEMO Control – S-band Rx Interface

As illustrated in **Figure 19**, the 'Get Telemetry' button, when clicked, prompts NEMO Control to send a custom command to the S-band Rx application thread on the POBC. Upon receiving this command packet from NEMO Control, the receiver thread generates a sequence of commands that the device needs to receive in order to initiate the telemetry generation. The receiver thread then sends this list of commands in NSP to the UART thread, specifying to the thread to send the packets to the receiver using non-NSP format. By doing this, the S-band receiver is able to receive and correctly interpret the command packets. Promptly after this command sequence is sent, the S-band receiver thread sends out a series of `0x00` commands, totaling up to 34 bytes, which prompts the S-band receiver to output its telemetry values, corresponding to 17 2-bytes values. The UART thread receives each telemetry value in non-NSP format, packages them in NSP wrappers, and routes them to the S-band Rx Thread. Once received on the S-band Rx Thread, the telemetry values are decoded and sent to NEMO Control in NSP. The final conversions are then applied to all the telemetry values received on the NEMO Control interface and displayed in the device window.

Both the NEMO Control interface and application thread for this S-band receiver can be easily configured and implemented on any future mission using receivers with an identical or similar type of ADC system.

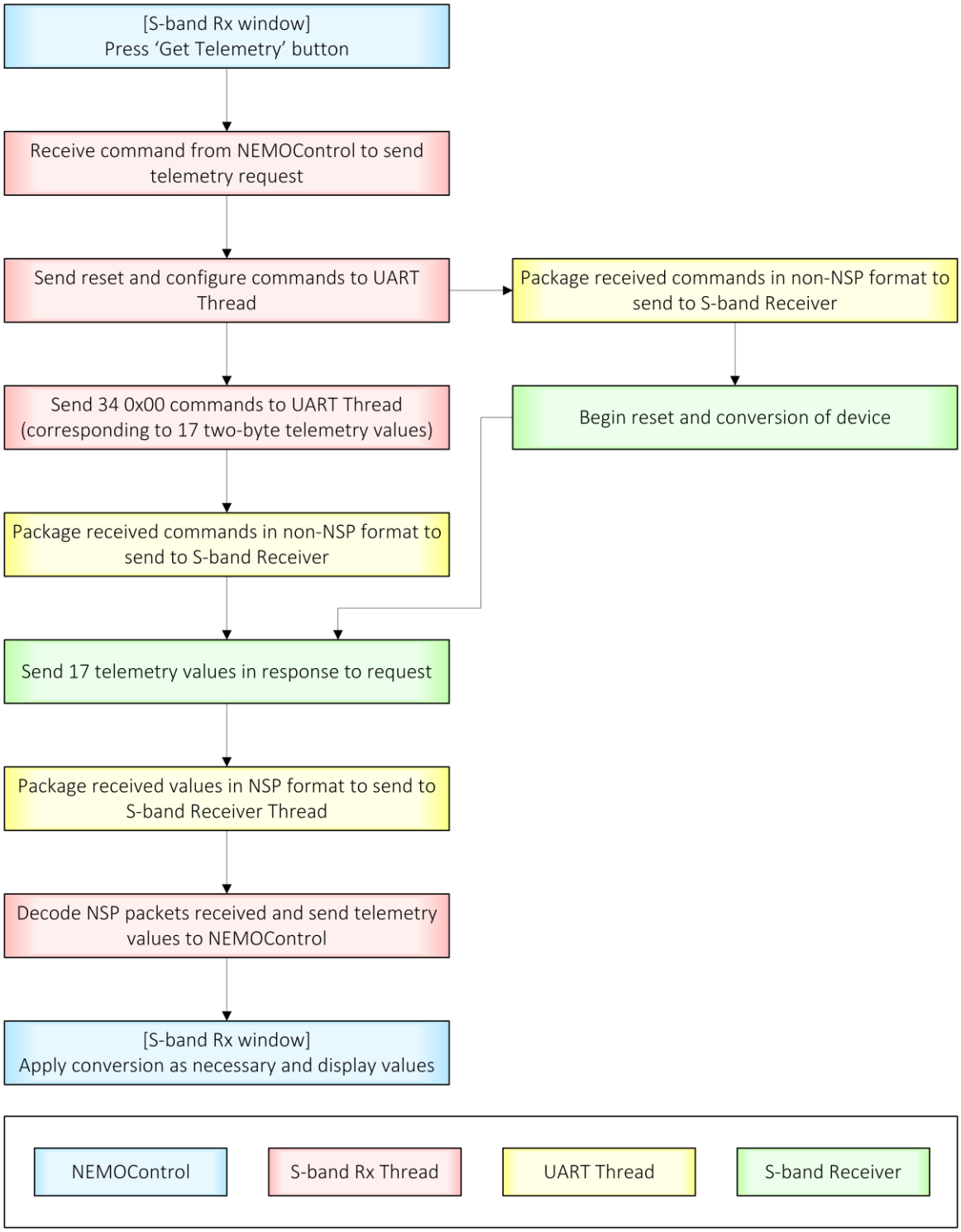

Figure 19: S-band Receiver Telemetry Retrieval Flow

6.5 NRD Payload Interface

The NRD payload uses NSP for communications, which includes all commands and responses to and from the NRD, as well as all data products sent from the NRD to the POBC. All payloads have a unique set of commands, which are outlined in an interface control document (ICD) provided by the manufacturers of that device. One of the author's main objectives for the NorSat-3 portion of this thesis was to develop a NEMO Control interface for the new NRD payload that would provide the operator with a user-friendly interface for command and control of the payload as per the ICD.

The NRD interface in NEMO Control, presented in this section, was created using Qt Designer, which is a tool used for designing and building graphical user interfaces [45]. This program allows users to compose and customize widgets and dialogs.

In total, there are six tabs in the NRD interface, and the description for each can be found below. Screenshots of the tabs that are relevant in the subsequent sections can be found in the following pages.

- **General tab:** shown in **Figure 20**, provides interface for setting payload baud rates, swapping payload data files, and accessing various parameters in the NRD application thread (discussed in greater detail in **Section 6.6**),
- **Manual tab:** contains a list of all commands as per the NRD ICD that the operator can use manually,
- **Data tab:** provides interface for commands related to output of payload data products,
- **Telemetry tab:** provides interface for main and extended telemetry values of the payload,
- **Image Control tab:** shown in **Figure 21**, provides interface for commands related to payload application images, and
- **Boot Settings tab:** provides interface for setting default and protection settings for the payload.

This section provides a high-level overview of the NEMO Control interface for the NRD payload, which can easily be used on future missions with the same payload.

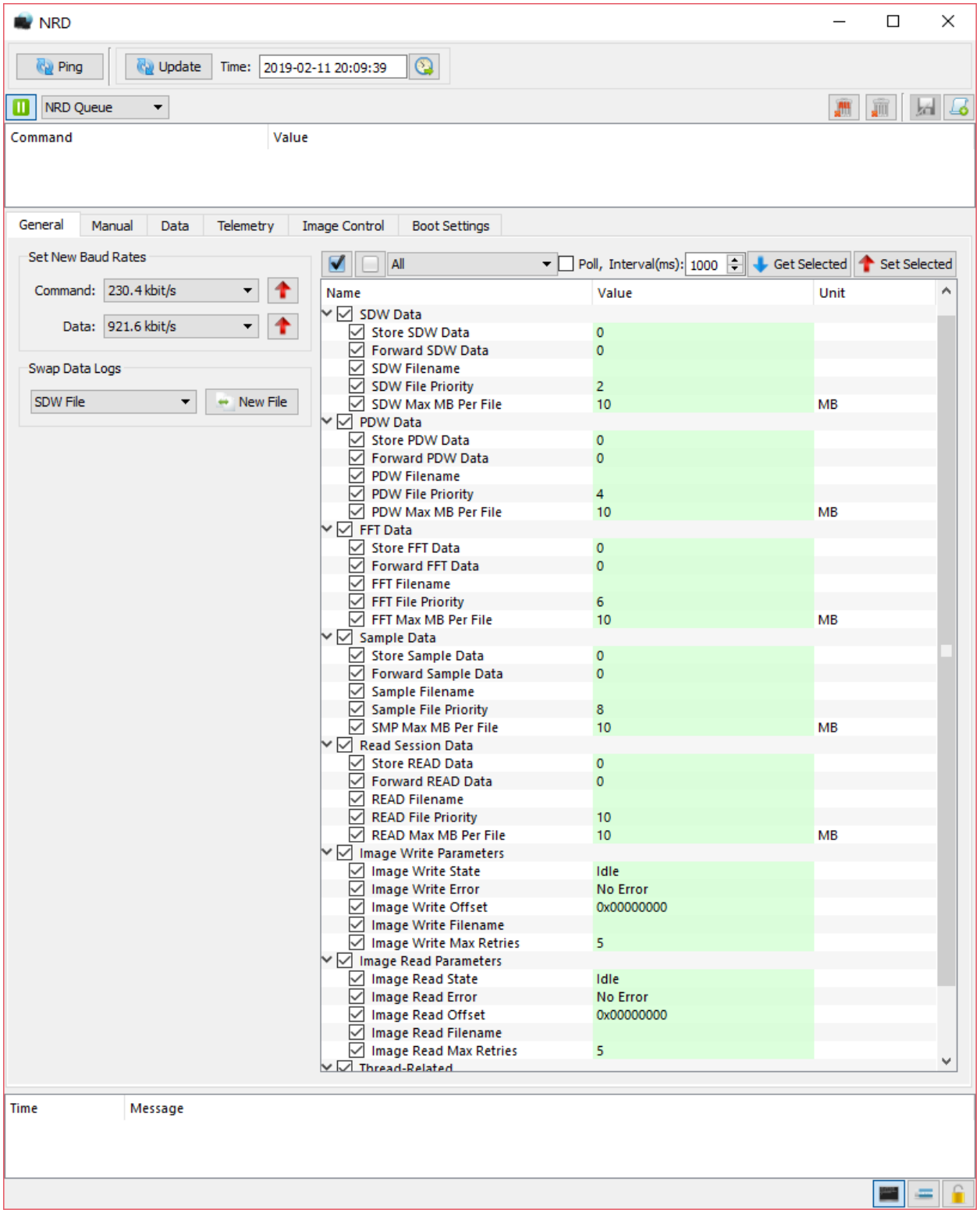

Figure 20: NEMO Control – NRD Interface (General Tab)

