NTNU – Trondheim
Norwegian University of
Science and Technology

Design and Implementation of a Reliable Transport Layer Protocol for NUTS

Erlend Riis Jahren

Master of Science in Electronics
Submission date: June 2015
Supervisor: Bjørn B. Larsen, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

"People who think they know everything are a great annoyance to those of us who do."

– Isaac Asimov

Abstract

The NTNU Test Satellite (NUTS) is a double CubeSat developed mainly by students and volunteers at the Norwegian University of Science and Technology (NTNU). One of the main goals of the NUTS mission is to establish a communication channel between the satellite and a ground station, enabling collection of data from the satellites sensors, and receiving pictures of the earth taken from the satellites camera.

The satellite is expected to be launched into low earth orbit, where the presence of cosmic radiation is known to cause erroneous behavior in electronic hardware. This generates a demand for reliability in the communication network, using redundant techniques for error detection and correction.

This thesis aims to improve the NUTS communication network by proposing a design and implementation of a reliable transport layer protocol. The protocol, named NUTS reliable protocol (NRP), features segmentation of large payloads, error detection, and error correction through retransmission of corrupt data. NRP was implemented for the NUTS software repository, with a simple API to facilitate further software development within NUTS.

The protocol has been tested and the results have been discussed, concluding a successful design and implementation of a transport layer protocol for the NUTS CubeSat mission. However, further testing is advised to maximise the protocols performance in the NUTS network.

Sammendrag

NTNU Test Satellite (NUTS) er en dobbel CubeSat utviklet hovedsakelig av studenter og frivillige ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU) . Et av de viktigste målene for NUTS er å opprette en kommunikasjonskanal mellom satellitten og en bakkestasjon, slik at sensordata og bilder tatt fra satellitten kan sendes til jorden.

Satellitten skal skytes ut i lav jordbane, der kosmisk stråling er kjent for å kunne forårsake feil i elektronisk maskinvare. Dette skaper et behov for pålitelighet i kommunikasjonsnettet, ved hjelp av redundante teknikker for feildeteksjon og feilkorreksjon.

Denne master oppgaven har som mål å forbedre NUTS kommunikasjonsnettverk ved å foreslå et design og en implementering av en pålitelig transportlags protokoll. Protokollen, kalt NUTS Reliable Protocol (NRP), tilbyr segmentering av store nyttelaster, feildeteksjon, og feilkorreksjon ved å sende korrupt data på nytt. NRP ble implementert i NUTS programvarebibliotek med et enkelt grensesnitt, slik at det tilrettelegger for videre utvikling av programvare innen NUTS.

Protokollen har så blitt testet, og resultatene diskutert. Konklusjonen er et vellykket design og implementering av en transportlags protokoll for NUTS, med et forbehold om at videre testing utføres for å maksimere protokollens ytelse i NUTS nettverket.

Acknowledgements

I would first like to thank my supervisor Bjørn B. Larsen for all the help and advice you have offered throughout this master thesis. I would also like to express my gratitude to NUTS supervisor Roger Birkeland, who has been guiding me in the right direction, and offered me helpful advice and knowledge.

The NUTS team should be thanked for all their individual contributions to this thesis, and also the entertaining discussions, theories and stories which have kept the spirit of development alive through tough hours on the rooftop of NTNU.

At last I want to thank all my friends and family for their encouraging motivation, and the support they have offered during this semester.

Erlend Riis Jahren
Trondheim, 09.06.2015

List of Figures

List of Tables

Contents

CHAPTER

1

INTRODUCTION

The NTNU Test Satellite (NUTS) program aims to build and deploy a satellite following the double CubeSat standards, mainly through the work of master students and volunteers from the Norwegian University of Science and Technology[1]. The main goal of the NUTS satellite mission is to educate NTNU students through projects and master theses, while also expanding the interest of student satellites in Norway[2]. Other, more concrete, goals includes successful two-way communication with a Ground Station (GS), stabilization and satellite control using an Attitude Determination and Control System (ADCS), and to be able to capture pictures of the earth from an on-board camera, and transmit the images successfully to a Ground Station (GS).

