University of tartu
Institute of Computer Science
Computer Science Curriculum

Teet Saar

Reliable File Transfer for the ESTCube-2 Nanosatellite On-Orbit

Bachelor's Thesis (9 EAP)

Supervisors:
Umesh Anilchandra Bhat, B.Tech
Boris Kudryashov, PhD
Vitaly Skachek, PhD

Tartu 2021

Reliable File Transfer for the ESTCube-2 Nanosatellite On-Orbit

Abstract:

ESTCube-2 is a student nanosatellite program, expected to launch in 2022. This bachelor's thesis focuses on the development of a reliable file transfer system for the ESTCube-2 Mission Control System for the upcoming mission. Functionality that allows automated and fail-safe file transfers is of great importance for the ESTCube-2 mission, as is the ability to downlink files from the satellite and uplink files from the ground station. The developed software solution analyzes, tracks, and synchronizes files and also deals with detection and resolution of different losses and errors that occur during data transmission between the satellite and the ground station. These requirements were fulfilled by the development of a file transfer service, accompanied by a visual interface incorporated into the front-end of the Mission Control System and a mock on-board Application Programming Interface for testing and development reference.

Keywords:

File transfer, Mission Control System, Data transmission, ESTCube-2

CERCS: P170 Computer science, numerical analysis, systems, control

Töökindel failiedastus orbiidil oleva ESTCube-2 nanosatelliidiga

Lühikokkuvõte:

ESTCube-2 on üliõpilaste poolt arendatav nanosatelliitprogramm, mis planeeritud startima 2022. aastal. Käesolev bakalaureusetöö keskendub ESTCube-2 missiooni juhtimissüsteemi jaoks töökindla failiedastussüsteemi väljatöötamisele, mis rakendatakse eelseisvale missioonile. Funktsionaalsus, mis võimaldab failide automaatset ja tõrkekindlat edastamist on ESTCube-2 jaoks väga suure tähtsusega, nagu ka failide satelliidilt alla linkimise ja maapealsest sidejaamast failide üles linkimise toimivus. Arendatud tarkvaralahendus analüüsib, jälgib ja sünkroniseerib faile ning tegeleb ka satelliidi ja maajaama vahel andmete edastamisel tekkivate kadude ja vigade tuvastamise ning lahendamisega. Tarkvarale esitatud nõuded täideti failiedastusteenuse ja pardal oleva rakenduse programmeerimisliidese arenduse ning satelliidimissiooni juhtimissüsteemi *front-end*'i integreeritud visuaalse liidese kaudu.

Võtmesõnad:

Failiedastus, missiooni juhtimissüsteem, andmeedastus, ESTCube-2

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Table of Contents

Abbreviations

API Application Programming Interface

AX.25 Amateur X.25 Communication Protocol

CMD Command

CRC Cyclic Redundancy Check

DST Destination

FTS File Transfer Service

GS Ground Station

HAL Hardware Abstraction Layer

ICP Internal Communication Protocol

LEN Length

LEO Low Earth Orbit

MCS Mission Control System

OBCS On-Board Computer System

PID Protocol Identifier

SRC Source

UUID Universally Unique Identifier

1. Introduction

ESTCube-2 is a nanosatellite under development by the Estonian Student Satellite Foundation[1] and students of the University of Tartu[2] (*Ehrpais et al., 2016: 1*). It's the successor to the ESTCube-1 mission, which had a mission lifespan from 2013 to 2015 (*ESTCube-1, 2020*). ESTCube-2 was originally scheduled to launch in 2018, but the launch of the satellite has been repeatedly postponed (*Nanosats, 2021: 23*). It is now expected to launch by 2022. One of the subsystems of the ground support structure under active development is the Mission Control System (MCS), which is a combination of hardware and software that serves as a post-launch support system (*ESTCube-2, 2020*). The purpose of the MCS is to provide steady communication between the ground station and the satellite in space by sending commands and archiving the data received from the satellite (*Liiv, 2017: 7*). The ESTCube-1 MCS provided spartan support for uplinking or downlinking files with the process requiring substantial manual user effort and supervision.

This bachelor's thesis delivers software that is integrated into the ESTCube-2 MCS. In order to make the linking processes more convenient and seamless for the current mission, additional services have been developed, which provide features such as splitting files into packets, stitching packets together into files, and managing errors/losses during data transmission. The software analyzes, tracks, and synchronizes files between a spacecraft on-orbit and an on-ground station. It improves over the ESTCube-1 MCS by automating the detection and resolution of different losses and errors that have occurred during data transmission. The finished software manages the following requirements in the subsequent order of implementation, which also act as the objectives of this thesis:

- Automatically resend packets lost or corrupted during transmission;
- Store information about each file and the state of its transmission without errors;
- Split files into packets of pre-determined sizes that can be transmitted;
- Automatically compile the file(s) from the received packets;
- Provide an API to uplink files just a command and the file;
- Provide a visual interface to view the detailed information of all files being uplinked and downlinked.

[1] https://www.estcube.eu/en
[2] https://www.ut.ee/en

The value proposition of this thesis is to create software that helps automate and facilitate the daily work of the ESTCube-2 mission control team. An important part of the work is to also elaborate on the theoretical side of file synchronization and the operation of the created software solution. The thesis is structured to firstly give an overview of the background of the ESTCube-2 mission and its MCS, explaining the basic ideas of data transmission, use cases and requirements. This overview is followed by an explanation of the design and implementation. The terms *spacecraft* and *satellite* have been used interchangeably and mean the same thing. The bachelor's thesis concludes by measuring the success of the developed software and analysing its capabilities. The software created within the framework of this bachelor's thesis would be primarily intended for the transmission of large-scale data packets, which include data and photos collected during the mission. The software developed for this bachelor's thesis would be ready just in time to be deployed for the current, and all future missions.

2. Background

2.1 CubeSats

Since the genesis of CubeSat development in 1999, they have become exponentially popular and widespread (*Li et al. 2013: 1*). They are now the defacto standard for space research by small and low-budget organisations such as universities (*Almansoori et al., 2018: 52*). This has been largely driven by the miniaturization of electronics and development of better manufacturing processes.

Although seemingly having lesser capabilities than their larger and expensive alternatives; some of the bigger restrictions of CubeSats have to do with the communication aspect; its small size and surface area does not allow accommodating powerful antennae and the communication window with the ground station is usually short (*Almansoori et al. 2018*).

2.2 ESTCube-2

By design, ESTCube-2 is an experimental 3-Unit CubeSatellite, meaning that it consists of three identical cubes connected together (*Mabrouk, 2017; Dalbins, 2017: 1*). The spacecraft is intended for low Earth orbit (LEO), meaning the distance of ESTCube-2 from Earth would be in the range of 500-700 kilometres *(Ehrpais et al., 2016: 2*). The ESTCube-2 mission is set to demonstrate different space-related technologies applicable for future missions including missions to the moon, and deep space (*Iakubivskyi et al., 2016*). The list of ESTCube-2 payloads is as follows:

- An electric solar sail (*e-sail*) by the Finnish Meteorological Institute[3];
- Plasma inhibition;
- Earth observation cameras developed in-house;
- A high-speed C-band downlink system developed in-house;
- A 1-unit miniature satellite bus developed in-house;
- A thin protective cover film by Captain Corrosion OÜ[4].

2.3 Mission Control System

Satellite missions in general consist of the space segment and the ground segment working together seamlessly (*Dessoy et al., 2017: 48*). The space segment consists of the spacecraft

[3] https://en.ilmatieteenlaitos.fi
[4] https://captaincorrosion.com

on-orbit comprising multiple subsystems. The ground segment consists of the ground station hardware providing a communication link to the spacecraft by transmitting and receiving radio waves and mission control software, which handles the telemetry commands, and data exchange.

2.4 Data Transmission

Communication system between satellite and ground station operates through two channels with different speeds - a slower connection channel, which is based on the telegraph signal and periodically provides simpler information about the satellite, and a faster connection channel, which is intended for large amounts of data (*ESTCube-1, 2020*). LEO satellites such as ESTCube-2 can receive and return information almost 10 times a day only during certain periods (*Liiv, 2017: 8*). The time window for information exchange is 5 to 15 minutes - this is the time during which the satellite is at a sufficient distance from the earth station (*Cakaj et al., 2007: 87*). Although the most favourable conditions for communication with terrestrial satellites are the most favourable, the connection between the earth station and the satellite is often disrupted (*Liiv, 2017: 8*). Due to the weak signal of the satellite and strong external noise, some data packets do not reach the earth station and the data connection speed is mostly under 10 000 bits per second (*Oro, 2018: 6; Liiv, 2017: 8*). This affirms the belief that constant optimization of critical data exchange infrastructure can benefit the ground control team’s day-to-day operations immensely.