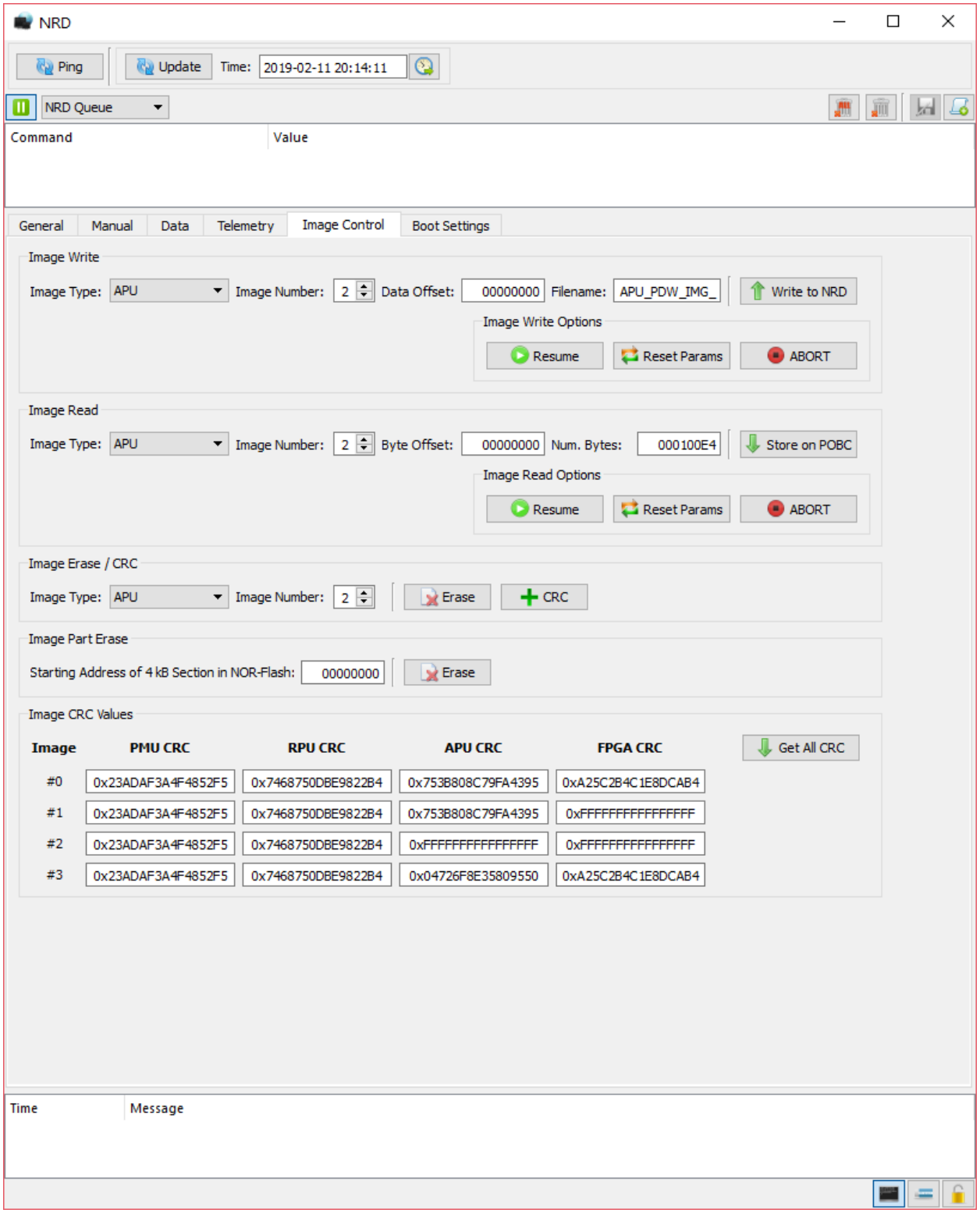

Figure 21: NEMO Control – NRD Interface (Image Control Tab)

6.6 NRD Thread

NS3-SYS-R007, *NS3-SYS-R009*, and *NS3-SYS-R010*, in **Table 7**, specify requirements on interfacing with the NRD payload, which involves commands as well as data handling. Since the NRD is a new payload to be flown on NorSat-3, one of the author's objectives included developing a new application thread, in addition to the interface discussed in **Section 6.5**, to accommodate any payload commands not supported by NEMO Control as well as payload data sent to the POBC.

At a high-level, the NRD Thread is capable of:

- acting as a middleman between the NEMO Control interface and the NRD payload for executing any commands that require additional preparation and processing than what NEMO Control is capable of,
- forwarding various types of data output by the NRD payload to ground during a pass, and
- storing various types of data output by the NRD payload into files on the POBC file system.

As shown in **Section 6.5**, although most of the commands in the ICD can be and are implemented in the NEMO Control interface directly, there are a few that are more practical and sensible to implement as a process rather than a command; the tradeoff being that this additional preparation and processing must be implemented in the application thread. Specifically, these commands refer to those related to the uploading/downloading of payload software application images to/from the payload. This is not only useful to implement for the NRD payload, but any payload, such as the AIS receiver, as these are generally common commands across payloads. To highlight the extendibility of the author's work on this, the implementation of the image-related commands as processes for the NRD payload, discussed in greater detail in **Section 6.6.1**, has also been implemented for the AIS receiver.

The other necessary responsibility of the NRD Thread is to handle the various types of data that the NRD outputs to the POBC. This includes implementing functions that ensure both transmission of data to ground directly during ground contact and/or storing of that data in files on the POBC file system to be downloaded later. As the specific types of NRD data products are proprietary, **Section 6.6.2** discusses the implementation of data handling functions on the NRD Thread from a general perspective.

6.6.1 Payload Image Upload/Download

A payload typically comprises a list of unique commands, defined in their respective ICD, which are used to operate the device. Among these commands, both the AIS receiver and the NRD payload on NorSat-3, as well as most payloads, contain commands related to the payload application images. These image-related commands most commonly include:

- an **IMAGE_WRITE** command, which is used to write partial or complete software application images to the flash memory of the payload, and
- an **IMAGE_READ** command, which is used to read partial or complete software application images in the flash memory of the payload.

The general format of the command and response packets of these commands can be found in **Table 10** and **Table 13**, respectively. The general format of the Data portion of the IMAGE_WRITE and IMAGE_READ packets can be found in **Table 11** and **Table 12** for the command, respectively, and in **Table 14** and **Table 15** for the response, respectively.

Table 10: IMAGE_WRITE/IMAGE_READ Command Packet Format (General)

PACKET FIELD	SIZE (BITS)	VALUE	COMMENT
DESTINATION ADDRESS	8	`0xYY`	Address of payload.
SOURCE ADDRESS	8	`0xNN`	Can be any.
P/F BIT	1	0 or 1	Payload to respond if set to 1.
B BIT	1	0 or 1	Determined by the protocol.
A BIT	1	N/A	Ignored by the payload.
COMMAND	5	`0x00 – 0x1F`	Unique to payload.
DATA	Varies	-	See **Table 11** for IMAGE_WRITE. See **Table 12** for IMAGE_READ.
CRC	16	`0xNN`	Unique to each packet.

Table 11: IMAGE_WRITE Command Data Field (General)

DATA FIELD	SIZE (BITS)	COMMENT
IMAGE SELECTION	Varies	Specifies which image is being written to.
ADDRESS OFFSET	Varies	The address offset of the first byte in following data.
DATA	Varies	Image data (typically 0 – 260 bytes).

Table 12: IMAGE_READ Command Data Field (General)

DATA FIELD	SIZE (BITS)	COMMENT
IMAGE SELECTION	Varies	Specifies which image is being read.
BYTE OFFSET	Varies	Specifies where to start the read in the selected image.
NUMBER OF BYTES	Varies	Specifies number of bytes to read and return.

Table 13: IMAGE_WRITE/IMAGE_READ Response Packet Format (General)

PACKET FIELD	SIZE (BITS)	VALUE	COMMENT
DESTINATION ADDRESS	8	`0xNN`	Address of command source.
SOURCE ADDRESS	8	`0xYY`	Address of the payload.
P/F BIT	1	1	Single packet reply.
B BIT	1	0 or 1	Value of command's B-bit.
A BIT	1	0 or 1	0 for non-acknowledgement. 1 for acknowledgement.
COMMAND	5	`0x00 – 0x1F`	Unique to payload.
DATA	Variable	Varies	A-bit = 0: Error code is returned. A-bit = 1: See **Table 14** for IMAGE_WRITE. A-bit = 1: See **Table 15** for IMAGE_READ.
CRC	16	`0xNN`	Unique to each packet.

Table 14: IMAGE_WRITE Acknowledged Response Data Field (General)

DATA FIELD	SIZE (BITS)	COMMENT
IMAGE INFORMATION	Varies	Some information in image that was written.

Table 15: IMAGE_READ Acknowledged Response Data Field (General)

DATA FIELD	SIZE (BITS)	COMMENT
IMAGE SELECTION	Varies	Copy of image selected to be read.
ADDRESS	Varies	The location of first byte in following message.
DATA	Varies	Image data (typically 0 – 260 bytes).

Using the NRD payload as an example, expected sizes of application images range from 55 kB to as large as 13 MB. As mentioned previously, an optimal maximum size for the data field of a packet is around 250 – 260 bytes. This means writing an image via traditional IMAGE_WRITE command packets, as per **Table 10**, would take approximately somewhere between 220 and 52,000

packets. Not only is this a tedious task, but a more significant issue is that uploading an image to the NRD payload will most likely require multiple passes, even with the new high-speed receiver on NorSat-3. As a result, the payload may have to be on when not required and the possibility of the image being corrupted will be higher.

From the IMAGE_READ perspective, although the response packets, presented in **Table 15**, are still limited to around 250 bytes of data per packet, issues seen with IMAGE_WRITE aren't particularly applicable. This is mainly attributed to the fact that downlink speeds are typically significantly faster than the uplink and there is no risk of corrupting an image in a read process. However, it is much more useful and practical to download an image or parts of it as a single file all at once as opposed to reading out 250 bytes of it, section by section.

Therefore, a safer and more efficient method of implementing these two commands was researched and implemented. The selected method was to design the payload application thread to be capable of executing the image upload and download processes, as opposed to having the operator perform them, using the respective commands. Similar to how the S-band receiver telemetry retrieval process was implemented, as explained in **Section 6.4**, the NRD interface in NEMO Control contains custom commands that can be sent to the payload thread and the payload thread is coded to execute various functions upon receipt of those commands. At a high level, the implementation of the two commands as automated processes via the POBC, which are explained in greater detail in the following sub-sections, are as follows:

- **Image Upload via POBC:** First, upload the payload image to the POBC file system via QMTP. Then send the custom write command via the NEMO Control payload interface, specifying the filename of the image uploaded to the POBC file system. The payload thread will then send the contents of the image in the file system as a series of IMAGE_WRITE commands, as per the ICD, until the file is fully written to the payload.
- **Image Download via POBC:** Send the custom read command via the NEMO Control payload interface, specifying which payload image type and number is desired for download. The payload thread will create a file – with a name based on the image type – in the POBC file system, then read the specified payload image using a series of IMAGE_READ commands and write all received image data to the file. The operator will then be able to download the read-out image as a file via QMTP.

The benefits of implementing the IMAGE_WRITE and IMAGE_READ commands such that they are executed as automated processes via the POBC are as follows:

- the QMTP software is very well tested and has been used on many successful missions,
- when uploading files to the OBCs via the QMTP, a response is not required for every packet sent, unlike the IMAGE_WRITE command as per the ICD; therefore, the effective uplink rate is better uploading via QMTP than uploading directly to the payloads via the IMAGE_WRITE commands,
- the upload and download processes in the QMTP software are fully automated and capable of handling partial uploads/downloads, which means transfers of large files can be executed without the need of operator intervention, and
- the actual uploading/downloading of an image from/to the file system to/from the payload is executed autonomously via the payload thread once commanded to do so.

As opposed to manually composing and sending IMAGE_WRITE/IMAGE_READ commands, programming the payload thread to be capable of performing the payload image uploads/downloads via the POBC is much more useful from an overall mission perspective. This method provides a reliable interface to facilitate any software updates as well as any debugging processes the payload(s) may need during the mission lifetime. In addition, this particular implementation supports additional commands, explained in the following subsections, which allow operators to pause and resume an upload/download at any point, thus, granting them freedom to execute other payload-related commands during the process. This work can easily be configured and used for other payloads on future missions, given that other payloads typically have similar structures for image-related commands. To confirm the extendibility of the author’s work, the image upload/download processes via POBC have also been tested and successfully implemented on the AIS Receiver Thread for the AIS receiver.

6.6.1.1 Image Control via POBC

To implement the image upload and download processes as custom commands, the respective NEMO Control interface and payload application thread have been programmed to do so. All code in the payload thread associated with image uploading/downloading are implemented via a state machine. This involves various functions that handle the image upload/download processes, described in **Table 18**, as well as the use of two structures, *IWriteInfo* and *IReadInfo*, which contain continuously updated parameters relevant to the write and read process, respectively, as described in **Table 19**. At a high level, state machines execute code and perform transitions between states while properly calling required functions based on the event that occurred and its current state.

The various states of the image upload process include:

- ***IWrite_Idle:*** when no upload process is underway,
- ***IWrite_Ping:*** when the thread is pinging the payload to ensure it is on before commencing or resuming an image upload process,
- ***IWrite_GetInfo:*** when the thread is opening the specified file – containing the image to upload – in the POBC file system and retrieving relevant information, and
- ***IWrite_Writing:*** when the thread is uploading the contents of the file to the payload.

Table 16 outlines the transitions between the various states and the combination of the event and current state that must hold for the transition to occur during an image upload.

Table 16: State Transition Table for Image Upload via POBC Process

From \ To	IWRITE_IDLE	IWRITE_PING	IWRITE_GETINFO	IWRITE_WRITING
IWRITE_IDLE	RESET	START RESUME	-	-
IWRITE_PING	TIMEOUT ABORT	-	PING_ACK	-
IWRITE_GETINFO	GETINFO_NACK ABORT	-	-	GETINFO_ACK
IWRITE_WRITING	COMPLETE TIMEOUT ABORT WRITE_NACK	-	-	WRITE_ACK

Similar to the upload process, the various states of the image download process include:

- ***IRead_Idle:*** when no download process is underway,
- ***IRead_Ping:*** when the thread is pinging the payload to ensure it is on before commencing or resuming an image download process,
- ***IRead_CreateFile:*** when the thread is creating and opening a file in the POBC file system to store image data to, and
- ***IRead_Reading:*** when the thread is reading and storing the specified image from the payload to the file in the POBC file system.