Radio communication between a satellite and a GS is highly prone to errors, due to the altitude, elevation angle and limited transmission power of the satellite[3]. In addition, the solar induced radiation present in space environments may produce faulty behaviour in the electronic modules of the satellite, generating a demand for correction techniques and redundant fault checks[4]. Several precautions has been made by the NUTS development team to limit the impact of erroneous communication, such as error detection and Forward Error Correction (FEC) on the radio link. Still, there are no higher level communication protocols implemented to assure validity of transmissions between the sub-modules of the NUTS communication network.

This master thesis aims to resolve the issue of secure end-to-end transmission be-

tween sub-modules of the satellite and the GS by introducing a transport layer[1] protocol, offering reliability throughout the entire NUTS communication network.

1.1 Background

In 2014, an implementation of a network/transport layer communication protocol, the CubeSat Space Protocol, was added to the NUTS software repository, simplifying end-to-end transmissions between sub-modules of the satellite and the NUTS GS[6]. When tests were performed on the implementation, it was discovered that errors did not propagate through the layers of the communication structure, meaning the process responsible of sending a packet would not be able to know if the packet was actually sent and received properly. This was believed to be caused by an error with the CubeSat Space Protocol (CSP) library, but later discoveries has found this to be caused by the CSP implementation using an unreliable protocol[7, p 5]. Its functionality was thereby limited to that of a network layer protocol, illustrated in Figure 1.1. As reliability is a wanted feature in NUTS communication network, members of the team requested further development of the end-to-end protocol.

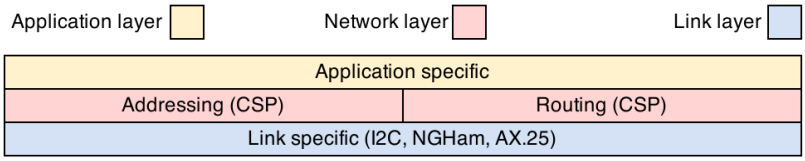


Figure 1.1: NUTS unreliable communication layer model

The CSP library includes an implementation of a Reliable Datagram Protocol (RDP), based on the RFC 908/RFC 1151 protocols. The lowest cost method of ensuring reliability in the NUTS communication system would be to use this feature of CSP, provided that it works "out of the box" as not much of the CSP documentation is publicly available. CSP's RDP feature was shortly tested in the start-up of this thesis, but was concluded not working.

1.2 Problem definition

The goal of this thesis is to design, implement and test a transport layer protocol for the NUTS CubeSat mission, featuring segmentation of payloads, error detection and recovery, and a connection oriented service. The implementation should be

[1] The Open System Interconnection (OSI) model presents a layered model for interoperability within communication networks[5, p. 73-76].

written in "C" language for the On-Board-Computer (OBC) software repository, available to all NUTS software developers. It should feature a simple Application Programming Interface (API), with easy to use method calls for transmitting payloads across the network, and well documented behavior. Figure 1.2 shows a visual presentation of the transport layer functionality this thesis aims to add to the NUTS communication structure.

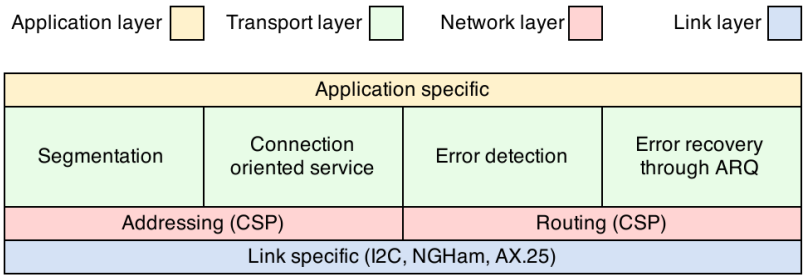


Figure 1.2: NUTS reliable communication layer model

In total, the thesis can be deconstructed into 4 individual goals;

- Design of a reliable protocol, with functionality according to the NUTS requirements
- Creating an implementation of the protocol, integrated into the software repository of the satellite
- Test the behavior and functionality of the implemented protocol
- Discuss the results of the tests and provide further improvements

CHAPTER

2

BACKGROUND THEORY

The purpose of this chapter is to introduce different theories and concepts used throughout this report. The chapter will give closer insight to the network and link layer protocols used in the NUTS communication network, and present common techniques used when designing reliable protocols.