3. Design of the Underlying System

3.1 The MCS Architecture

The ESTCube-2 MCS uses a microservice-based architecture, which means that the system consists of many different loosely coupled services each self-sufficient and performing a specific task. Every microservice has its own API and usually its own data storage. This modularity, even though making the entire application seem more complex, simplifies the application at the service level (*Singh & Peddoju, 2017:1*). Each service is easily understandable, self-documenting, testable, scalable, and resistant to architecture erosion. The services communicate with each other by using HTTP requests. The MCS scaling relies on the Docker Swarm[5] architecture. Docker Swarm is a container orchestration tool allowing users to conveniently manage several parallel deployments of the microservices.

The ESTCube-2 MCS comprises several microservices, illustrated in *Appendix 1*.

The microservices relevant in the context of this thesis are:

- MCS Front-end - A React-based[6] web application that serves as the visual user interface;
- MCS Back-end - Python 3 with FastAPI[7] web framework. Used to provide data access for the Front-end;
- UUID Generator - Generates universally unique IDs for packets and files;
- ICP Codec - An API for encoding and decoding data packets using the ICP protocol;
- AX.25 Codec - An API for encoding and decoding data packets using the AX.25 protocol;
- Command Table - A database used to store different commands;
- Packet Processor - Receives downlinked packets, parses them, and forwards them to other services.

3.2 The Nanosatellite Software Architecture

The ESTCube-2 on-board software architecture is multilayered (*Amor, 2018: 7*). Akin to the microservices architecture on-ground, the different logical layers of the subsystems allow a

[5] https://docs.docker.com/engine/swarm
[6] https://reactjs.org
[7] https://fastapi.tiangolo.com

reasonably complex system to exist while minimising the risk of failure and easing software development. All the layers communicate with the one directly below using APIs.

The different layers of the ESTCube-2 on-board software architecture include:

- **Application**: The custom code and utility logic reside here. Examples include scheduled tasks, logging, telemetry gathering etc.
- **Device Driver**: Contains the code required to address all connected peripherals through memory addresses and specified pins.
- **HAL Driver**: The HAL driver coordinates the exclusive use and freeing up of connected internal peripherals.
- **FreeRTOS**: A real-time operating system for microcontrollers.
- **Hardware Abstraction Layer**: Provides an abstraction over the complex hardware below to simplify software development. This includes abstracting the memory locations for a specific microcontroller.
- **Hardware**: The physical hardware on which the compiled binary/machine code executes.

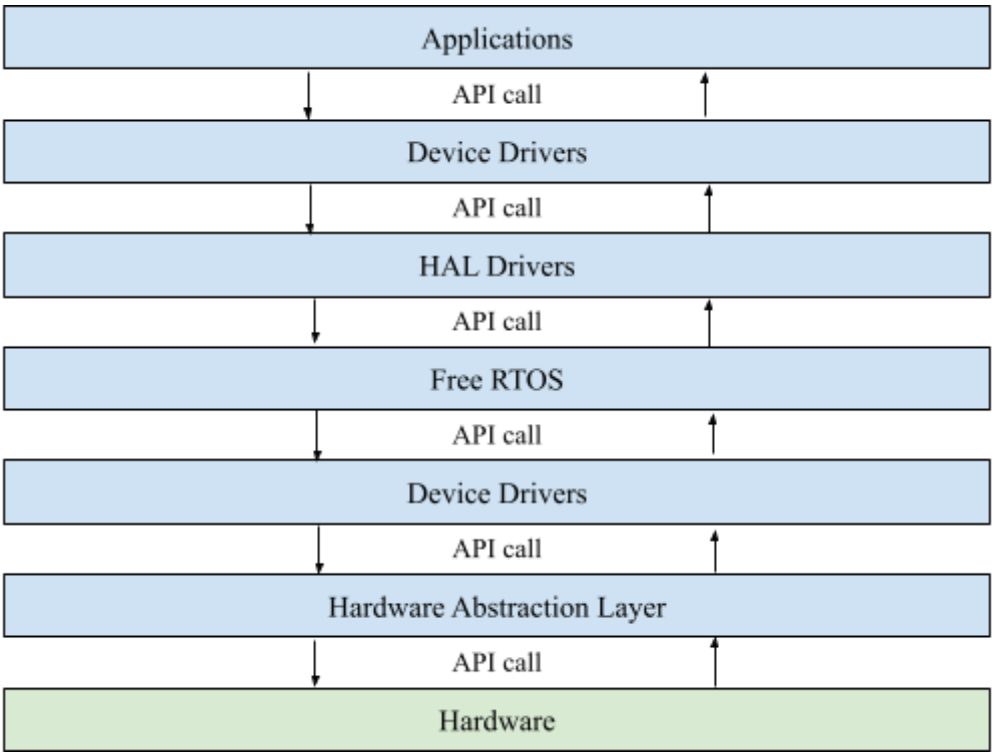


Figure 1. ESTCube-2 On-board software architecture

3.3 Packet Structure

The information exchange between the spacecraft and the ground stations are performed using radio signals (*Campbell, 2017*). For this, the information is divided into data packets. For ESTCube-2, the data is first encoded using the Internal Communication Protocol (ICP), a custom protocol developed for the ESTCube missions. The ICP packets have the following fields structure:

- Destination - The destination subsystem of the spacecraft or MCS;
- Source - The source subsystem of the spacecraft or MCS;
- Length - Length of the packet;
- Command - Directive defining how to behave when receiving a packet.
- Universally unique identifier (UUID) - A unique key allowing packets to be identified and grouped.
- Mode - Priority of the packet;
- Argument Blob - Arguments to execute the command;
- CRC - Cyclic redundancy check, an error-detecting code.

Table 1. Structure and size of the ICP packet

ICP Packet (10 - 248 bytes)							
DST	SRC	LEN	CMD	UUID	Mode	Argument blob	CRC
8 bits	8 bits	8 bits	8 bits	24 bits	8 bits	maximum 238 bytes	16 bits

The ICP packets are then encoded for a second time, using the AX.25 protocol. AX.25 is a data link layer protocol derived from the X.25 protocol and designed for use by amateur radio operators. The AX.25 packet contains standard fields, necessary to establish connections between two clients and to exchange information (*Beech et al. 1998*).

Table 2. Structure and size of AX.25

AX.25 Packet (maximum of 276 bytes)							
Flag	Address	Control	PID	INFO field		CRC	Flag
				ICP	HMAC-SHA224		
8 bits	112 bits	8 bits	8 bits	maximum 256 bytes		16 bits	8 bits

The structure of the packets has an effect on the way files have to be transferred. The size of the ICP packet argument blob is maximum 238 bytes which means that each packet can transfer a maximum 238 bytes of payload data. This is a relatively small space to house data, meaning files with the size of 1 Megabyte must be transferred using a minimum of over four thousand packets. For comparison, the size of the pictures taken by the satellite and downlinked to the ground station can exceed 50 Megabytes.

3.4 Communication

A queue system based on the RabbitMQ[8] message broker is used for sending and receiving data from the satellite. RabbitMQ is a lightweight open-source message message-queueing software, providing the functionality to create queues to which data packets can be pushed to or fetched from.

For the uplink, all packets to be transmitted are pushed into the uplink queue. From this queue the packets are fetched by the main ground station located at the Tartu Observatory[9] and sent to the satellite in orbit. For the downlink, packets sent by the satellite are received by a ground station and pushed to the downlink queue. From this queue, the packet processor service fetches the packets, decodes them using the ICP and AX.25 codecs and forwards the decoded contents down the communication pipeline.