Table 17 outlines the transitions between the various states and the combination of the event and current state that must hold for the transition to occur during a download process.

Table 17: State Transition Table for Image Download via POBC Process

From \ To	IREAD_IDLE	IREAD _PING	IREAD _ CREATEFILE	IREAD _ READING
IREAD _IDLE	RESET	START RESUME	-	-
IREAD _PING	TIMEOUT ABORT	-	PING_ACK	-
IREAD _CREATEFILE	CREATEFILE_NACK ABORT	-	-	CREATEFILE_ACK
IREAD _READING	COMPLETE TIMEOUT ABORT READ_NACK	-	-	READ_ACK

Table 18: Functions used for Image Upload/Download via POBC

FUNCTION NAME	DESCRIPTION
FUNCTIONS TRIGGERED BY NEMO CONTROL PACKET	
Cmd_WriteStart Cmd_ReadStart	Function called by 'Write to NRD' or 'Store on POBC' buttons, respectively, in 'Image Control' tab of NRD NEMO Control window as per **Figure 21**.
Cmd_WriteReset Cmd_ReadReset	Functions that handle the reset of IWriteInfo and IReadInfo parameters as per **Table 19**, respectively.
Cmd_WriteResume Cmd_ReadResume	Functions that handle the resume of writing an image to or reading an image from the payload.
Cmd_WriteAbort Cmd_ReadAbort	Functions that stop the current write or read process, respectively.
FUNCTIONS TRIGGERED BY PAYLOAD PACKET	
Process_Ping	Function that handles PING response packets from the payload.
Process_Write	Function that handles IMAGE_WRITE response packets from the payload.
Process_Read	Function that handles IMAGE_READ response packets from the payload.
FUNCTIONS TRIGGERED BY STATE MACHINE EVENT	
SM_Idle	Empty function which represents an 'Idle' state for the state machine. Typically called when an abort, timeout, error, or NACK occurs during a write/read process.
SM_CleanupIdle	Function responsible for resetting all parameters before going into idle state. Typically called on reset.
SM_Ping	Function responsible for sending a PING command to the payload. Typically called before starting or resuming a write or read process.
SM_GetInfo	Function responsible for opening and retrieving relevant information of the image file in POBC file system to be uploaded to the payload. Called after PING is ACKed for starting a write process.
SM_CreateFile	Function responsible for creating file in POBC file system to store image data to during an image download process. Called after PING is ACKed for start a read process.
SM_WriteNext SM_ReadNext	Functions responsible for sending the next IMAGE_WRITE or IMAGE_READ commands as per the IWriteInfo or IReadInfo parameters, respectively. Called upon WRITE_ACK or READ_ACK via the Process_Write and Process_Read functions.
SM_WriteComplete SM_ReadComplete	Functions responsible for closing the file associated with the write or read process as well as resetting the parameters before returning to the idle state. Called when SM_WriteNext or SM_ReadNext has no more bytes to write or read, respectively.
SM_WriteTimeout SM_ReadTimeout	Alarms are set whenever a command packet is sent to the payload: PING, IMAGE_WRITE, or IMAGE_READ. These alarms are triggered, which calls either of these two functions, when the thread does not receive an expected response from the payload within a time range. These functions, based on the number of times the command has already been resent, will either terminate the process with an error or resend the respective command.

Table 19: Parameters in IWriteInfo/IReadInfo Structure

STRUCTURE PARAMETERS	DATA TYPE	DESCRIPTION	
State	UINT8	Specifies the state of the upload/download process:	
		0: IWrite_Idle	4: IRead_Idle
		1: IWrite_Ping	5: IRead_Ping
		2: IWrite_GetInfo	6: IRead_CreateFile
		3: IWrite_Writing	7: IRead_Reading
Error	UINT8	Specifies the error, if any, that has stopped the process.	
		0: No Error	No error encountered (default).
		1: Event	Wrong event sent.
		2: Ping NACK	Payload responded with NACK to ping command.
		3 – 9: File	Error associated with file opening, closing, writing, etc.
		10 – 11: Write	Payload responded with NACK to IMAGE_WRITE command.
		12: Read	Payload responded with NACK to IMAGE_READ command.
		13: Buffer	Buffer issue with NSP packets.
		14: Max Retries	Command resent too many times without response.
		15: Aborted	Process aborted by the operator.
Offset	UINT32	Offset value in the data field of the next IMAGE_WRITE or IMAGE_READ command packet to be sent.	
FileName	UCHAR	Name of the file in file system to upload to payload or store image data to.	
BytesRemaining	UINT32	Number of bytes left to write or read to or from the payload.	
BytesExecuted	UINT32	Number of bytes written or read in the latest IMAGE_WRITE or IMAGE_READ command.	
Retries	UINT8	Maximum number of retries allowed for a command sent to the payload. Default value is 5 retries, but this parameter is user configurable.	
Type	UINT8	Image selection and number.	
Resuming	BOOL	True if process is resuming from last terminated place. False if process is starting from beginning.	
TimeoutAlarm	ALARM*	Pointer to an alarm that is set every time a command is sent to the payload. If response is not received before alarm goes off, the command is considered to have timed out, and command will be resent until the maximum number of retries is reached.	
TimeoutCmd	UCHAR	Command that triggered the alarm.	

As a result of implementing the IMAGE_WRITE/IMAGE_READ commands as upload/download processes via the POBC, other commands, which are not necessarily applicable when using the commands manually, are required. These additional custom commands for both the upload and download processes can be found in **Figure 21** under the 'Image Write' and 'Image Read' sections, respectively, in the 'Image Control' tab of the NRD NEMO Control interface. In total, the custom commands related to an image upload and download include:

- **Write/Read:** initiated by the 'Write to NRD'/'Store on POBC' button, which commands the NRD thread to perform the image upload/download with the specified image, explained in greater detail in **Section 6.6.1.2**.
- **Resume:** initiated by the 'Resume' button, which commands the NRD thread to continue an image upload/download that was terminated/aborted before completion, explained in greater detail in **Section 6.6.1.3**.
- **Abort:** initiated by the 'ABORT' button, which commands the NRD thread to terminate the currently running image upload/download process, explained in greater detail in **Section 6.6.1.4**.
- **Reset:** initiated by the 'Reset Params' button, which commands the NRD thread to reset all the parameters in the *IWriteInfo*/*IReadInfo* structure, explained in greater detail in **Section 6.6.1.5**.

6.6.1.2 Write/Read Command Implementation

In order to execute the image upload process as per the method outlined in **Section 6.6.1**, both the NEMO Control interface and the application thread for the NRD payload have been implemented to facilitate this. Although the NRD payload and its images are used to explain the author's work, it is important to note, as mentioned previously, all work discussed in this section can be easily configured and extended for other payloads.

The image upload process begins by uploading the desired image to the POBC file system via QMTP. As an example, a screenshot of the QMTP interface showing the POBC file system with an uploaded NRD image can be found in **Figure 22**, where the desired image to write in this example is named 'APU_PDW_IMG_'.

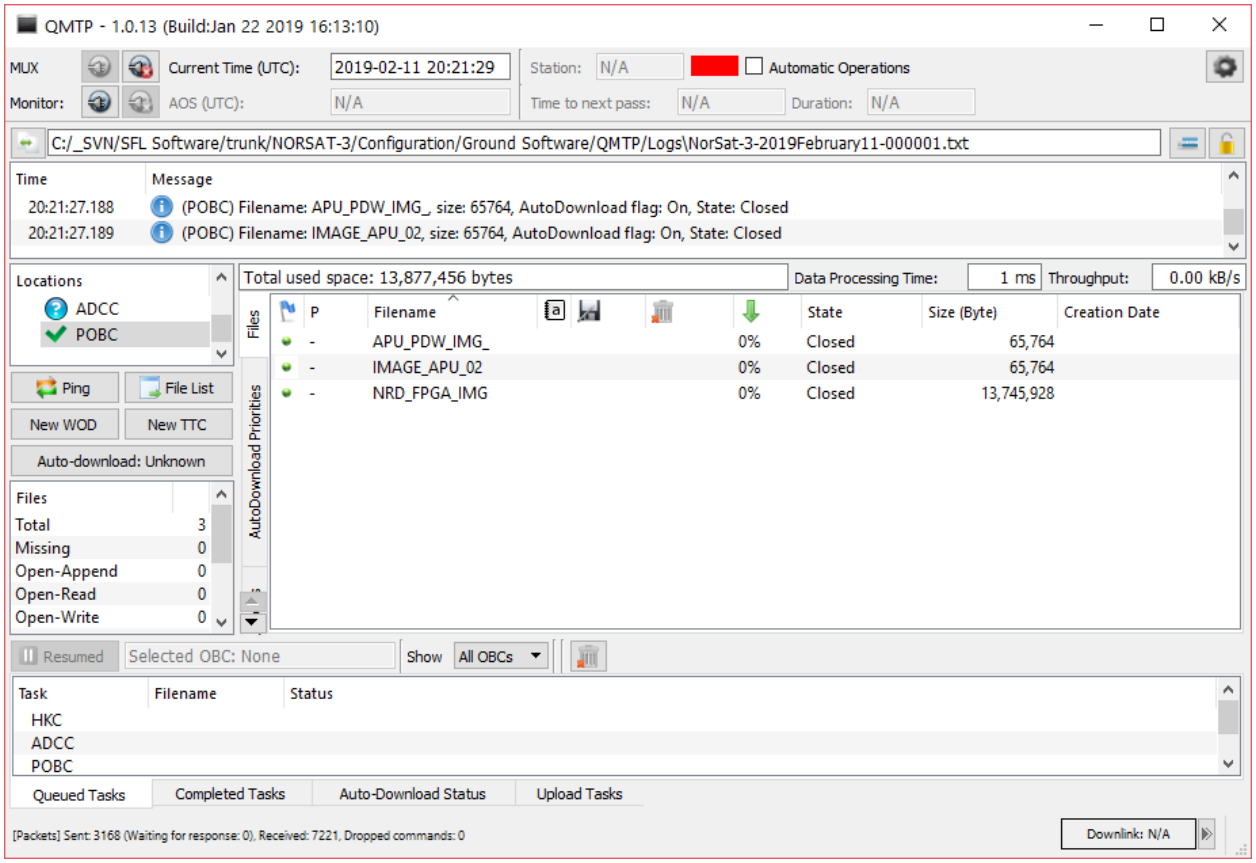

Figure 22: Image Upload/Download via POBC – QMTP

Once the desired image is successfully uploaded to the POBC file system, the operator uses the custom write command implemented in the 'Image Write' section located in the 'Image Control' tab of the NRD interface in NEMO Control, shown in **Figure 21**.

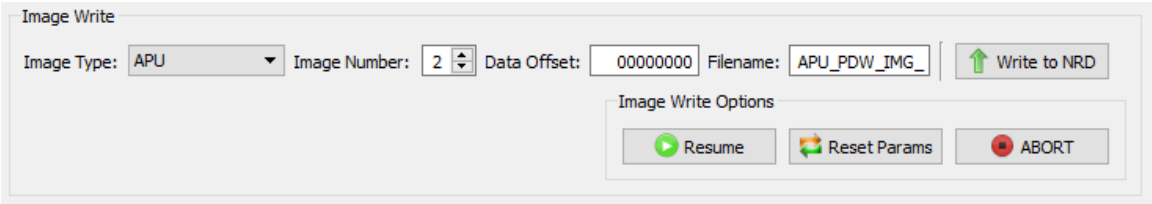

Figure 23: Image Upload via POBC – Write Command (NRD Interface)

Figure 23 provides a closer look at the write command, where the operator is required to specify:

- image type and number that they wish to perform the image upload to,
- the address of the image slot that they wish to begin the write from, and
- the name of the file in the file system that they wish to upload to the payload.