2.1 The CubeSat Space Protocol

The CubeSat Space Protocol (CSP) is a network/transport layer protocol aimed at simplifying connectivity between distributed embedded systems[7]. The protocol was originally designed by students from Aalborg University, but was later handed over to GomSpace, a danish company specializing in cost-effective solutions for developers of nano- and cube satellites[8].

An implementation of the basic CSP functionality is released by GomSpace as a software library under GNU Lesser General Public Licence (LGPL), allowing the material to be copied and used by the public[9]. The implementation is written in GNU C, and has been ported to several different operating systems, including FreeRTOS, the OS used on the NUTS satellite. The library can be configured to use different MAC-layer drivers, authentication techniques and reliability checks, making it a viable option for the network/transport layer of the NUTS communication structure.

2.1.1 Overview

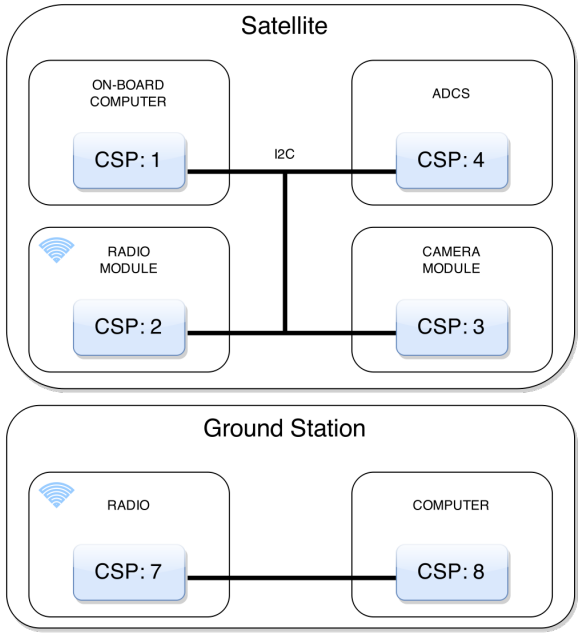


Figure 2.1: Example setup of CSP addressing

The network topology of CSP is similar to the common TCP/IP model used in the Internet[7, p 3]. Sub-modules are given addresses, each with up to 64 ports available. Each port represents an endpoint of a transmission, and can be accessed

individually. Figure 2.1 illustrates a possible setup of CSP in a satellite, similar to the setup used in the CSP user manual. Here, sub-modules within the satellite are given addresses between 0 and 6 and sub-modules of the ground station are given addresses in the range of 7 to 15. By using a routing table, all packets from the satellite to the base station will be sent through the radio module, which will pass the packet over on the radio link. When the packet is received at the base station radio module, it will be transmitted onto the base station link and received at the requested destination.

2.1.2 Header

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Priority		Source					Destination					Destination port						Source port						Reserved				HMAC	XTEA	RDP	CRC
32	Data (0-65536)																															

Figure 2.2: CSP header structure, v.1.0+

Figure 2.2 illustrates the 32 bit basic header of a CSP transmission. The first two bits of the header contains a *priority* field, giving a total of 4 different priorities for the packet[10]. The following 22 bits represents the source and destination addresses and ports of the packet. The destination fields are used in the CSP router core and determines where the content of the packet is received, depending on a preprogrammed routing table[7, p 5]. The final 8 bits of the header are flags used for other supported features by the CSP library, such as authentication(HMAC), encryption(XTEA), reliability(RDP) and checksum(CRC). Setting the flags generates extra overhead to the packet, but documentation of the added header fields are not publicly available and will therefore not be covered. Finally, the maximum payload size of one CSP packet is set to 65535 bytes.

2.1.3 I^2C driver and I^2C -to-CSP interface

Internally, the NUTS satellite uses an I^2C data bus for communication between sub-modules[11]. The bus is built into the backplane of the satellite, reaching all sub-modules while minimizing the physical footprint.

Even though GomSpace offers I^2C drivers and a CSP to I^2C , "glue", interface module, these parts of the library are not included under the LGPL licence, and is therefore not used in the NUTS satellite mission. Instead, NUTS uses an I^2C driver and a glue interface developed by NUTS member Giskeødegård in 2012[12]. His implementations were later tested and concluded functional for NUTS by an Experts in Teams (EiT) group from NTNU and Jahren[6, 13].