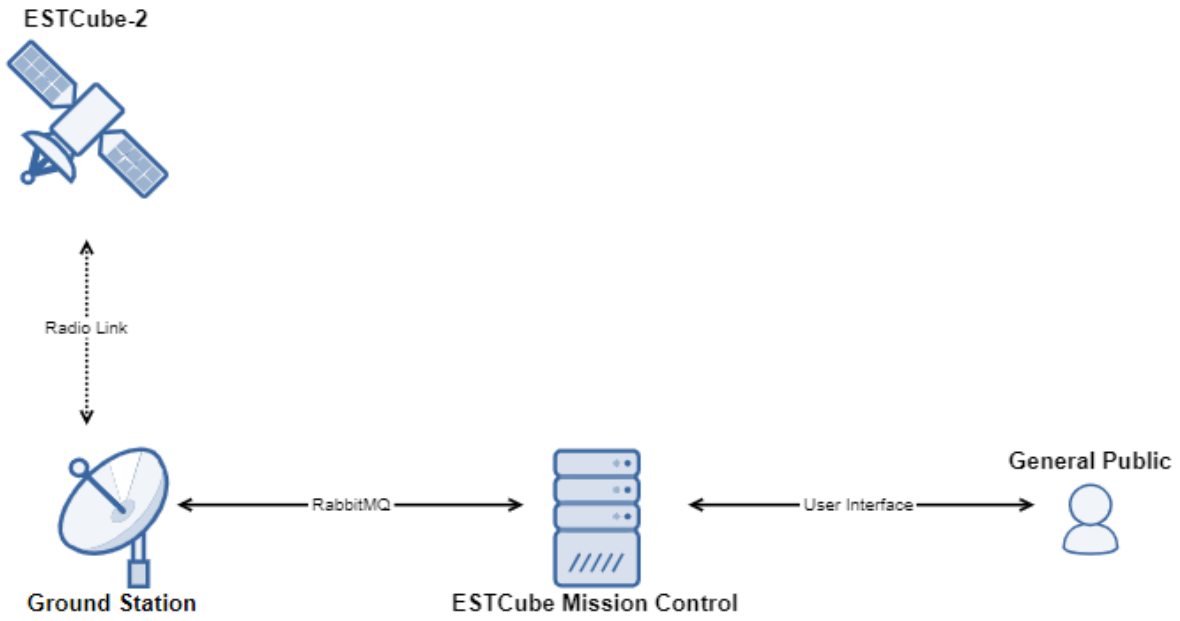


Figure 2. ESTCube-2 mission communication pipeline

[8] https://www.rabbitmq.com
[9] https://kosmos.ut.ee/en

The transmission rates of both the uplink and downlink are expected to be 9600 bits per seconds. Considering the maximum size of AX.25 packets is 276 bytes, around four packets will be transmitted each second. Both ICP and AX.25 frames contain CRC codes to identify errors in the data. If an erroneous packet is received by the satellite or the MCS, the whole packet is discarded.

4. Practical Solution

4.1 Problem Statement

Functionality that allows reliable and automated file transfers is of great importance for the ESTCube-2 mission, as is the ability to both downlink from and uplink files to the spacecraft. For a successful space mission, the spacecraft has to be able to downlink files to the ground station consisting of mission logs, payload data and photographs. The uplinked files that are sent from the ground station to orbit consist of firmware updates.

There are multiple factors that make the task of transferring files a difficult and complex procedure. The data exchange with the satellite is done on a packet by packet basis - the packets have a size of a few hundred bytes; thus to transfer files that can reach the size of over 50 Megabytes, the files have to be split up across multiple packets and then merged together at the destination. Also, the communication with the satellite is unreliable - packets can be lost and corrupted due to environmental and technical factors.

4.2 State of the Art

The MCS for the ESTCube-1 mission provided basic support for large file transfers and firmware updates, but suffered from the lack of automation. The ground control team had to constantly manage and monitor the state of file transmissions - if packet losses occurred, they had to be manually retransmitted or requested again. The ESTCube-1 MCS was a desktop based application (*powered by Python*) and used a different internal communication protocol making the reuse potential of code from the ESTCube-1 MCS to the ESTCube-2 mission extremely limited.

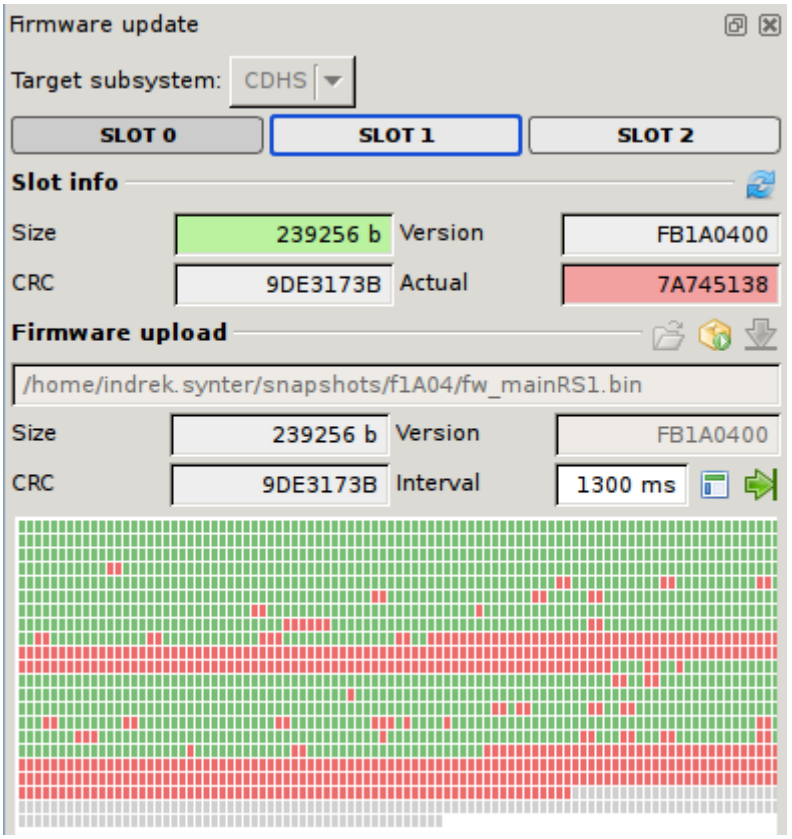


Figure 3. ESTCube-1 MCS file transfer window

The third-party mission control solutions were also considered; but in-house development was chosen because of the following reasons:

- The purpose of ESTCube is to train participating volunteers and develop expertise in-house.
- Third-party solutions often had proprietary implementations making them hard to integrate into the existing system.
- Open-source solutions such as YAMCS[10] and NASA's Open MCT[11] were also considered, but ultimately passed over as they had incompatible architectures or different mission philosophies.
- Also, the current satellite mission control systems have been developed and upgraded without a common architectural model, which has caused difficulties in the efforts to rework the software and to apply it to other satellite missions (*Rohling et al. 2019*).

[10] https://yamcs.org

[11] https://github.com/nasa/openmct

There was also considerable gap between the ESTCube-1 mission and ESTCube-2 mission, resulting in change of the entire development team, and lack of documentation for the ESTCube-1 MCS. Hence, it was decided that it would be wise to not inherit code from the previous mission control, but valid concepts like the pagemap have made it across to the ESTCube-2 MCS.

4.2 Requirements

The numerous requirements that the software has to satisfy, can be ordered by the level of priority. This separation has helped in managing the different tasks in the consequent order of implementation.

Table 3. Functional requirements of the software and their levels of priority

Functional Requirement	**Priority**
Automatically resend packets lost or corrupted without user input.	High
Store information about each file and the state of its transmission without errors.	High
Split files into compliant encoded packets which can be transmitted	High
Merge a file together using the data from received packets.	High
Provide functionality to uplink files with the only user input being inputting a command and providing the appropriate file.	High
Provide functionality to downlink files with the only user input being inputting a command and providing the path of the appropriate file.	High
Provide a visual interface where the user can see the different statuses of file transmissions.	Medium
Packets that have been successfully transmitted and received should not be retransmitted.	Medium

The developed solution has two main focus points. Firstly, the transfer of a file has to be reliable, meaning that it can handle packets being lost in transmission and can recover and

resend the lost data. Secondly, the solution should require minimal human input and intervention - the transfer process has to be fully automated.

Table 4. Performance Requirements of the software and their levels of priority

Performance Requirement	**Priority**
During uplink or downlink of a file the percentage of overhead (*percentage of data which is not file contents*) should be less than 20%.	High
The service should be able to operate on a machine with 16GB of RAM and Intel I-8250 processor (*server requirements*).	Medium
The service should be able to perform over 200% of the predicted maximum capacity of 9600 bits a second.	Medium

The efficiency and performance of the service are also important as the bitrates of the transmission links are slow - under 10,000 bits per second.

4.3 Deliverables

The required functionalities are to be provided by a service developed for the ESTCube-2 MCS. The service is called *File Transfer Service*. The created service is integrated with the existing front-end of MCS, to provide a visual user interface from which the functionality is accessible to the user.

The intended functionality also needs to be supported by the on-board software. As the on-board software system is in nascent stages of its development by the ESTCube-2 workgroup, the base to build the functionality does not yet exist. Thus, the work in this thesis includes the development of a mock implementation, which stands as a reference to help implement the functionality in the actual software used during the mission.