Once all the required fields are filled, the 'Write to NRD' button is pressed to initiate the upload process as laid out in **Figure 24**. The button sends a custom command to the NRD Thread, which calls the function **Cmd_WriteStart**. This function checks the *State* parameter of the *IWriteInfo* structure, and only proceeds with the upload if it is in the *IWrite_Idle* state. If it is not in that state, it means there is already an upload process currently underway that either needs to be completed or terminated by the operator or an error before another upload process can begin. If it is confirmed that the state is in *IWrite_Idle*, the function updates the *FileName*, *Offset*, and *Type* parameters based on the information provided by the custom write command via NEMO Control, and sends a START event. This updates *State* to *IWrite_Ping* and calls the **SM_Ping** function.

The **SM_Ping** function creates a PING command packet, sends it to the payload, and sets a 3-second alarm: *TimeoutAlarm* with the *TimeoutCmd* set to PING. If the NRD Thread does not receive a response for the PING from the payload within that time frame, the *TimeoutAlarm* is triggered, calling the **SM_WriteTimeout** function. If the number of times this command has been resent for this command is less than the maximum allowed as per the *Retries* parameter, this function sends a TIMEOUT event to execute **SM_Ping** again. Otherwise, this function sends an ABORT event, which causes the process to return to the *IWrite_Idle* state. The *Error* parameter is also updated to reflect why the process has been terminated: exceeded maximum retries.

When the NRD thread does receive a PING response from the payload, the **Process_Ping** function is called to handle the response packet. This function stops the *TimeoutAlarm* and sends a PING_ACK event, updating the *State* to *IWrite_GetInfo*, and calls the **SM_GetInfo** function.

In the **SM_GetInfo** function, the file with the name stored in *FileName* is opened for reading and the size of the file is extracted and stored in the *BytesRemaining* parameter. If the *Offset* specified by the operator is a number larger than 0, this is subtracted from *BytesRemaining*. If there is any error associated with retrieving the specified information, the *Error* parameter is updated accordingly, and a GETINFO_NACK event is sent, updating the *State* to *IWrite_Idle* and terminating the process. If the file is successfully opened and the number of bytes is successfully retrieved, the *BytesExecuted* parameter is initialized to 0 and a GETINFO_ACK event is sent, updating the *State* parameter to *IWrite_Writing* and calling the **SM_WriteNext** function.

The **SM_WriteNext** function executes in a loop manner with the **Process_Write** function; this is where the actual upload process occurs. Using relevant parameter values: *BytesRemaining*, *Offset*, and *Type*, the **SM_WriteNext** function creates an IMAGE_WRITE packet and updates the *BytesExectued* parameter to reflect the bytes of data that will be written using that packet. The function sends the packet to the payload and sets a 3-second alarm: *TimeoutAlarm* with the *TimeoutCmd* set to IMAGE_WRITE. Similar to the PING process, if the NRD Thread does not receive a response to the IMAGE_WRITE command within the 3 seconds, the *TimeoutAlarm* is triggered and the **SM_WriteTimeout** function is called to either resend the command or terminate the process. Unlike the timeout triggered by the PING alarm, if the timeout is triggered during the image upload process and the maximum number of resends for the command has exceeded, the file, which was opened in **SM_GetInfo**, is closed before terminating the process.

When the NRD Thread does receive a response to an IMAGE_WRITE command, the **Process_Write** function is called to handle the response packet. If the IMAGE_WRITE command was not successfully executed, the payload will have sent a NACK packet; in this case, the **Process_Write** function closes the file in the file system, updates the *State* to *IWrite_Idle*, updates the *Error* parameter accordingly, and sends a WRITE_NACK event, terminating the process. If the command was successful, the payload will have sent an ACK; in this case, the **Process_Write** function updates *BytesRemaining* and *Offset* parameters based on the *BytesExecuted* and sends a WRITE_ACK event, calling the **SM_WriteNext** function to create and send the next appropriate IMAGE_WRITE packet. This process repeats between the **SM_WriteNext** and **Process_Write** functions until the *BytesRemaining* parameter reaches 0, in which case, the **SM_WriteNext** function updates the *State* to *IWrite_Idle* and sends a COMPLETE event, calling the **SM_WriteComplete** function.

The **SM_WriteComplete** function closes the file in the file system and resets all parameters in the *IWriteInfo* structure.

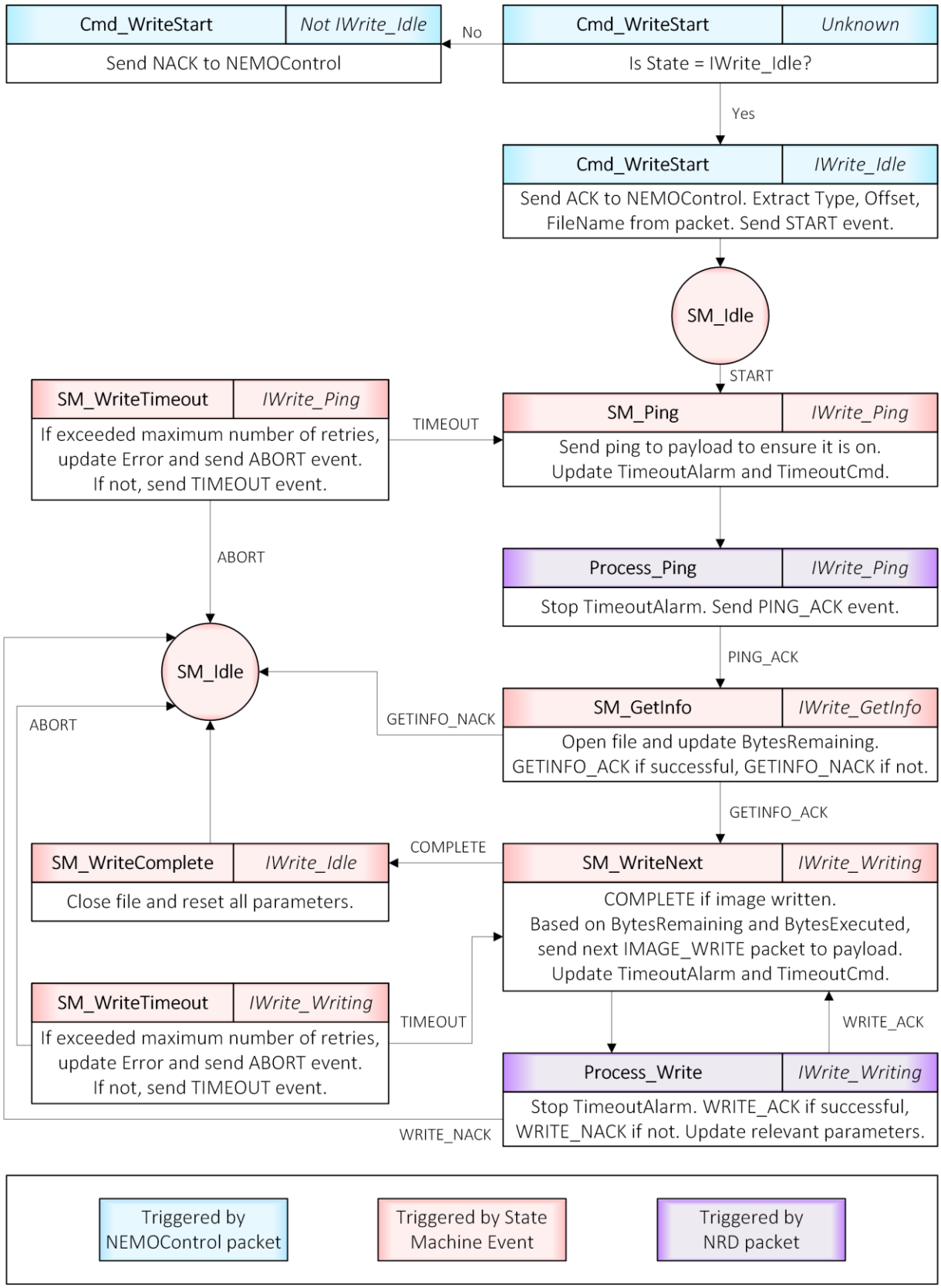

Figure 24: Flow of Image Upload via POBC – Write Command (NRD)

The image download process is initiated via the custom read command implemented in the 'Image Read' section location in the 'Image Control tab of the NRD interface in NEMO Control, shown in **Figure 21**.

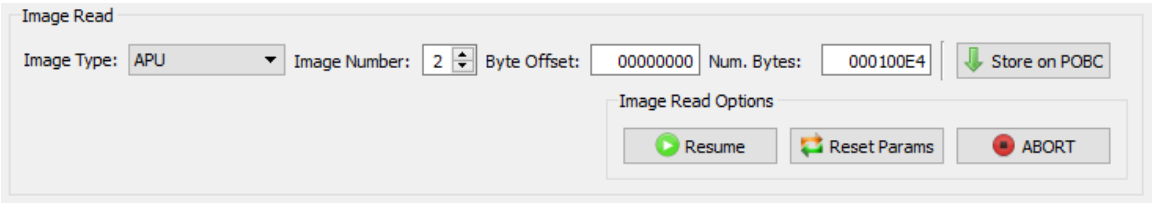

Figure 25: Image Download via POBC – Read Command (NRD Interface)

Figure 25 provides a closer look at the read command, where the operator is required to specify:

- image type and number that they wish to download,
- the address of the payload's flash memory that they wish to begin the download from, and
- the number of bytes of the image they wish to download.

Once all the required fields are filled, the 'Store on POBC' button is pressed to begin the download process as laid out in **Figure 26**. This button sends a custom command to the NRD thread, which calls the function **Cmd_ReadStart**. This function checks the *State* parameter of the *IReadInfo* structure, and only proceeds with the download if it is in the *IRead_Idle* state. If it is not in that state, it means there is already a download process currently underway that either needs to be completed or terminated by the operator or an error before another download process can begin. If it is confirmed that the state is in *IRead_Idle*, the function updates the *Offset*, *BytesRemaining*, and *Type* parameters based on the information provided by the custom read command via NEMO Control, then sends a START event. This updates *State* to *IRead_Ping* and calls the **SM_Ping** function. This function is executed identically as in the upload process, but the **SM_CreateFile** function is called after the PING_ACK event is sent.

In the **SM_CreateFile** function, a file to store image data into is created and opened for writing in the POBC file system. If, for some reason, the file could not be created or could not be opened for writing, a CREATEFILE_NACK event is sent, which updates the *State* to *IRead_Idle* and terminates the process. If the file is successfully created and opened, *FileName* is updated with the name and *BytesExecuted* is initialized to 0. After that, a CREATEFILE_ACK event is sent, updating the *State* parameter to *IRead_Reading* and calling the **SM_ReadNext** function.

The **SM_ReadNext** function executes in a loop manner with the **Process_Read** function; this is where the actual download process occurs. Using relevant parameter values: *BytesRemaining*, *Offset*, and *Type*, the **SM_ReadNext** function creates an IMAGE_READ packet and updates the *BytesExecuted* parameter to reflect the bytes of data that will be requested for reading using that packet. The function sends the packet to the payload and sets a 3-second alarm: *TimeoutAlarm* with the *TimeoutCmd* set to IMAGE_READ. In the case where no response is received for that command within the alarm time, the *TimeoutAlarm* triggers the **SM_ReadTimeout** function, which executes in the same manner as described in the image upload process.

When the NRD thread does receive a response to an IMAGE_READ command, the **Process_Read** function is called to handle the response packet. If the IMAGE_READ command was not successfully executed, the payload will have sent a NACK packet; in this case, the **Process_Read** function closes the file that the image data is being stored to in the process, updates the *State* to *IRead_Idle*, updates the *Error* parameter accordingly, and sends a READ_NACK event, terminating the process. If the command was successful, the payload will have sent an ACK with the bytes of image data as requested in the command; in this case, the **Process_Read** function ensures the received data length is equal to the *BytesExectued* parameter value, and updates *BytesRemaining* and *Offset* parameters based on the *BytesExecuted*. The received image data is appended to the file created in **SM_CreateFile** and a READ_ACK event is sent, calling the **SM_ReadNext** function to create and send the next appropriate IMAGE_READ packet. This process repeats between the **SM_ReadNext** and **Process_Read** functions until the *BytesRemaining* parameter reaches 0, in which case, the **SM_ReadNext** function updates the *State* to *IReadIdle* and sends a COMPLETE event, calling the **SM_ReadComplete** function.

The **SM_ReadComplete** function closes the file in the file system and resets all parameters. Once the download process is complete, the file (in this example, 'IMAGE_APU_02') where the data was stored to can be found on the POBC file system, as shown in **Figure 22**, and downloaded during a pass via QMTP.