2.2 Automatic Repeat Request

Automatic Repeat Request (ARQ) protocols provides reliability to data transfers in communication networks, using techniques such as error detection, receiver feedback and retransmission of corrupted data[5, p. 233-235]. The protocols could be implemented at both OSI- link and transport layer, offering a method of controlling the order of packets in a stream, and that all packets are received validly. A *stream* of packets refers to multiple separate packets containing data that belongs together, which is often a result of a payload being to big to send as one packet. The event in which a payload is split into smaller chunks is referred to as *segmentation*.

Error detection is a mechanism where redundant bits are added to a packet prior to sending it, in order for the receiver to calculate if the packet has been corrupted during the transmission.

Receiver feedback is the event in which a receiver of a transmission sends feedback to the sender about a received packet. Common examples of feedback includes the Acknowledgement (ACK), where the feedback states the packet to be received without corruption, or Negative acknowledgement (NAK), where the packet was lost in transmission or corrupted at arrival. A timer is often used in addition to the receiver feedback, indicating when a packet has been completely lost.

When the sender of a packet realises that the packet did not arrive successfully at the receiver, it will retransmit the packet, and potentially other packets. The procedure is known as a *retransmission*.

Several protocols providing ARQ functionality has been published, where the trade-off amongst them is the cost of implementation and efficiency. Sections 2.2.1 to 2.2.3 presents the three most commonly used ARQ protocols in modern communication networks.

2.2.1 Stop-And-Wait

The Stop-And-Wait (SW) protocol ensures reliability and correct ordering of packets at the cost of inefficiency. A transmission is initiated by the client sending the first packet of a sequence to the server, before entering a wait state, as illustrated in Figure 2.4[5, p 235]. Here, the client waits for feedback from the server, or a timeout indicating the packet was lost. If an ACK is received, signaling the client that the packet was successfully received at the server end, it transmits the next packet of the sequence, illustrated in Figure 2.3. This procedure loops until all packets of the sequence has been sent and acknowledged, and the transmission has completed. If the client receives a NAK, or the feedback times out, indicating that the packet was corrupted or not received at the server end, it retransmits the last sent packet and remains in the wait state.

Figure 2.5 illustrates the server, replying either ACK's or NAK's depending on the content of the received packet.

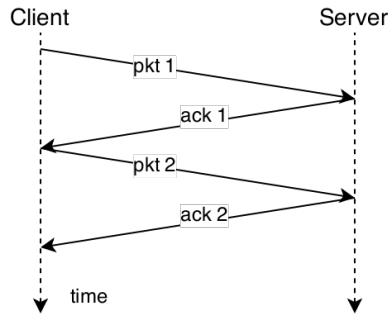


Figure 2.3: The Stop-And-Wait ARQ protocol

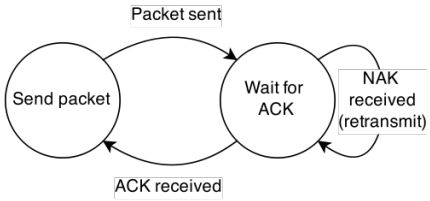


Figure 2.4: Client (sender)

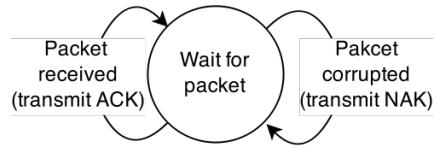


Figure 2.5: Server (receiver)

Benefits and shortcomings

Burns and Wellings claim the main benefit of the Stop-And-Wait (SW) protocol is its simplicity, as only one packet is actively awaiting acknowledgment at any time. This reduces both implementation cost and the possibility of faulty behavior to occur[5]. In addition, sequence number fields in the header can be reduced to only one bit using the Alternating Bit Protocol.

However, the simplicity comes with a drawback. Transmissions are necessarily bounded by both transmission and propagation delay, making the protocol slower than most of its alternatives.

2.2.2 Go-Back-N

A Go-Back-N (GBN) protocol utilizes a sender window of size w when transmitting a packet sequence, as illustrated in Figure 2.6 and 2.7[5, p 244-249]. The sender window, often referred to as a *sliding window*, keeps track of the packets that has been sent, but not yet acknowledged. In Go-Back-N (GBN), when the leftmost packet of the sender window in Figure 2.6 has been acknowledged, the window slides one position to the right, allowing a new packet to be sent to the receiver. If a packet gets lost during transmission, the sender will retransmit the sequence of packets within its sender window, as shown in Figure 2.8. The GBN protocol

demands a more complex implementation than the SW protocol, as it has to keep track of the packets currently being within the sender window.