In summary, the developed software can be divided into three bigger work packages:

- File Transfer Service (FTS);
- A visual interface incorporated into the MCS Front-end;
- A mock on-board Application Programming Interface (API) for testing and development reference.

5. Implementation

5.1 File Transfer Service

File Transfer Service (FTS) is a Python back-end application providing the functionality to manage and execute the transfer of files. FTS uses the FastAPI framework to create endpoints to initialize the up/downlink of files and parse incoming data packets. FastAPI is a modern, high-performance framework for quickly building robust APIs. FastAPI was selected as the framework, due to its wide adoption in other services powering the ESTCube-2 MCS. Preventing the MCS technology stack from becoming too diverse is important, as an extremely diverse stack is difficult to maintain.

Table 5. API endpoints provided by the FTS

Endpoint	**HTTP Method**	**Parameters**	**Purpose**
`/uplink_file`	POST	File, destination path, destination subsystem	Initiate the uplink of a file
`/downlink_file`	POST	Destination path, destination subsystem	Initiate the downlink of a file
`/files`	GET	-	Get a list of all files
`/file/{uuid}`	GET	UUID	Download a file
`/packet`	POST	Decoded ICP packet	Process a received packet

The `/uplink_file` and `/downlink_file` endpoints are used to initiate the uplink or downlink of a file. The `/files` endpoint provides a list of all files in the database. `/file/{uuid}` is used to download a specific file. The `/packet` endpoint is used to POST downlinked packets to the service.

The application is divided into separate python modules. Each module contains functionality for a specific subdomain:

- `main.py`: The main module which starts the service and initiates the API. The API provides five endpoints;
- `file_operations.py`: Provides methods for file input and output operations such as creating files, fetching data from files, writing received data to files and etc;

- `db_interface.py`: Contains functions allowing the application to access its internal database and fetch and update data. It uses the apsw[12] library to provide interfacing support;
- `packet_handler.py`: Contains the logic for handling all received packets from the satellite;
- `queue_checker.py`: Has a series of workers which resend packets should no response from the satellite is received. Every set interval, it checks the status of all files in the database and fetches the contents of the RabbitMQ uplink queue and analyses if any responses have timeout. If so, it resends the necessary packets;
- `rabbitmq_connector.py`: Provides interfacing with the RabbitMQ message broker using the pika library, which is a RabbitMQ client library for Python;
- `service_provider.py`: Provides methods to connect to other MCS services. It provides access to the UUID Generator and AX and ICP Codecs;
- `utils.py`: Provides a variety of different functions which are used by the other modules.

The application uses a SQLite[13] database for data storage. SQLite is a widely used open-source database engine (*Romanowski, 2020*). It is light, fast, and highly reliable, and fulfills all the storage requirements of the FTS .

5.2 User Interface

The user interface of the FTS is a component added to the existing React based MCS Front-end. The service uses the Axios[14], a JavaScript library, for contacting the back-end API in order to communicate with the FTS. It is available for the user of the MCS from the side menu and consists of two pages - one for viewing and managing the uplink of files, the other for the downlink.

[12] https://rogerbinns.github.io/apsw
[13] https://www.sqlite.org
[14] https://github.com/axios/axios

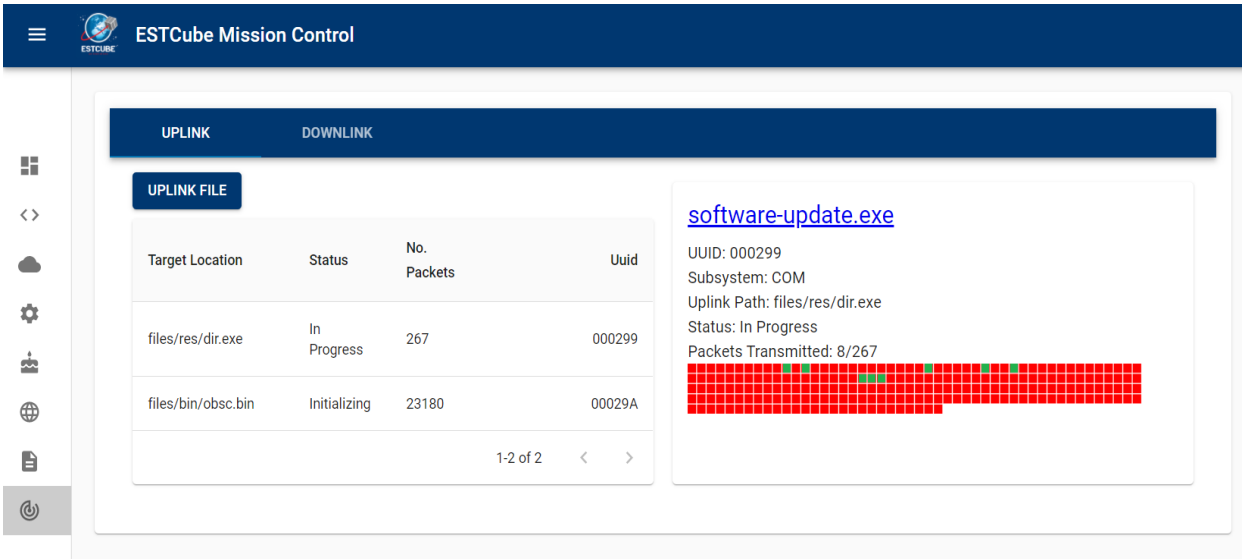


Figure 4. FTS user interface, showing the uplink page

Both pages display a table of all the uplinked and downlinked files. The table rows can be clicked upon to see the details of each file and the state of its transmission. A bitmap is rendered which displays the status of each packet. A form to initiate the uplink or downlink of a file is also available.

5.3 Onboard Software Simulator

A mock of the proposed on-board functionality was implemented using Python Flask[15], a micro-web framework. This application allows the created service to be tested and serves as a reference for further development, when the service will eventually be integrated with the actual on-board software and rewritten in the C programming language.

```
def handle_packet(packet):
    # Handles received packets depending on command

    # Decode received packet
    icp = disassemble(packet)

    # Decide action based on command
    if icp.command == Commands.START_UPLINK:
        start_uplink(icp)
    elif icp.command == Commands.INIT_DOWNLINK:
```

[15] https://flask.palletsprojects.com/en

```
        initialize_downlink(icp)
    elif icp.command == Commands.START_DOWNLINK:
        start_downlink(icp)
    elif icp.command == Commands.DATA_PACKET:
        save_file_chunk(icp)
    elif icp.command == Commands.REQUEST_BITMAP:
        transmit_bitmap(icp)
    elif icp.command == Commands.BITMAP:
        update_bitmap(icp)
```

Figure 5. Code for handling packets

The created mockup provides the functionality to uplink and downlink files. It fetches packets directly from the RabbitMQ uplink message queue and decodes them.

5.4 Data Structure

5.4.1 File Transfer Service Database

The database used by the File Transfer Service comprises two data tables - Files and Bitmaps. In the *Files* table, each entity represents a file which is being downlinked or uplinked. The *Bitmaps* table is used to represent sections of the bitmaps.

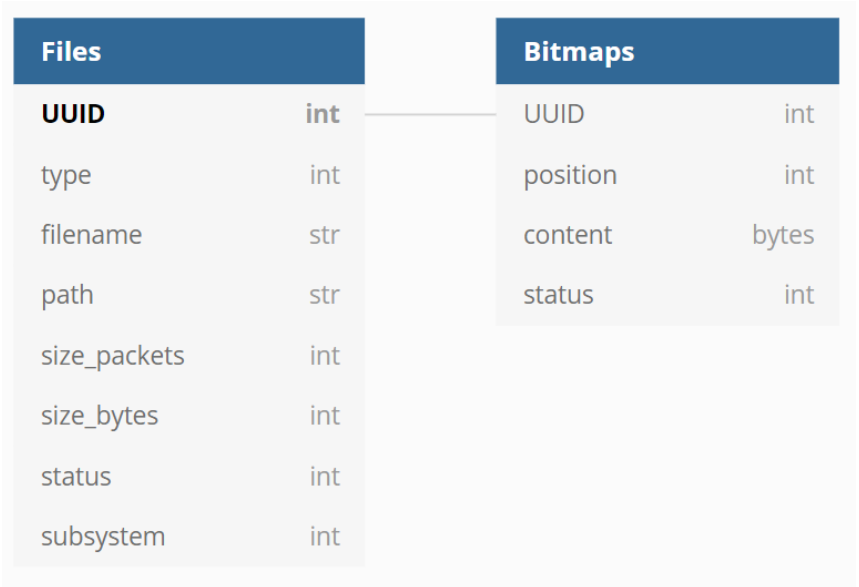


Figure 6. Structure of the database

All files that were uploaded to the service for uplink are stored by the service in the `/uploads` directory.