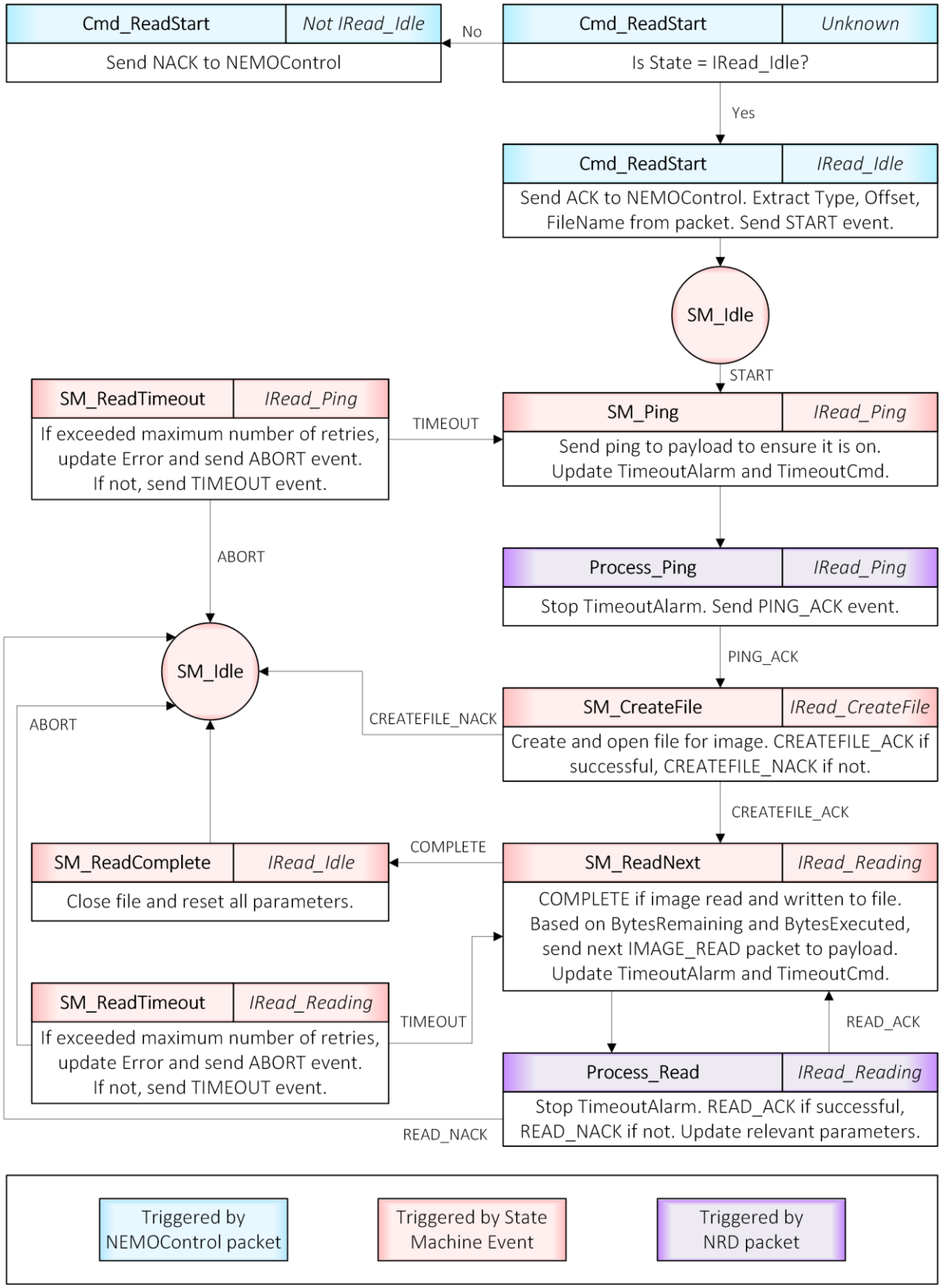

Figure 26: Image Download via POBC Process – Read Command (NRD)

6.6.1.3 Resume Command Implementation

In the case where an upload or download is terminated before completion, either due to an error or an abort, the operator has the option to resume the process from the point where it was terminated. As shown in **Figure 27** and **Figure 28**, resumes are identical to the overall upload and download processes, respectively, except the first functions triggered by NEMO Control command and the **SM_GetInfo** and **CreateFile** functions, respectively.

When the write/read 'Resume' button is pressed, this sends a custom command to the NRD thread to call the **Cmd_WriteResume**/**Cmd_ReadResume** function. This function not only ensures that the current *State* is set to *IWrite_Idle*/*IRead_Idle*, but that the *IWriteInfo*/*IReadInfo* structure is populated (e.g. *BytesRemaining* > 0). If the *State* is not *IWrite_Idle*/*IRead_Idle*, this indicates that the upload/download process is currently executing. If the structure parameters, specifically *Offset*, *FileName*, and *BytesRemaining*, are not populated, this indicates that there is nothing to resume. Once it is determined that a resume can occur, *Resuming* is set to true to indicate that the current process is a continuation, and the function sends a START event, which updates *State* to *IWrite_Ping*/*IRead_Ping* and executes the **SM_Ping** function as described in **Section 6.6.1.2**.

While all other functions proceeding **Cmd_WriteResume**/**Cmd_ReadResume** execute as previously described, the other difference exists in the execution of the **SM_GetInfo**/**SM_CreateFile** function. When resuming an upload/download, the parameter *BytesRemaining* should already contain a value from when the process was terminated. In the case of resuming an upload, the **SM_GetInfo** function merely opens the file as per *FileName*, and sends a GETINFO_ACK event if the file is opened successfully or a GETINFO_NACK event otherwise. In the case of resuming a download, the **SM_CreateFile** function opens the previously created file, *FileName*, for appending, and sends a CREATEFILE_ACK event if the file is opened successfully or a CREATEFILE_NACK event otherwise. All functions proceeding step execute as described in **Section 6.6.1.2**.

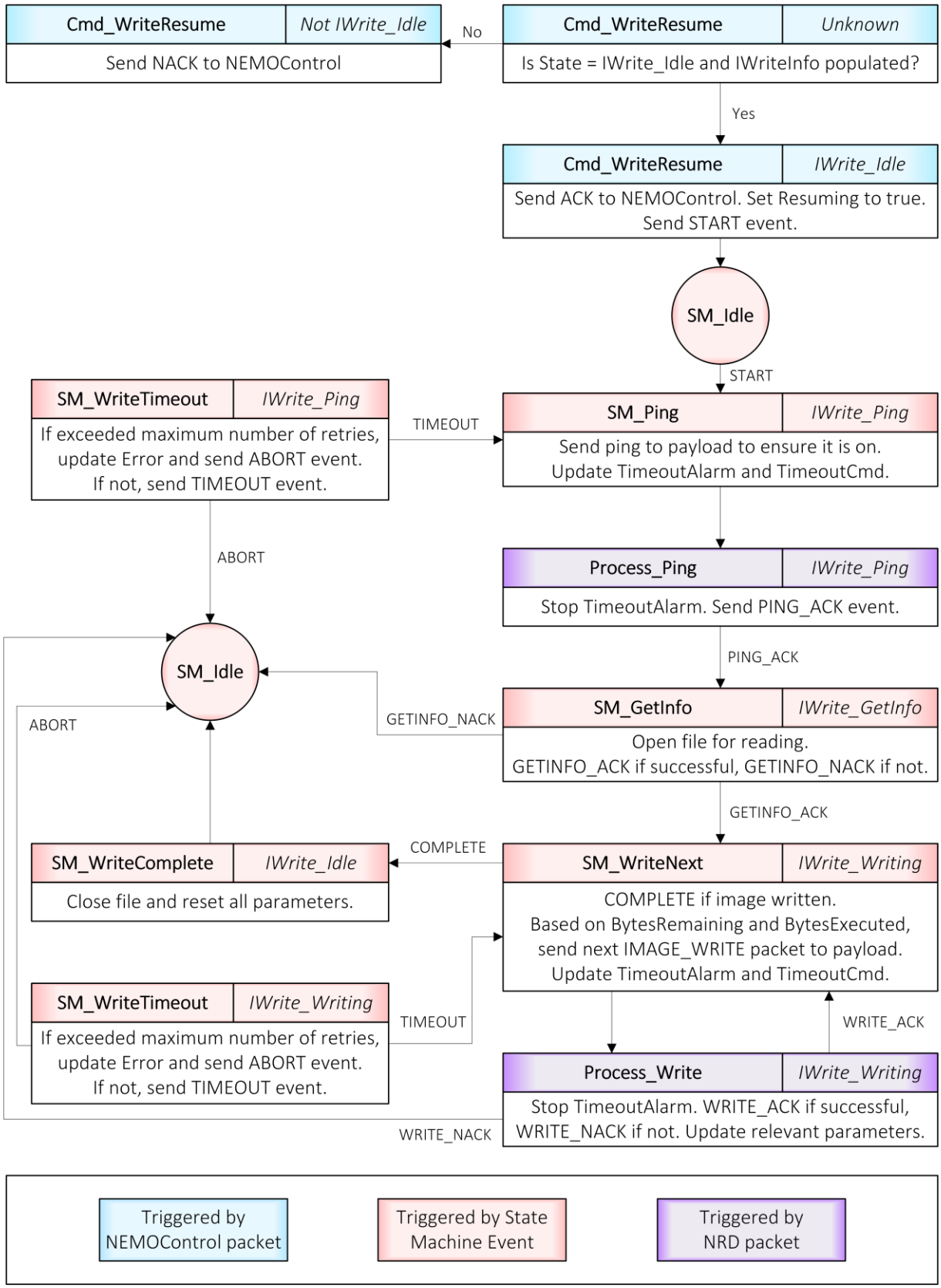

Figure 27: Image Upload via POBC Process – Resume Command (NRD)

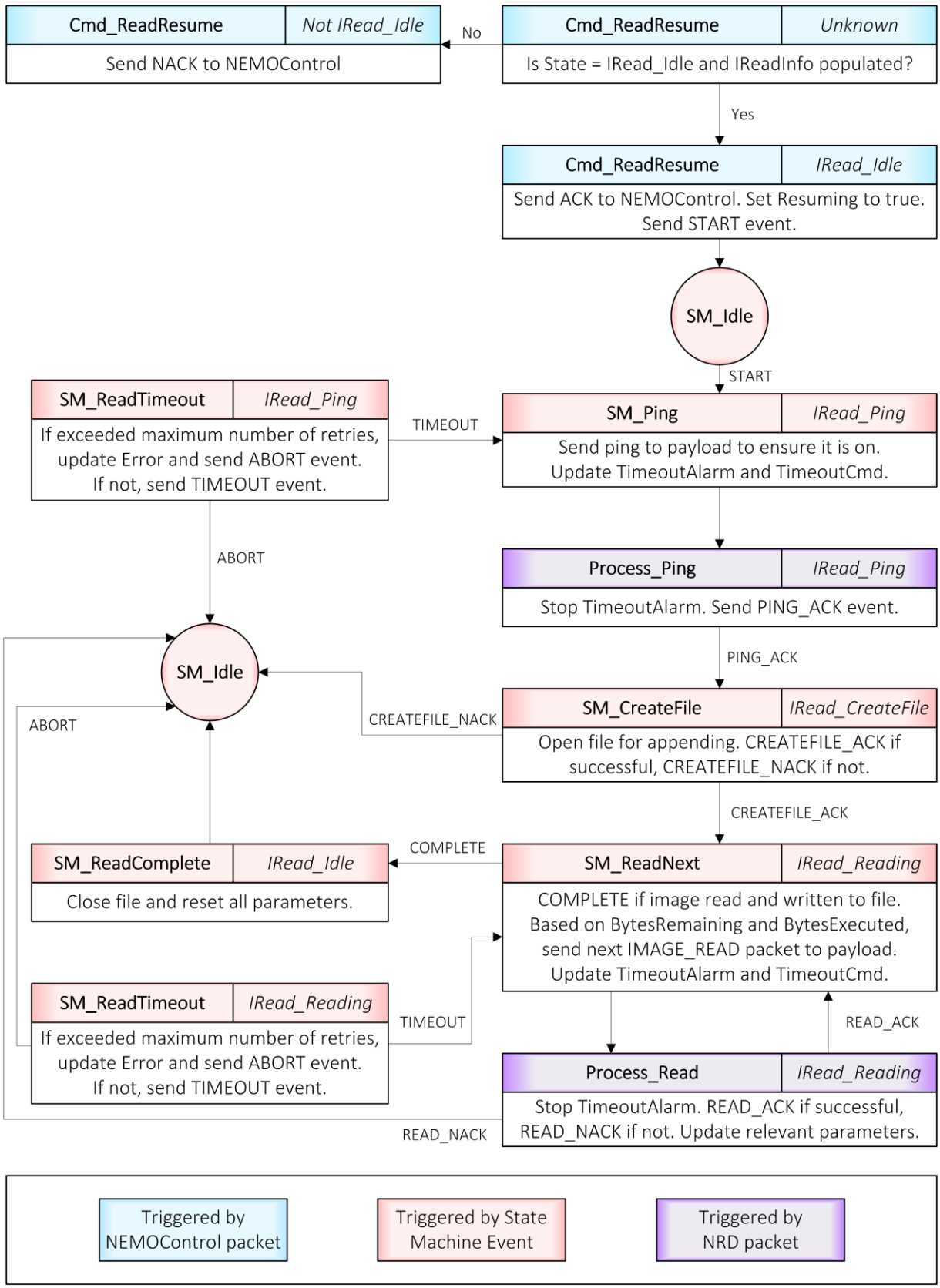

Figure 28: Image Download via POBC Process – Resume Command (NRD)