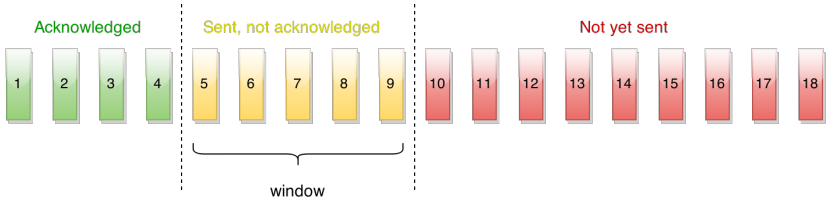


Figure 2.6: A visual representation of a sender window, $w = 5$

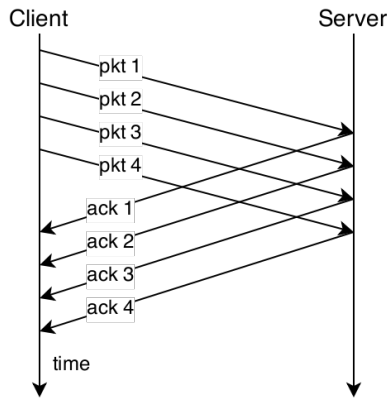


Figure 2.7: A GBN timeschedule with window size $w = 4$

Benefits and shortcomings

The GBN protocol has the benefit of transmitting several packets directly after each other, increasing channel utilization, thus making the reliable transmission faster compared to a SW protocol[5]. On the other side, as the Packet Error Rate (PER) increases, the number of unnecessary retransmissions made by the Go-Back-N protocol increases, filling the channel with superfluous information. The GBN protocol is more complex than the SW protocol, resulting in a higher implementation cost.

2.2.3 Selective Repeat

As with the GBN protocol, the Selective-Repeat (SR) protocol transmits packets within a sender window, and awaits acknowledgments for the sent packets[5, p 249-256]. However, whereas the GBN protocol will retransmit all packets succeeding

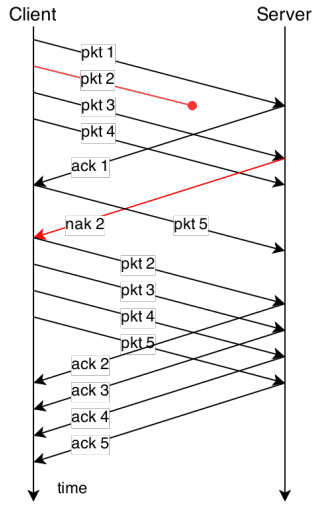


Figure 2.8: Packet-loss handled by Go-Back-N

a lost packet, the Selective-Repeat (SR) protocol only retransmits the packets that were lost, greatly reducing unnecessary transmissions and increasing channel utilization. Figure 2.9 presents an illustration of a possible implementation of SR.

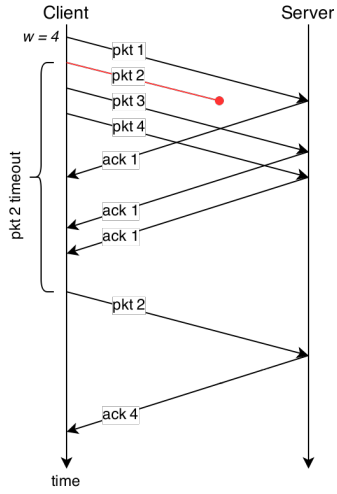


Figure 2.9: Packet-loss handled by Selective Repeat

Benefits and shortcomings

SRs improvement of the GBN protocol depends on the probability of erroneous transmissions. When the *PER* is 0, Selective-Repeat utilisation of the channel is the same as with Go-Back-N. As the *PER* increases, the SRs utilization and throughput decreases at a slower rate than with the GBN[14].

The main disadvantage with the SR protocol is its higher implementation cost, as the receiver will need to keep track of packets that are received out of order[15].

2.3 NUTS radios

The NUTS satellite is stated to carry two radios, one Ultra High Frequency (UHF) radio transmitting over the amateur satellite frequency at 437 MHz, and one Very High Frequency (VHF) radio transmitting in the frequency range of 135 to 165 MHz[16, 17]. For simplicity, the two radios will be referred to as the UHF and the VHF radios throughout this report.