5.4.2 Bitmaps

A bitmap based solution is used to keep track of which packets have been successfully transmitted. The bitmap map is an array of bits in which each bit signifies the status of a given packet - if the bit at a given index is one, then the packet mapped to that location has been successfully delivered, if it is zero then it has not. Bitmaps provide a compact method to track and transmit the status of packets - with a single packet which contains a bitmap it is possible to update the statuses of hundreds of packets.

A single bitmap can store information as many packets as it has bits - meaning the size of the bitmap is equal to the number of packets the file is split into. For larger files the bitmaps get large enough that a single packet cannot contain the whole bitmap. Thus the bitmap must be split into sections with a position index. For uplink, the bitmap is stored on-board and for downlink the bitmap is stored on ground.

5.4.3 Onboard Storage

Onboard storage is a simple mapping. The referencing is done when receiving a new packet with an initialization command for uplink or downlink. The provided file path is referenced to the packet's UUID and in case of uplink initialization a bitmap is created which is also referenced to the UUID. As a result, when packets containing file chunks are received, the UUID is used to work out the file path where the chunks should be saved.

5.4.4 Commands

Each ICP packet in the ESTCube-2 mission contains a command field, which defines the intention of the packet and how the receiver should behave when receiving it. Each command has a set of arguments, which are stored in the ICP packet argument blob. Tables 6 and 7 showcase all provided commands used by the developed functionality. They are to be stored in the Command Table - a MCS service which stores all the commands in use for the mission.

Table 6. Commands sent from ground station to satellite

Command Name	**Arguments**	**Description**
Initialize Uplink	[:3] File Size	Initializes the uplink. Satellite maps the

	[3:] File Path	UUID to the provided path and generates bitmaps for the provided file size
Initialize Downlink	File Path	Initializes the downlink. Provides the file path of the file to be downlinked.
Confirm Downlink	-	Confirms that the MCS has received the necessary information and is ready to receive data packets.
Data Packet	[:3] Position [3:] File Data	Data packets which contain the contents of the uplinked file. The satellite will write the contents to the specified position and update the bitmaps.
Request Bitmap	[0] Bitmap Index	Request for the bitmap to have data which sections of the file have been successfully downlinked
Bitmap	[0] Bitmap Index [1:] Bitmap	Satellite received the bitmap to have data which shows which packets have been successfully downlinked

Table 7. Commands sent from satellite to ground station

Command Name	**Arguments**	**Description**
Initialize Uplink	-	Confirms that the satellite has received the necessary information and is ready to receive data packets.
Initialize Downlink	[3:] File Size	Initializes the uplink. MCS generates bitmaps for the provided file size
Data Packet	[:3] Position [3:] File Data	Data packets which contain the contents of the uplinked file. The MCS will write the contents to the specified position and update the bitmaps.
Request Bitmap	[1] Bitmap Index	Send a request to receive the bitmap with the given index
Bitmap	[1] Bitmap Index [1:] Bitmap	MCS received the bitmap to have data which shows which packets have been successfully uplinked

The commands and the structures of their arguments affect the capabilities of the service. For bitmap packets, one byte is used to define the bitmap index. This sets the limit to the largest file size which is supported by the service. The maximum size of the file can be calculated using the equation:

$$Maximum\ size\ =\ maximum\ index\ \times\ maximum\ bitmap\ length\ \times\ data\ packet\ chuck\ size\ \times\ bits\ per\ byte$$

As the maximum index is 256, maximum bitmap length is 237 bytes, data packet chuck size is 235 bytes and there is 8 bits per byte, the maximum supported file size is roughly 112 Megabytes.

5.5 Linking

5.5.1 Uplinking

Uplinking is the process of sending data to the satellite from the ground station. For the ESTCube-2 mission, files containing software updates have to be uplinked to the satellite. A bitmap based solution was implemented to provide the functionality to uplink files.

The uplinking process is started by the end user from the MCS frontend file transfer section. The user opens the form to start the uplink, specifies the destination path of the file, target subsystem and also uploads the file, which will be uplinked. The data is sent to the FTS using a post request.

The FTS contacts the UUID generator services to generate an UUID for the uplinked process. Next the service analyses the uploaded file and reads its size. Now the initialization packet is generated, which informs the satellite of the proceeding data transmission and provides its parameters - destination path and size. It uses the command Initialize Uplink. This packet is encoded using the AX.25 Codec and ICP Codec services and sent to the RabbitMQ Uplink queue to be uplinked from the ground station. When the satellite receives the initialization packet, it will create the file in the provided path and save the mapping from the UUID to the path in a database. In addition, bitmaps are generated based on the provided file size to keep track which packets have been received. Then a confirmation packet which also uses the Initialize Uplink command is sent back to the ground station.

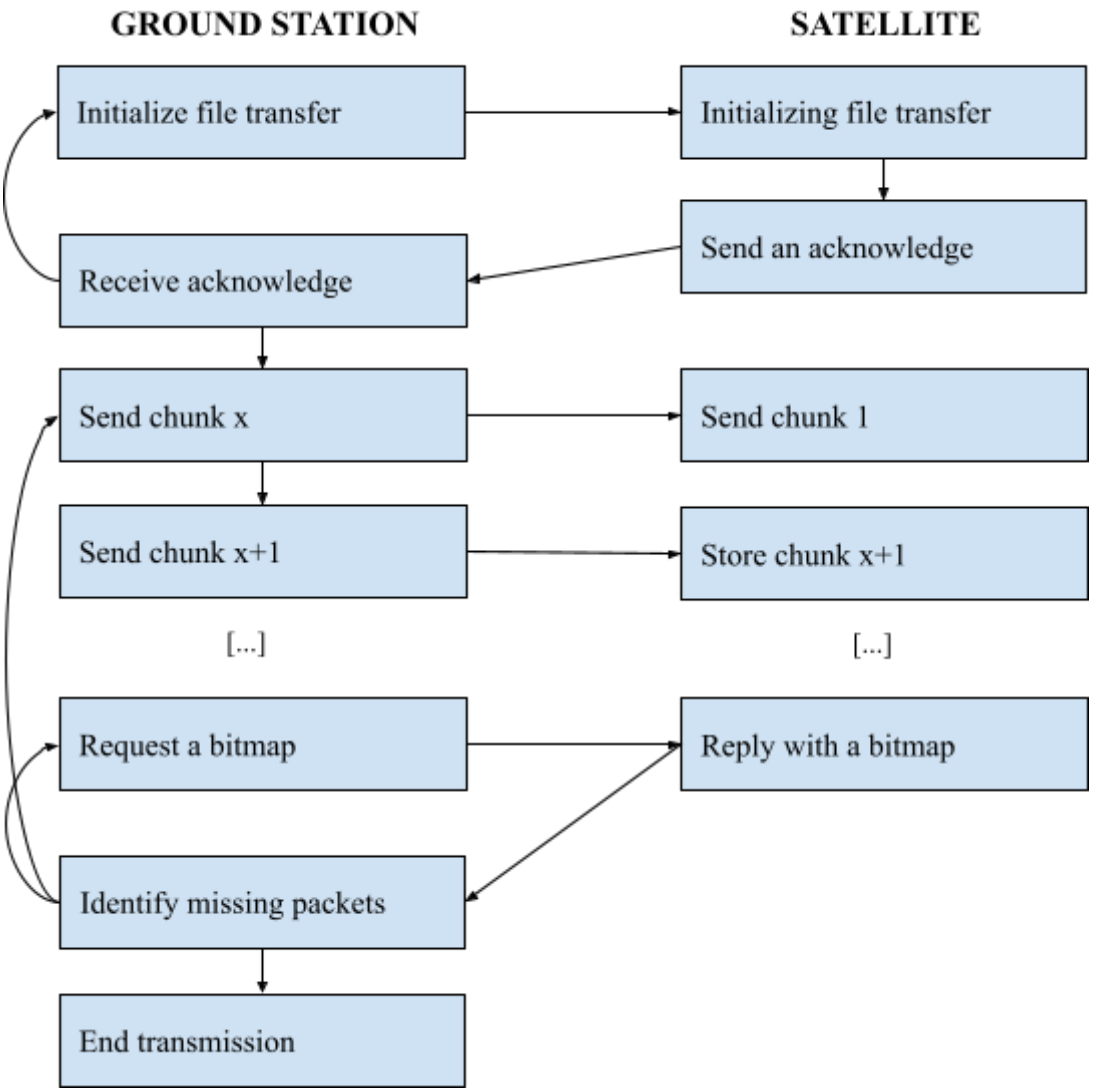


Figure 7. Uplink logic for ESTCube-2

When the response packet is received by the MCS, the transfer of data packets will begin. The FTS will divide the file that is being uplinked into chucks, which are small enough to be contained in the ICP packet. Three bytes, which reflect the position of the chuck in the file are prepended to the chuck. Each file chuck is encoded using the ICP and AX.25 Codec services and uses the Data Packet command. The packets are then uplinked to the satellite. After transmitting all the packets containing file chucks, packets for requesting the updated bitmap sections are transmitted.