6.6.1.4 Abort Command Implementation

The write/read 'ABORT' button can be pressed anytime during an upload/download to terminate the process. Only the abort process for an upload is illustrated – in **Figure 29** – as the overall flow is similar for aborting a download. The 'ABORT' button sends a custom command to the NRD Thread to call the function **Cmd_WriteAbort**/**Cmd_ReadAbort**. This function checks the current *State* and only executes the termination if the *State* is not *IWrite_Idle*/*IRead_Idle*. If the *State* is in *IWrite_Idle*/*IRead_Idle*, this indicates that there is no upload/download process to abort. Once it has determined that the *State* is not idle, the function checks if the file associated with the image in the file system is open, closes it if it is, and sends an ABORT event. The ABORT event causes any state other than *IWrite_Idle*/*IRead_Idle* to return to the idle state, thereby terminating any other running function. In the case of an abort, all parameters, at the time the abort command was executed, are saved in the respective structure. This provides the operator the ability to resume the upload/download from when the process was terminated as described in **Section 6.6.1.3**.

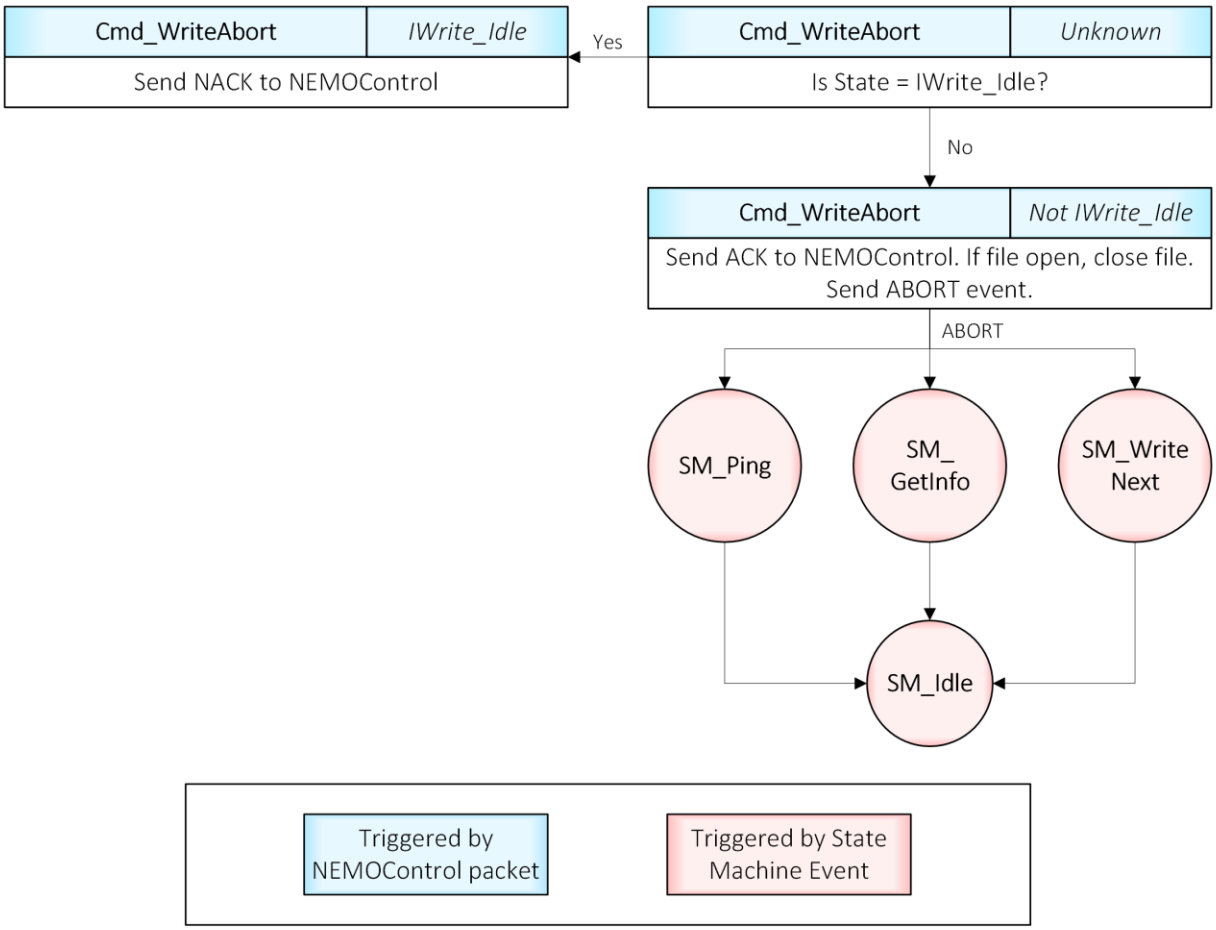

Figure 29: Image Upload Process via POBC – Abort Command (NRD)

6.6.1.5 Reset Command Implementation

The operator also has the option to reset the *IWriteInfo*/*IReadInfo* structure (i.e. clear all parameter values). Only the reset of *IWriteInfo* is illustrated – in **Figure 30** – as the overall flow is similar for resetting *IReadInfo*. When the write/read 'Reset Params' button is pressed, this sends a custom command to the NRD Thread to call the function **Cmd_WriteReset**/**Cmd_ReadReset**. This function ensures the *State* is *IWriteIdle*/*IReadIdle* before proceeding and simply resets the structure. This command is only valid when the *State* is in the idle state when no upload/download process has started, an upload/download process was completed, or an upload/download process was terminated either by error or an abort. Although valid in the idle state, as the *IWriteInfo*/*IReadInfo* structure is naturally reset upon initialization as well as after completion, this reset command is meant to be used after a termination when the process is stopped in the middle of an upload/download and the respective structure is populated. In the case where the operator wants to prevent a terminated process from being resumed, this reset command can be used.

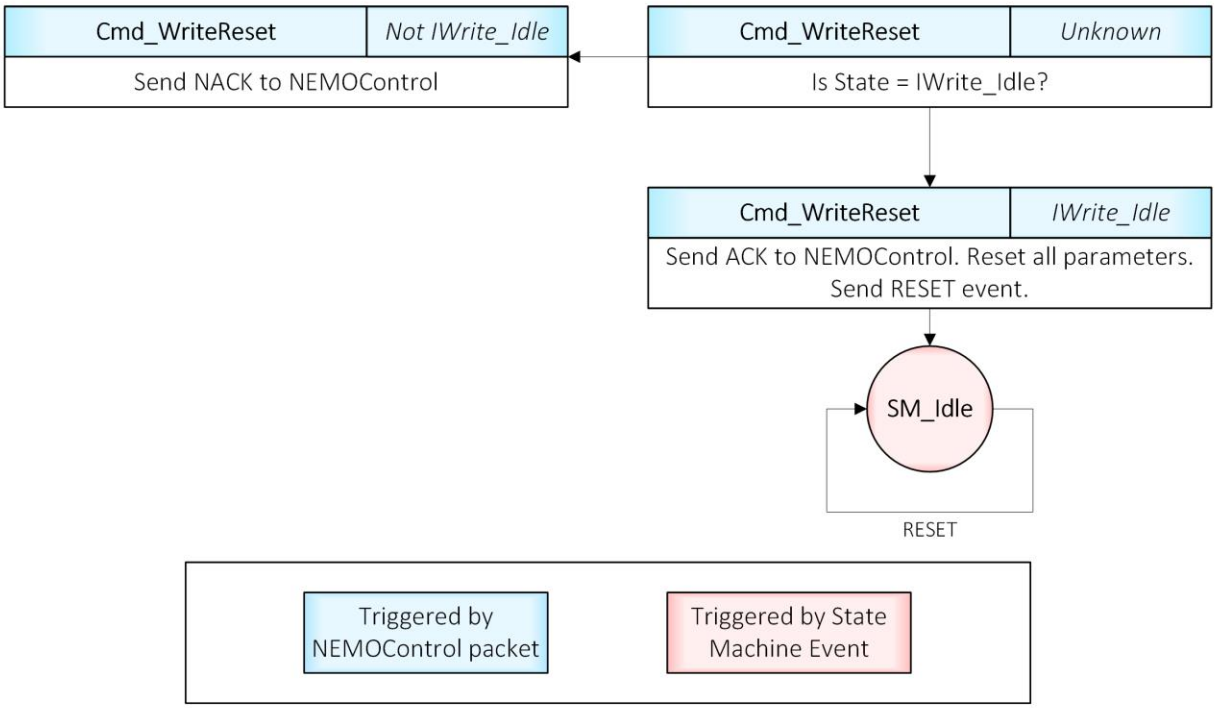

Figure 30: Image Upload Process via POBC – Reset Command (NRD)

6.6.2 Payload Data Handling

As mentioned in **Section 1.2**, the mission objective(s) are fulfilled by the payload subsystem(s), which highlights the importance of the data generated and output by the payload(s). As a result, one of the main responsibilities of the C&DH subsystem is to properly handle and manage all data output by the payload(s) over the mission lifespan.

Although this section discusses the author's work on NRD payload data handling, due to proprietary reasons, the specific types of NRD data products and their respective formats are not described in detail. Conventionally, each payload data type is assigned a unique command byte in the header of the NSP packet, which is used in the respective payload threads for differentiating between the different data products and handling them accordingly. The general format of a payload data packet can be found in **Table 20**. Typically, all data products output by the payload are destined to the POBC application thread for that payload, and the data product contents are found in the Data field of each packet.

Table 20: General NSP Format of Payload Data

PACKET FIELD	SIZE (BITS)	VALUE	COMMENT
DESTINATION ADDRESS	8	`0xNN`	Address of payload application thread.
SOURCE ADDRESS	8	`0xYY`	Address of the payload.
P/F BIT	1	0 or 1	For single packet replies, 1 for all packets. For sequence of packets, 1 for last and 0 for all others
B BIT	1	0 or 1	Value of command's B-bit.
A BIT	1	0 or 1	0 for non-acknowledgement. 1 for acknowledgement.
COMMAND	5	`0x00 – 0x1F`	Unique to payload and data type.
DATA	Variable	Varies	A-bit = 0: Error code is returned. A-bit = 1: Content of payload data
CRC	16	`0xNN`	Unique to each packet.

Much like the other work presented in this thesis, the author's contributions to the data handling portion of the NRD Thread is very significant as the overall code and logic can easily be implemented on other payload threads to handle the respective payload data on future missions.