Birkeland has announced that both of the radios may be responsible of transmitting CSP packets between the satellite and the ground station[2]. Although the radios use different protocols for their transmissions, they should from an end-to-end point of view be regarded as a single unit, following the abstraction principle of the OSI network layering model[5].

2.3.1 AX.25

Members of NUTS have decided that the UHF radio will be transmitting over the well documented AX.25 protocol v.2.2[16], officially published by Beech, Nielsen and Knoper in 1998[18]. The protocol is a known standard within amateur radio communication, enabling the satellite to communicate with a number of ground stations around the world. Beech et al. states that AX.25 features both connectionless and connection oriented service routines together with a segmenter, capable of splitting payloads into smaller packets when necessary. The protocol header also includes a Cyclic Redundancy Check (CRC) of 2 bytes used for error detection.

Flag	Address	Control	PID	Info(data)	FCS	Flag
01111110	112/224 bits	8/16 bits	8 bits	N*8 bits	16 bits	01111110

Figure 2.10: AX.25 Information packet frame

Payload efficiency

The official publication document of the AX.25 protocol reveals the default information field of the packet frame presented in Figure 2.10 to be bounded to 256 bytes, and that the information packet header itself has a maximum of 35 bytes[18]. With the CSP header v.1.0+ of 4 bytes we can calculate the efficiency of the protocols on the radio link. The proposed NUTS Authentication Protocol (NAP) of 22 bytes by Marius Münch[19], which was previously intended to be used on the up-link, is left out, as it will be difficult to implement and is by Birkeland considered to be an unlikely part of the radio link protocols[2]. As a result, the efficient payload calculations are the same for both up-link and down-link.

$$pl_{eff} = \frac{B_{pay}}{B_{tot}} \tag{2.1}$$

Using Equation 2.1, where pl_{eff} is the payload efficiency, B_{pay} is the payload in bytes, and B_{tot} is total bytes in the packet, we obtain the theoretical payload efficiencies in Table 2.1.

Table 2.1: Payload efficiency with the AX.25 protocol

Case	$\mathbf{B_{pay}}$	$\mathbf{B_{tot}}$	$\mathbf{pl_{eff}}$
best	252	291	$\approx 87.0\%$
worst	1	40	$\approx 2.5\%$

2.3.2 NGHam

The NGHam protocol, developed by NUTS member Jon Petter Skagmo, is implemented to the NUTS satellite with the intent to improve aspects of the AX.25[20]. NGHam is stated to have higher throughput, better spectral efficiency and higher robustness than AX.25, much due to its use of FEC. The protocol structure is illustrated in Figure 2.11

Payload efficiency

Table 2.2: Payload efficiency with the NGHam protocol

Case	$\mathbf{B_{pay}}$	**Padding (B)**	$\mathbf{B_{tot}}$	$\mathbf{pl_{eff}}$
best	216	0	266	$\approx 81.2\%$
worst	1	27	78	$\approx 1.3\%$

NGHam has the possibility of 7 packet sizes, the biggest giving a payload of 220 bytes, with an additional 46 Bytes of header fields. By including the CSP in the

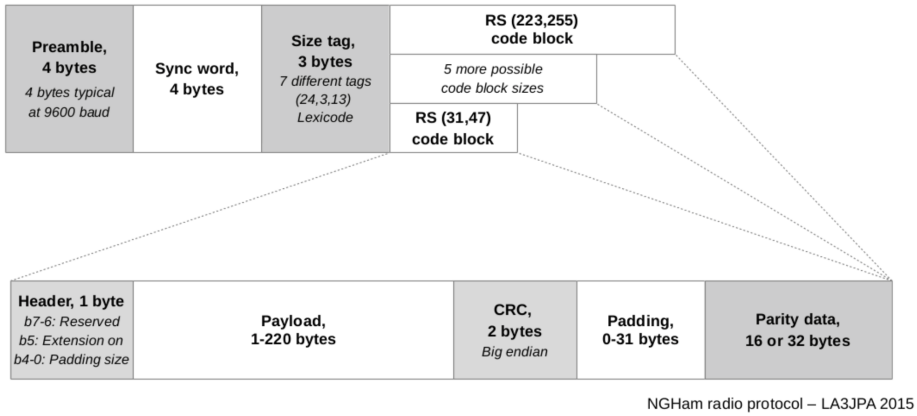


Figure 2.11: NGHam frame, illustration by Skagmo[20]

payload of the link layer, we obtain the theoretical payload efficiencies presented in Table 2.2.