When the satellite receives a data packet, it reads the first three bytes of the ICP argument blob - this reflects the position of the file chuck. The rest of the ICP argument blob contains the file chuck - this is written to the corresponding file in the provided position and in the

bitmap the bit at the provided position is updated to one - reflecting the fact that the file chuck at that position has been received. When the bitmap request is received, the updated bitmap is downlinked. When the bitmap is received by the MCS, the positions of the packets that have not been successfully uplinked are identified by inspecting the zeros in the updated bitmap. All these packets are retransmitted and a request for the bitmap is again transmitted. This process continues until the bitmap only consists of *1*s meaning that every packet has been received. Here the uplink process ends and is marked as completed.

5.5.2 Downlinking

Downlinking is the reverse process of uplinking - transmitting data from the satellite to the ground station. There are three different types of data that will be downlinked from the ESTCube-2 nanosatellite - logs, payload data, and subsystem data. The downlink process is analogous to the uplink process as the satellite will transmit packets to the ground station and receive back a bitmap to update packets' status and retransmit lost packets.

To downlink a file from the satellite, the user has to initiate the downlink from the user interface which is provided by the MCS frontend service. The user needs to provide two variables - the path of the file they wish to downlink and the target subsystem. This information is posted to the FTS. The FTS contacts the UUID Generator service to receive an UUID to identify the downlinking instance. An initialization packet is generated, encoded in ICP/AX.25. This is forwarded to the RabbitMQ to be uplinked to the satellite.

When the satellite receives the initialization packet, it reads the information about the specified target file. The path of the file is mapped to the UUID in the on-board database and a confirmation packet, which specifies the size of the file is generated and downlinked. When the FTS receives the confirmation packet, it uses the provided file size to generate bitmaps which denote which packets have been received and which not. A packet is sent back to the satellite confirming that the ground segment is ready to receive packets containing file chunks.

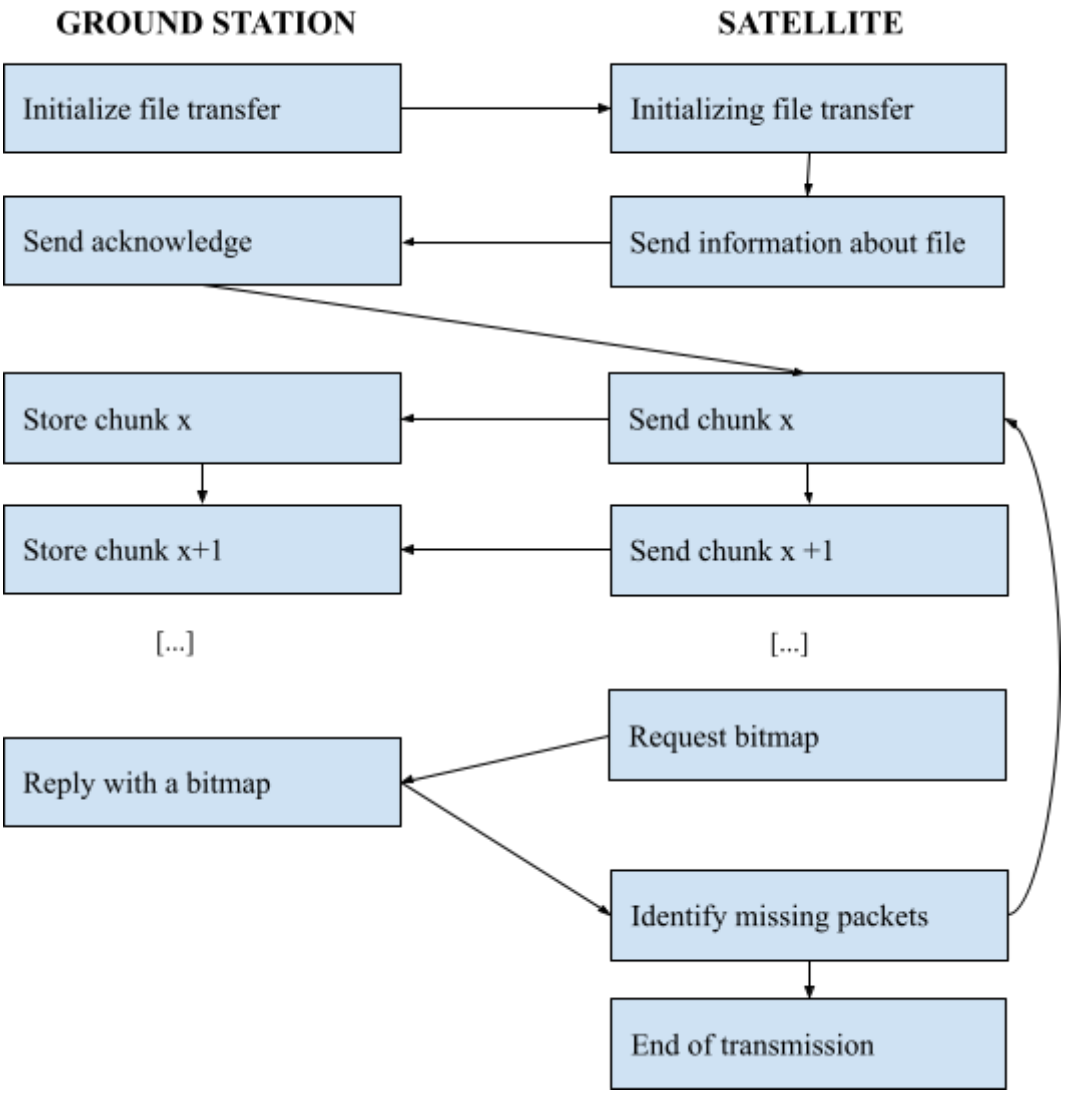


Figure 8. Downlink logic for ESTCube-2

When the satellite receives that packet which informs that all is ready for downlink, it proceeds with starting the transmission of data packets. The file, which is to be downlinked is divided into chunks, each chunk is encoded and downlinked. After transmitting all the packets containing file chucks, a packet containing the request for the updated bitmap is transmitted.

The data packets are received by the FTS and their contents are written to a file and the bitmap is updated - the updated bitmaps are uplinked when a request for them is received. The satellite received the updated bitmap and retransmitted the chucks that were not successfully transmitted and again a request is sent for the update bitmap. This process continues until all file chunks have been successfully transmitted.

5.5.3 Packet Retransmission

As the packets can become lost or corrupted, a method to retransmit them is required. For the packets which contain file chunks, the bitmap method which was previously described is used. But for packets, which manage the transmission process like the initialization packets or request for bitmaps, an alternative method is required.

This is managed by the `queue_checker.py` module in the FTS. Every set interval of time, it fetches the files in the database and checks if any of them have sent out a request and are waiting for a response to it. If they are, then it checks if the request is still in the uplink queue. If not, then the request has been transmitted to the satellite and a response should be received soon. Thus a timer is started. If the timer ends before a response is received, then it is assumed that the packet was lost and thus the packet is retransmitted.

6. Results

6.1 Testing

To validate the functionality of the developed services and evaluate its performance, a series of tests were conducted. Both uplink and downlink functionalities were evaluated. Three files with different sizes were used - 10 Kilobytes, 500 thousand bytes and 10 million bytes. One series of tests was run without simulating packet loss and in the other 10% of packets were lost.

To verify that the file was transmitted successfully and all functional requirements were met, SHA-256 checksums were generated for both the original file and the received file. If the checksums are identical then it is reasonable to conclude that the file was transmitted without errors and both files are identical (*A2 Hosting, 2021*). To simulate packet corruption and erasure, the Python random library was used to drop random packets.