6.6.2.1 Issue with Conventional Method of Handling Real-time Payload Data

NS3-CDH-R008 in **Table 7** states that the default baud rate for the NRD payload shall be 921.6 kbps. Not only is this the fastest baud rate option, but the NRD payload has a dedicated channel for its data output compared to the AIS payload which has one channel for both data and commands. This means the full 921.6 kbps on one NRD channel is purely reserved for payload data. As mentioned in **Section 6.6**, in order to comply with *NS3-SYS-R009* and *NS3-SYS-R010*, the thread must be capable of handling payload data, which includes storing them in files and forwarding them to ground. At a high-level, the approach used on previous missions handles each payload data packet individually. Although this method has been successful on previous missions, it is incapable of keeping up with the 921.6 kbps data rate imposed by the NRD payload, especially when both the NRD and AIS payloads output data simultaneously. Specifically, in cases where data is output at 921.6 kbps, the conventional method stores the received data at a slower rate than it receives them; therefore, as the volume of data grows, the larger the lag becomes. In order to accommodate the faster than usual data rate, the code for handling data needed to be optimized.

Similar to the debugging approach used for the NEMO-HD portion of this thesis, timing measurements were taken in order to pinpoint which section of the data handling code was consuming the largest amount of time. After various tests, it was determined that the most time-consuming portion was storing packet data into files in the POBC file system. The function which handles the writing of data to a file involves multiple steps beyond the writing aspect, including locating the file, checking various aspects of the file system, and other file system related tasks. In the conventional method of storing payload data, this function is called for every packet received; therefore, the overhead involved in the storing was consuming a significant amount of time.

The method explored and taken to overcome this issue was to modify the code to store multiple packet data in an array first, then write the bulk of data stored in said array to the file once the array is full. Once timing measurements were gathered, it became evident that the total time taken to store a fixed number of bytes into a file on the file system decreases significantly as the number of bytes per write increases. This relation can be visualized in **Figure 31** which plots the estimated worst-case data handle rate of writing a total of 50 MB of data against varying number of bytes per write. The data used for this graph is captured in **Table 21**, where it was conservatively approximated that the storing of data takes up a tenth of the overall data handling time.

Table 21: Estimated Worst-Case Data Handling Rates using Various Bytes per Write

# OF BYTES PER WRITE	TIME TAKEN TO WRITE (MS)	WRITE DATA RATE (KBPS)	ESTIMATED SLOWEST DATA HANDLING RATE (KBPS)
50	247094	1618.817	161.882
100	138895	2879.873	287.987
200	83634	4782.744	478.274
500	51471	7771.366	777.137
800	42822	9340.993	934.099
1000	40429	9891.441	989.144
2000	34946	11446.231	1144.623
5000	31702	12617.500	1261.750
8000	30903	12943.727	1294.373
10000	30643	13053.552	1305.355

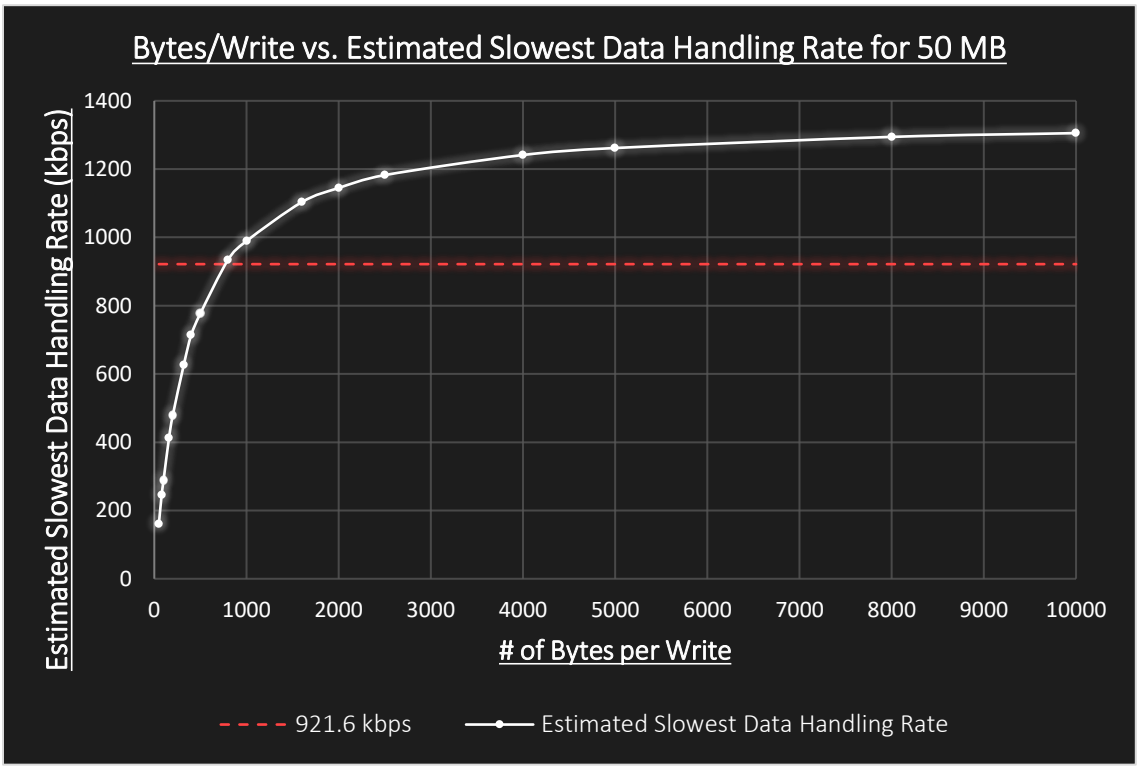

Figure 31: Graph of Bytes per Write vs. Estimated Slowest Data Handling Rate for 50 MB

As it can be seen from **Figure 31**, the data handling rate significantly grows as the number of bytes per write increases in the 50 – 1000 bytes per write range, but plateaus beyond the 2500 bytes per write point. Although a positive correlation between the two is evident, there is not an infinite amount of memory to allocate for the arrays and it is clear that after a certain point, the benefits become less significant; therefore, a reasonable array size was required to be selected.

6.6.2.2 Optimized Code for Handling Real-time Payload Data

The overall flow of the optimized method for handling payload data is laid out in **Figure 32**. The process begins when the payload outputs data destined to the respective payload thread on the POBC. Whenever a payload packet is received, the **Process_Command** function is called to handle it. This function checks the command byte to determine which data product type it is and forwards it to the appropriate **Handle_Data_Products** function.

The handler function then sends the packet to the respective **Check_Data_Packet** function to check whether the packet meets the requirements for that specific data product. If it is not valid, it will update relevant metrics and discard the packet. If it is valid, it will return to the respective **Handle_Data_Products** function to properly handle the verified data packet.

The handler function checks if either or both the 'Forward' and/or 'Store' parameter for that data type is enabled. These parameters can be enabled/disabled by the operator via the 'General' tab of the NRD interface, shown in **Figure 20**. If the 'Forward' parameter is enabled, the packet is sent to a **Forward_Data_Products** function to send the packet to ground over radio. If the 'Store' parameter is enabled, the handler function checks if there is enough space in the array – for that data type – to store the current packet data. If there is enough space, the data contents in the packet are extracted, appended to the array, and the array size is updated. If there is insufficient space, the contents in the array must be written to the designated file and the array must be reset before storing the packet data.

This is handled by calling the **Write_DataBlock** function. This function first checks whether a file for the specific data product exists in the file system. If there is no such file, it calls the **New_Data_File** function to create and open a file. If there is a file or once a file has been successfully created, the **Write_DataBlock** function ensures that the file has not reached the maximum size allowed, as per 'Max MB Per File' parameter in the 'General' tab. If the file is under the maximum size allowed, the function writes all the array data to the file and appends the packet data to the reset array. If the file has reached the maximum size allowed, the **New_Data_File** function is called to close the current file for that data type and create a new file before the contents of the array can be written, after which, the array is reset and the packet data are appended to the reset array.

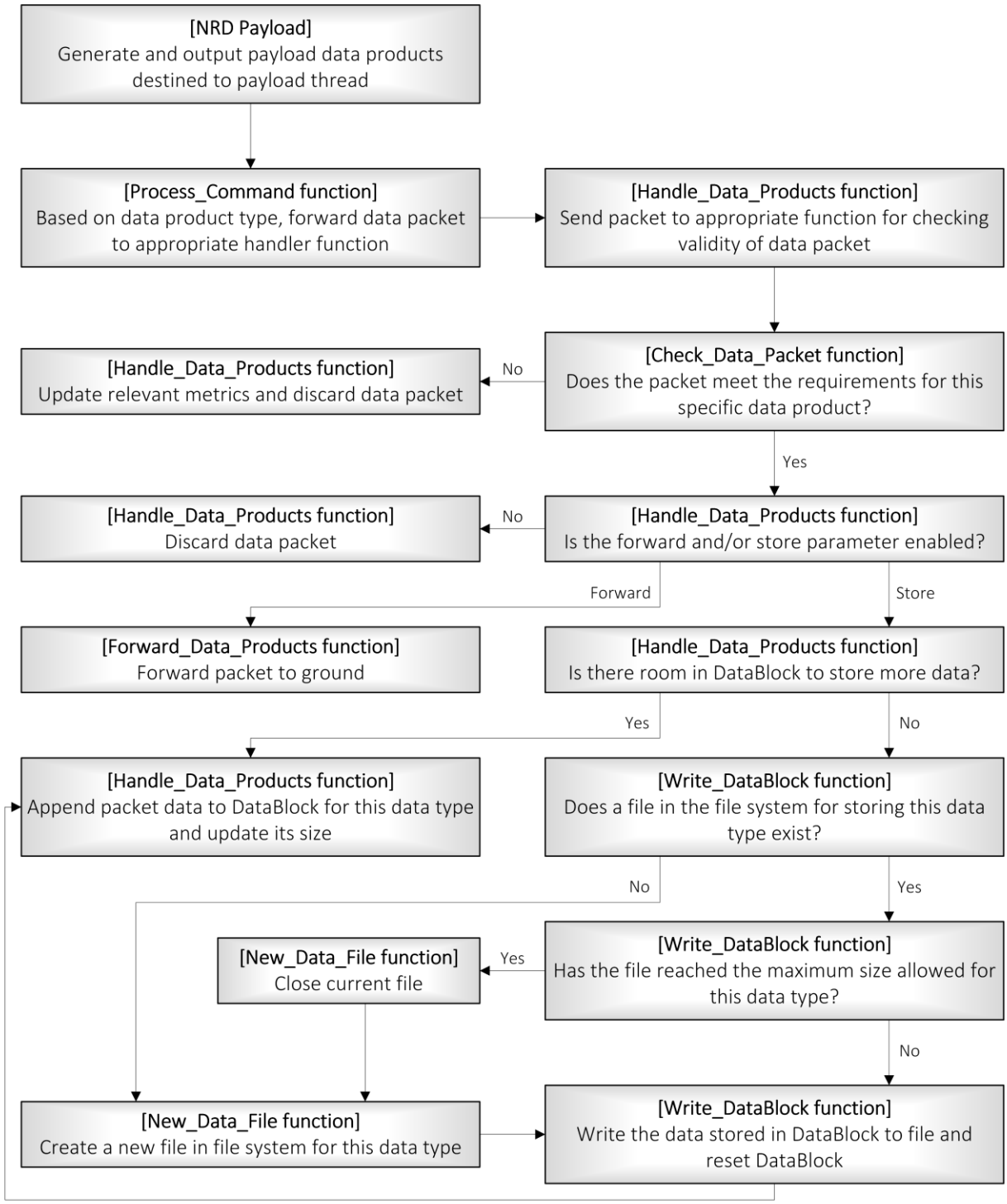

Figure 32: Payload Data Handling Flow (NRD)

The red line in the graph shown in **Figure 31** represents the default baud rate of the NRD payload data channel. Anything above 800 bytes per write seems to meet the requirement from a conservative view (i.e. conservatively assuming that the storing portion takes a tenth of the time

of the overall data handling process), and anything above 2500 does not seem to provide much benefit when taking into account the memory allocation. In order to finalize an array size, timing measurements were gathered using the method laid out in **Figure 32** to handle and store simulated NRD data with varying array sizes between 1000 – 3000. The results are captured in **Table 22**.