6.1.1 Uplink

For errorless uplink, the overhead percentages were in the range of 14% to 17%, as seen in table 8. Most of this can be attributed to the AX.25 and ICP packet overhead - For the testing each packet had 30 bytes of overhead from fields and the contained a maximum of 227 bytes of data - resulting in a minimum overhead of 13.21% per packet. If this is taken into account, the overhead percentage resulting from the operations of the FTS is less than 5% for all file sizes. Bigger file size resulted in relatively smaller percentages of overhead - this is the consequence of the fact that the initialization packets made up a smaller portion of the total bytes transmitted.

Table 8. Test results for uplink with no errors

File size in bytes	**Bytes transmitted from ground to satellite**	**Bytes transmitted from satellite to ground**	**Total Bytes transmitted**	**Percentage overhead**
10000	11557	67	11624	16.24%
500000	573792	372	574164	14.83%
10000000	11474097	6900	11480997	14.81%

The percentage of of extra bytes needed to be transmitted with a given error rate can be calculated with the formula:

$\sum_{n=1}^{\infty} x^n$, where x is the error rate (in this test case, 0.1 = 10%)

Using the formula, retransmitting packets should the overhead percentage increase by 11.11%. But by subtracting this percentage, the overhead percentage is still larger than before - around 16% for the larger files and almost 30% for the small file. This increase can be explained by the need to transmit bitmaps multiple times - the amount of bytes transmitted from satellite to ground increased up to 4 times.

Table 9. Test results for uplink with 10% error rate

File size in bytes	**Bytes transmitted from ground to satellite**	**Bytes transmitted from satellite to ground**	**Total Bytes transmitted**	**Percentage overhead**	**Percentage overhead minus predicted retransmission rate**
10000	13688	215	13903	39.03%	27.92%
500000	638033	1398	639431	27.89%	16.78%
10000000	12739674	29121	12768795	27.69%	16.58%

The performance requirements for overhead are achieved for all cases except the file with the size of 10000 bytes and 10% error rate. But as the file is small and it's transmission doesn't take significant resources, the larger that required overhead does not constitute a significant issue. Also other requirements for met - the service performed its functionality without problems on the required hardware and managed transmission rates far greater than required - file with the size of 10 million bytes was transferred in 10 minutes (*at a rate of 9600 per second it would take more than two hours*).

6.1.1 Downlink

The results of testing the downlink functionality were very similar to the results of testing the uplink functionality. For errorless transmission the overhead percentages were around 15% - a bit greater for smaller files, smaller for larger files.

Table 10. Test results for downlink with no errors

File size in bytes	Bytes transmitted from ground to satellite	Bytes transmitted from satellite to ground	Total Bytes transmitted	Percentage overhead
10000	132	11550	11682	16.82%
500000	437	573785	574222	14.84%
10000000	6511	11474028	11480539	14.81%

With an error rate of 10%, the amount of overhead was elevated, but remained in the acceptable limits. The transfer of the 10000 byte file was surprisingly efficient - this could be explained by crucial initialization and bitmap packets not being lost.

Table 11. Test results for downlink with 10% error rate

File size in bytes	Bytes transmitted from ground to satellite	Bytes transmitted from satellite to ground	Total Bytes transmitted	Percentage overhead	Percentage overhead minus predicted retransmission rate
10000	169	12609	12778	27.78%	16.67%
500000	1323	642816	644139	28.83%	17.72%
10000000	29652	12765398	12795050	27.96%	16.84%

All performance requirements were achieved for the downlink functionality. Hardware was sufficient, amount of overhead was low and transmission speeds were far greater than required.

6.2 Proposed Improvements

6.2.1 Packet Level Error Correction

Packet level error correction is a method of error correction where additional redundant packets are generated and transmitted. The receiver can use these redundant packets to recover other packets that were lost (*Milani, 2006: 18*). Implementing a packet level error

correction solution could increase the performance of the developed solution and reduce the amount of retransmissions (*Mazzotti et al., 2011: 1*). A greater percentage of file chunks could be with each iteration of transmissions, thus the amount of bitmap requests and responses could be limited - this would result in greater speed and efficiency of the file transfer process. An optimal algorithm for this error correction should be chosen - taking account of the properties of the erasures and errors that occur and the computing power of the on-board software.

6.2.2 Improved Analytics

The developed solution provides just primitive logs and statistics about its operations. Many important metrics can be recorded to provide a comprehensive overview of the transfer process:

- Time of transfer initiation;
- Time of transfer completion;
- Total amount of packets sent;
- Retransmission count for each file chunk;
- Packet loss rate.

All these metrics could help give a better info of the health of the communication link and help better monitor the transfer of files.

6.2.3 Onboard File Directory Browsing

When initiating the uplink or downlink of a file, the user has to input the path of the file as a string. This can cause issues when the file or directory does not exist and requires independent work to look up the paths in the on-board file system. The developed solution provides no overview of the or information about on-board directory structure. This makes the workflow much more difficult.

A solution should be developed, which would allow the user to browse the on-board directory and select the desired path or file. It would request information from the satellite about the state and structure of its file directories. The data could then be used to display the on-board file system from the MCS and allow users to easily find and define the file paths and destinations.

7. Conclusion

This thesis presented the results of creating a reliable and automated service for transferring files that will be implemented on the ESTCube-2 nanosatellite mission. The objective of the thesis was to create functionality for the ESTCube-2 MCS, which would analyze, track and synchronise files and also deal with detection and resolution of losses and errors that occur during data transmission between the nanosatellite and the ground station. To reach this objective, three main deliverables were created - a service named File Transfer Service was developed for the MCS, a user interface to support the functionality was implemented in the ESTCube MCS front-end and a mock-up of the proposed on-board software was created for future reference and testing.

A set of requirements were set for the functionality and performance of the service. The fulfillment of these requirements was verified by a series of tests - files of different sizes were uplinked and downlinked in both an errorless scenario and a scenario with 10% error rate. All the set requirements were fulfilled with one small exception, which is insignificant in the context of actual space mission situations.

Proposals were also provided for the further development of the service, to ensure its maximal efficiency and reliability. These proposals included the implementation of a packet level error correction, logging and file directory browsing functionality.

The objectives of this thesis can be considered fulfilled, as a working solution was provided. To finalize the work in this thesis, the on-board software mockup should be implemented in the actual on-board software.

References

1. A2 Hosting (2021). How to work with file checksums. https://www.a2hosting.com/kb/developer-corner/linux/working-with-file-checksums [Accessed 07.05.2021]
2. Almansoori, F., Marpu, P., Aung, Z. (2018). Improved Power Control Approach for Better Data Throughput in CubeSat Nanosatellites. *International Conference on Innovations in Information Technology (IIT), 2018, pp. 52-57, doi: 10.1109/INNOVATIONS.2018.8605944.*
3. Amor, E. (2018). Design and implementation of prototype firmware for ESTCube-2 primary communication subsystem. *UT Institute of Computer Science Bachelor's Thesis* https://dspace.ut.ee/handle/10062/60289
4. Beech, W. A., Nielsen, D. E., Taylor, J. (1998). AX.25 Link Access Protocol for Amateur Packet Radio. *American Radio Relay League (ARRL) and the Tucson Amateur Packet Radio Corporation (TAPR)* http://www.tapr.org/pdf/AX25.2.2.pdf
5. Cakaj, S., Keim, W., Malarić, K. (2007). Communications duration with low earth orbiting satellites. *Proceedings of the Fourth IAESTED International Conference on Antennas, Radar, and Wave Propagation / Yao, J. - Montreal : IAESTED, 2007, 85-88.* https://publik.tuwien.ac.at/files/pub-et_12772.pdf
6. Campbell, A. (2017). How do satellites communicate? *National Aeronautics and Space Administration* https://www.nasa.gov/directorates/heo/scan/communications/outreach/funfacts/txt_satellite_comm.html [Accessed 07.05.2021]
7. Dalbins, J., Ehrpais, H., Ilbis, E., Iakubivskyi, I., Oro, E., Eenmae, T., Sünter, I., Janhunen, P., Envall, J., Toivanen, P., Sate, J., Trops, R., Pajusalu, M., Noorma, M., Slavinskis, A. (2017). ESTCube-2 integrated platform for interplanetary missions. *EPSC Abstracts. Vol. 11, EPSC2017-946, 2017, European Planetary Science Congress 2017.*
8. Dessoy, D., van Schie, B., Schwarzenbarth, K., Masquelier, V., Lahaye, V. (2017) Preliminary ground segment impact analysis. *SESAR Joint Undertaking under European Union's Horizon 2020 research and innovation programme.* https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5b6bd6341&appId=PPGMS