Table 22: NRD Payload Data Handling Rates using Various Array Sizes

	ARRAY SIZE (# OF BYTES PER WRITE)				
	1000	**1500**	**2000**	**2500**	**3000**
STORING 20,000 PACKETS IN ONE FILE (960,000 BYTES)					
TIME TAKEN TO HANDLE DATA (MS)	4146	4020	3914	3938	3912
DATA HANDLING RATE (KBPS)	1852.4	1910.5	1962.2	1950.2	1963.2
STORING 200,000 PACKETS IN ONE FILE (9,600,000 BYTES)					
TIME TAKEN TO HANDLE DATA (MS)	40923	40288	39377	39018	39072
DATA HANDLING RATE (KBPS)	1876.7	1906.3	1950.4	1968.3	1965.6
STORING 200,000 PACKETS IN 10 FILES (9,600,000 BYTES)					
TIME TAKEN TO HANDLE DATA (MS)	42615	41465	40757	40451	40525
DATA HANDLING RATE (KBPS)	1802.2	1852.2	1884.3	1898.6	1895.1

From analyzing the data, it can be observed that all the array sizes considered in this particular test allow the payload thread to be capable of handling data products received at the fastest rate. It is also important to note that the benefits in terms of increase in data handling rates are the most noticeable in the 1500 and 2000 bytes per write range. As observed in **Figure 31**, once the array exceeds 2500 bytes, the rate seems to plateau. This is confirmed in the simulated tests as per **Table 22**, where the rates seem to increase less as the array size approaches 2500 bytes. Accounting for the memory allocation and the timing measurements provided, the array size was determined to be 2000 bytes.

As the original method of storing packet by packet is not fast enough to keep up with the data output rate from the NRD, the tests conducted in **Table 22** could not be performed using that method. Instead, timing measurements were gathered for a smaller number of packets totaling up to 4800 bytes, which indicated a data handling rate of around 500 kbps. Therefore, compared to the original method of storing packet by packet, where the Data field of the packets can range from 48 to 256 bytes for the NRD payload, the method that the author has presented can provide over a 277% increase in payload data handling rates.

Chapter 7

Conclusion

This chapter summarizes the author's contributions to the command and data handling subsystem on two different microsatellite missions – NEMO-HD and NorSat-3 – that can be implemented and extended for use on future SFL missions, as well as the author's recommendations resulting from their work.

7.1 NEMO-HD

The main goal for the NEMO-HD portion of this thesis was to configure the existing software programs, namely PMTP, to achieve effective data downlink rates of at least 24 Mbps for the two scenarios of interest in the worst-case scenario using an uplink rate of 3.5 kbps. As presented in the NEMO-HD portion of this thesis, the author was successfully able to implement an upgraded algorithm and configure the various software programs to accomplish this goal. Specifically, the author's contributions resulted in effective downlink data rate increases by up to 194%.

This work can easily be extended on future missions with similar data requirements as NEMO-HD, in other words, data-intensive missions where the downlink rate is significantly faster than the uplink rate. The performance of the current PMTP software shows that the problem, presented in this thesis, can be solved via software.

A recommendation that the author has after her experience on this mission is to implement the flow control capability on the TNC, now known as SFIC. This will eliminate the need for the middle interface, in this case the MuxMaster program, to regulate the packet dispatch rate to ensure that the SFIC does not receive them too quickly. Realistically, the dispatch rate on the Mux side needs to be slightly slower than the full dispatch rate allowed by the SFIC, which means the packets are not actually uplinked at the fastest rate capable. If the flow control capability were to be implemented on the SFIC side, the packets could be allowed to queue up in the SFIC to be dispatched as soon as the previous packet is sent.

7.2 NorSat-3

The main goal for the NorSat-3 portion of this thesis was to design, develop, and test the C&DH subsystem for the satellite. The portion of the author's work on NorSat-3 presented in this thesis are contributions that can be used and implemented on future missions, which included development of the new SIB, application thread and interface for the new high-speed S-band receiver telemetry, and application thread and interface for the new NRD payload, with focus on general payload image upload/download processes and payload data handling methods.

The new SIB design presented in this thesis has been successfully manufactured and tested for use on NorSat-3, and its adaptable and versatile design will allow it to be easily implemented on future missions as well. The application thread and corresponding NEMO Control interface for requesting and retrieving telemetry from the new high-speed S-band receiver has been developed and are configurable for use on any future mission utilizing receivers with similar telemetry systems. Another aspect presented was the implementation of uploading/downloading payload images via the POBC, which has been proven to be successful and extendable across other payloads (e.g. the AIS receiver). Lastly, the author's work on the data processing portion in payload threads, which can also easily be adopted on future SFL missions, presents an optimized method in preference to the previously used methods at SFL for handling data products output at the fastest rate of 921.6 kbps. Specifically, compared to the original method of storing payload data packet by packet, the author's upgraded method of storing array by array results in payload data storage rate increases by over 270%

Bibliography

[1] Union of Concernd Scientists, "UCS Satellite Database," Union of Concernd Scientists, 7 November 2017. [Online]. Available: https://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database#.W1N8QtIzpQA. [Accessed 21 July 2018].

[2] Canadian Space Agency, "Satellites in our everyday lives," Canadian Space Agency, 8 June 2018. [Online]. Available: http://www.asc-csa.gc.ca/eng/satellites/everyday-lives/default.asp. [Accessed 14 July 2018].

[3] G. Bonin, "Microspace and Human Spaceflight," The Space Review, 10 August 2009. [Online]. Available: http://www.thespacereview.com/article/1441/1. [Accessed 14 July 2018].

[4] R. E. Zee, "Microsatellite Science and Technology Center: Canada's Centre for Microspace Innovation," 2010. [Online]. Available: http://utias-sfl.net/wp-content/uploads/52_Zee_ASTRO2010.pdf. [Accessed 21 July 2018].

[5] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "About SFL: Mission Statement," UTIAS-SFL, 25 January 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=426. [Accessed 14 July 2018].

[6] R. E. Zee, "Spacecraft Design and the Microspace Philosophy (Lecture Notes)," UTIAS-SFL, Toronto, 2017.

[7] P. Stibrany and K. Carroll, "The Microsat Way in Canada," in *11th CASI Canadian Astronautics Conference*, Toronto, 2000.

[8] J. Bouwmeester, "Spacecraft Technology (Lecture Notes)," Delft University of Technology, [Online]. Available: https://ocw.tudelft.nl/wp-content/uploads/1.0-Command-and-Data-Handling-Lecture-Notes.pdf. [Accessed 14 July 2018].

[9] B. R. Elbert, The Satellite Communication Ground Segment and Earth Station Handbook, Norwood: Artech House Publishers, 2005.

[10] J. H. (. Choi, "Ground Segment Software Design and Development for Nanosatellite Space Missions," UTIAS-SFL, Toronto, 2013.

[11] European Space Agency, "Telemetry & Telecommand," European Space Agency, [Online]. Available: http://m.esa.int/Our_Activities/Space_Engineering_Technology/Onboard_Computer_and_Data_Handling/Telemetry_Telecommand. [Accessed 14 July 2018].

[12] D. D. Kekez, "Nanosatellite Protocol (NSP) Versions 3 and 4," UTIAS-SFL, Toronto, 2014.

[13] M. Mancini and D. Kekez, "NorSat-3 Command and Data Handling Subsystem Design Document," UTIAS-SFL, Toronto, 2018.

[14] ELPROCUS, "Overview on Electronic Communication Protocols," ELPROCUS, [Online]. Available: https://www.elprocus.com/communication-protocols/. [Accessed 12 December 2018].

[15] M. Dwyer, "Embedded Software Design for the Canadian Advanced Nanospace eXperiment Generic Nanosatellite Bus," UTIAS-SFL, Toronto, 2009.

[16] J. Lifshits, "NEMO-HD Software Architecture," UTIAS-SFL, Toronto, 2013.

[17] I. Poole, "BER Bit Error Rate Tutorial and Definition," Radio-Electronics, [Online]. Available: https://www.radio-electronics.com/info/rf-technology-design/ber/bit-error-rate-tutorial-definition.php. [Accessed 22 July 2018].

[18] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Arianespace to Launch Slovenian NEMO-HD Microsatellite," UTIAS-SFL, 3 December 2018. [Online]. Available: https://www.utias-sfl.net/?p=3094. [Accessed 21 December 2018].

[19] N. Orr, "SFL-NHD-SYS-R001 - System Requirements V1.3," UTIAS-SFL, Toronto, 2012.

[20] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "SPACE-SI Awards NEMO-HD Contract to SFL," Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), 22 December 2011. [Online]. Available: https://www.utias-sfl.net/?p=990. [Accessed 19 July 2018].

[21] N. Orr, "NEMO-HD System Design Description," UTIAS-SFL, Toronto, 2013.

[22] N. Orr and J. Lifshits, "NEMO-HD Instrument Electronics Requirements," UTIAS-SFL, Toronto, 2012.

[23] F. M. Pranajaya, "NEMO-HD Instrument Requirements," UTIAS-SFL, Toronto, 2012.

[24] N. Orr, "NEMO-HD Communication Requirements," UTIAS-SFL, Toronto, 2012.

[25] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Microsatellites: NEMO-HD," UTIAS-SFL, 25 January 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=1233. [Accessed 19 July 2018].

[26] F. M. Pranajaya and I. Majid, "NEMO-HD Primary Instrument Design Document," UTIAS-SFL, Toronto, 2013.

[27] G. Rose, "Pan Sharpening," [Online]. Available: http://imstrat.ca/uploads/files/Brochures/PanSharpening.pdf. [Accessed 20 July 2018].

[28] J. Lifshits, "NEMO-HD Mission Operations Concept," UTIAS-SFL, Toronto, 2014.

[29] J. Lifshits, L. Stras and M. Fournier, "NEMO-HD Instrument Electronics Design Description v2.0," UTIAS-SFL, Toronto, 2013.

[30] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "NEMO-HD Optimal Downlink Packet Size," UTIAS-SFL, Toronto, 2015.

[31] UTIAS-SFL, "NEMO-HD Optimal Downlink Packet Size," UTIAS-SFL, Toronto, 2015.

[32] J. Harr, *Microsatellites for Maritime Surveillance (An update on the Norwegian Smallsat Program),* Norwegian Space Centre, 2018.

[33] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Nanosatellites: AISSAT-1," UTIAS-SFL, 25 January 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=214. [Accessed 19 November 2018].

[34] J. Harr, T. Jones, B. N. Andersen, T. Eriksen, A. N. Skauen, K. Svenes, E. V. Blindheim, I. Spydevold, A. Beattie, L. M. Bradbury, Kekez, D. Kekez, P. Mehradnia, R. E. Zee and F. Storesund, "Microsatellites for Maritime Surveillance - an update on the Norwegian Smallsat Program," in *International Astronautical Congress (IAC)*, Bremen, 2018.

[35] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Nanosatellites: AISSat-2," UTIAS-SFL, 25 Jannuary 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=1265. [Accessed 2 December 2018].

[36] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Nanosatellites: AISSat-3," UTIAS-SFL, 25 January 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=1268. [Accessed 24 Match 2019].

[37] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Microsatellites: NORSAT-1," UTIAS-SFL, 25 January 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=1260. [Accessed 2 December 2018].

[38] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "Microsatellites: NORSAT-2," UTIAS-SFL, 25 January 2014. [Online]. Available: https://www.utias-sfl.net/?page_id=2312. [Accessed 2 December 2018].

[39] B. Cotten, "NorSat-3 Systems Design Document," UTIAS-SFL, Toronto, 2017.

[40] Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), "NorSat-3 Ordered by Norwegian Space Centre, Satellite Under Construction at SFL," Space Flight Laboratory (University of Toronto Institute for Aerospace Studies), 10 January 2018. [Online]. Available: https://www.utias-sfl.net/?p=2730. [Accessed 20 July 2018].

[41] B. Cotten, "SFL-NS3-SYS-R001 - NorSat-3 Requirements and Verification Matrix," UTIAS-SFL, Toronto, 2018.

[42] International Maritime Organization (IMO), "AIS transponders," International Maritime Organization (IMO), [Online]. Available: http://www.imo.org/en/ourwork/safety/navigation/pages/ais.aspx. [Accessed 6 August 2018].

[43] E. V. Blindheim, "Customer Specification NorSat-3," Norwegian Space Centre (NSC) & Norwegian Defense Research Establishment (FFI), Kjeller, 2017.

[44] C. Mok, "Design and Implementation of the Flight Application Software and Nanosatellite Protocol for the CANX-2 Nanosatellite," UTIAS-SFL, Toronto, 2005.

[45] Qt, "Qt Designer Manual," Qt, [Online]. Available: http://doc.qt.io/archives/qt-4.8/designer-manual.html. [Accessed 11 December 2018].

[46] National Aeronautics and Space Administration, Earth Science Reference Handbook, Washington: Sterling Spangler, 2006.