9. Ehrpais, H., Sünter, I., Ilbis, E., Dalbins, J., Iakubivskyi, I., Kulu, E., Ploom, I., Janhunen, P., Kuusk, J., Sate, J., Trops, R., Slavinskis, A. (2016). ESTCube-2 mission and satellite design. *4S 2016 Final Proceedings: The 4S Symposium (Small Satellites Systems & Services), 30 May-03 June, 2016, Valletta, Malta.* ESA, 1−7.
10. ESTCube-1 (2020). https://www.estcube.eu/estcube-1-0 [Accessed 07.05.2021]
11. ESTCube-2 (2020). http://www.estcube.eu/en/estcube-2 [Accessed 07.05.2021]
12. Haljaste, H. (2017). Electronics design and testing for ESTCube-2 on-board computer system with sensors for attitude determination. *UT Master's Thesis*, *UT Institute of Technology*. https://dspace.ut.ee/handle/10062/56547
13. Iakubivskyi, I., Ehrpais, H., Dalbins, J., Oro, E., Kulu, E., Kütt, J., Janhunen, P., Slavinskis, A., Ilbis, E., Ploom, I., Sünter, I., Trops, R., Merisalu, M. (2016). ESTCube-2 mission analysis: Plasma brake experiment for deorbiting. *67th international astronautical congress (IAC 2016): Making space accessible and affordable to all countries (IAC-16,E2,4,4,x33190),* International Astronautical Federation. (Cit. on pp. 6, 7).
14. Li, J., Post, M., Wright, T., Lee, R. (2013). Design of Attitude Control Systems for CubeSat-Class Nanosatellite, *Journal of Control Science and Engineering, vol. 2013, Article ID 657182, 15 pages, 2013.* https://doi.org/10.1155/2013/657182
15. Liiv, K. (2017). Contact Automation For The ESTCube-2 Mission Control System. *Bachelor's Thesis, UT Institute of Computer Science* https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=58329
16. Mabrouk, E. (2017). What are SmallSats and CubeSats? https://www.nasa.gov/content/what-are-smallsats-and-cubesats [Accessed 07.05.2021]
17. Mazzotti, M., Paolini, E., Chiani, M., Gadat, B., Bergeron, C., Fracchia, R. (2011). Analysis of Packet-Level Forward Error Correction for Video Transmission. *1-5. DOI: 10.1109/VETECS.2011.5956635*.
18. Milani, S. (2006). Source and Joint Source/Channel Video Coding for lossy networks. *Ph.D. Thesis, University of Padova, Padova, Italy.* http://www.diegm.uniud.it/rinaldo/PRIN/PRIN_2005/Forward_Error_Correction_codes_at_the_packet_level.html [Accessed 07.05.2021]
19. Nanosats (2021). https://www.nanosats.eu/sat/estcube-2 [Accessed 07.05.2021]
20. Oro, E. (2018). Delta Updates for ESTCube-2 Onboard Computer Software. *UT Bachelor's Thesis, UT Institute of Computer Science* https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=61695

21. Rohling, A.J., Neto, V.V.G., Ferreira, M.G.V., Dos Santos, W.A., Nakagawa, E.Y. (2019). A reference architecture for satellite control systems *Innovations Syst Softw Eng 15, 139–153.* https://doi.org/10.1007/s11334-019-00322-w
22. Romanowski, J. (2020). The Most Popular Databases in 2020. https://learnsql.com/blog/most-popular-sql-databases-2020/ [Accessed 07.05.2021]
23. Singh, V., Peddoju, S. K. (2017). Container-based microservice architecture for cloud applications. *International Conference on Computing, Communication and Automation (ICCCA), 2017, pp. 847-852, doi: 10.1109/CCAA.2017.8229914.*

Appendices

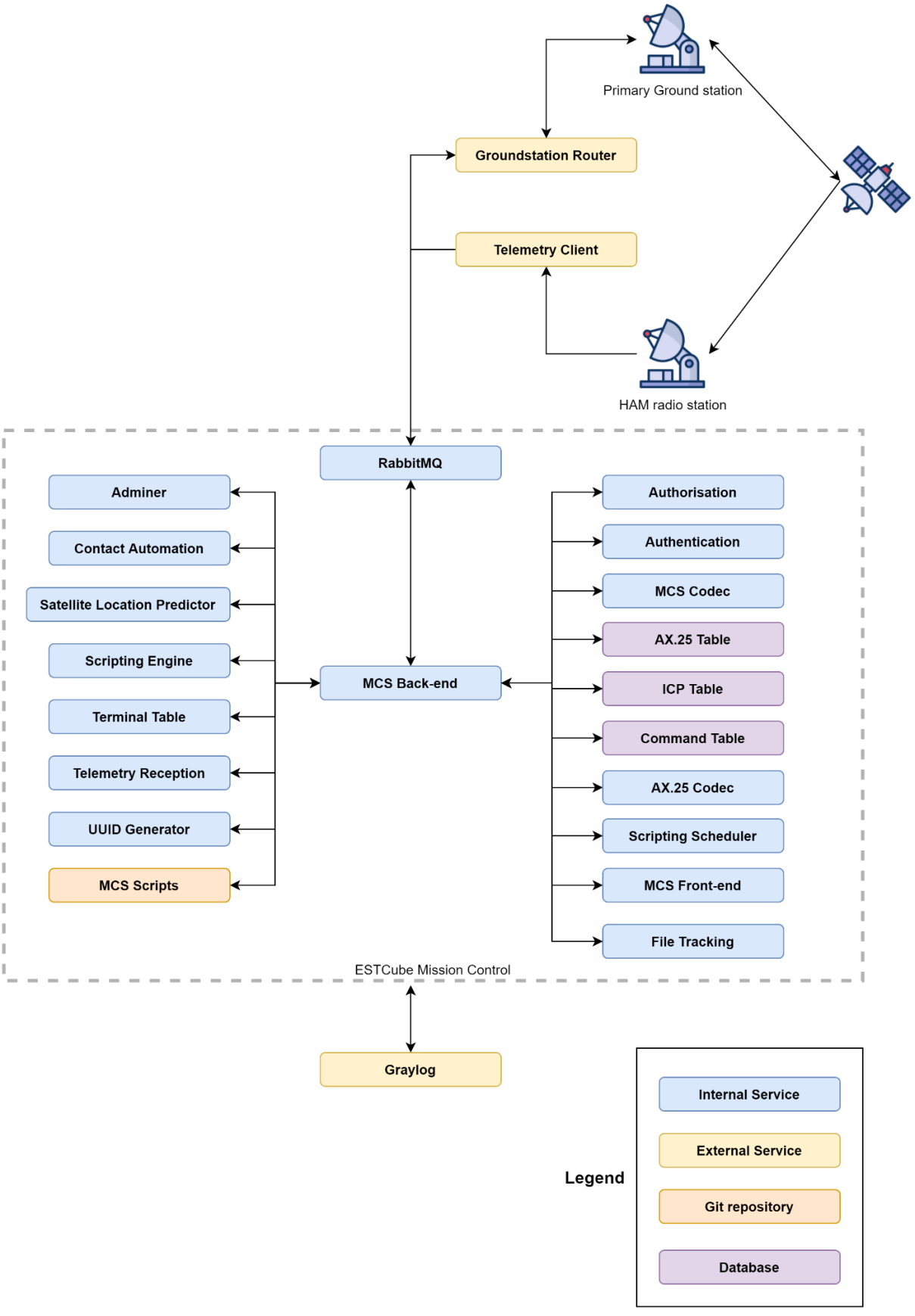
I Source Code

The source code of the software developed in the course of this work is available in the ESTCube-2 git repositories.

The File Transfer Service:
https://bitbucket.estcube.eu/projects/ECIIMCS/repos/experimental-packet-handling.
The Front-End:
https://bitbucket.estcube.eu/projects/ECIIMCS/repos/gui-mcs-front-end.
The On-Board Software Simulator:
https://bitbucket.estcube.eu/projects/ECIIMCS/repos/api-packet-processor-onboard.
Instructions on how to start the services are provided in the readme files or here https://confluence.tudengisatelliit.ut.ee:8433/display/EC2MCS/Getting+started.

Permission to access the code repositories and documentation is managed by the Estonian Student Satellite Foundation. For more access, please contact: Umesh Anilchandra Bhat <*umesh.bhat@estcube.eu*>.

II The Complete ESTCube MCS Architecture



III Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Teet Saar,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

    1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

    1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright, of my thesis

Reliable File Transfer for the ESTCube-2 Nanosatellite On-Orbit,

supervised by Umesh Anilchandra Bhat, Dr. Boris Kudryashov, and Dr. Vitaly Skachek.

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Teet Saar
Tartu, **07.05.2021**