2.3.3 Link capacity

$$R_{eff} = pl_{eff} * \underbrace{R}_{\text{bit rate}} \tag{2.2}$$

The radio links are the weakest link of the NUTS communication network, making it the bottleneck of any Ground Station to satellite transmission. Table 2.3 presents the theoretical best case (b-c) and worst case (w-c) effective throughput R_{eff} of the two radios, calculated using Equation 2.2. However, it is important to note that the actual throughput may be significantly lower due to the noise present in real-world scenarios.

Table 2.3: Theoretical effective throughput of the VHF and UHF radios

Freq. band	**Protocol**	**R**	**estimated $\mathbf{R_{eff}}$**
UHF	AX.25	$9600bit/s$	b-c: $\approx 8350bit/s$ w-c: $\approx 250bit/s$
VHF	NGHam	$9600bit/s$	b-c: $\approx 7800bit/s$ w-c: $\approx 125bit/s$

CHAPTER

3

PROTOCOL REQUIREMENTS

A protocols design resembles the intention it was made for. In this chapter we will look at the functionality and features of a reliable transport protocol that is useful for NUTS, and investigate how this can be implemented in a protocol with efficient utilization of the communication network.

3.1 Reliability

3.1.1 Reliable delivery

Assured delivery of payload in the NUTS network is impractical, as the satellite at any point can become unable to communicate with the GS over a substantial amount of time. Hence, the reliable service of the transport layer protocol should be limited to notifying the sender if a transmission was successful or not. If the sender is notified that the transmission was a success, it must be assured beforehand that the payload was received successfully and without corruption at its final destination. If the transmission was not successful, the sender should be notified with an error, thus making the transport service reliable.

3.1.2 Error detection

The NUTS networks most unreliable links, the radio links, are protected by error detection and correction techniques through the protocols described in Section 2.3. However, Lantz II claims space environments are exposed for solar radiation, causing soft errors such as bit-flips in electronic hardware[4]. To prevent corrupt information from being received and interpreted as valid, the end-to-end transport layer protocol should include an error detecting technique.

3.2 NUTS link budget

In *Antenna systems for NUTS*, Marholm presents estimated download link capacities for the different satellite altitudes shown in Table 3.1.

Table 3.1: Marholms estimations of the NUTS down-link capacity

Altitude	**Min. elevation angle**	**Average capacity**
350km	21°	581kB/day
500km	28°	627kB/day
650km	34°	636kB/day

The orbital altitude is dependent on the specific rocket-mission responsible of launching the NUTS CubeSat into space, which has yet to be decided as of June 2015[2]. Elevation angle is a time variant variable, as the satellite passes the GS with different angles for each orbit, resulting in various transmission capacities for each pass. The report *Image Compression* by Bakkebø et al. uses simulations of the satellites potential orbits to calculate the duration the satellite will have contact with the GS as it passes[21]. Their calculations are presented in Table 3.2.

Table 3.2: Estimated minimal duration of satellite to GS connectivity for 90% of the passes by Bakkebø et al.

Altitude	**Min. duration (90% of passes)**
350km	85 seconds
500km	93 seconds
650km	103 seconds

Using the information in Table 3.2, and the *b-c* bit-rates calculated in Section 2.3, we obtain the efficient radio link capacities for each pass, presented in Table 3.3.

Table 3.3: Theoretical minimum radio-link capacity in 90% of satellite passes

Protocol	**Altitude**	**Duration**	$\mathbf{R_{eff}}$**(*b-c*)**	**Capacity**
AX.25	350km	85s	8 350bit/s	$\approx 88.7kB$
AX.25	500km	93s	8 350bit/s	$\approx 97.1kB$
AX.25	650km	103s	8 350bit/s	$\approx 107.5kB$
NGHam	350km	85s	7 800bit/s	$\approx 82.9kB$
NGHam	500km	93s	7 800bit/s	$\approx 90.7kB$
NGHam	650km	103s	7 800bit/s	$\approx 100.4kB$

Bakkebø et al. further claims that around 50% of the passes will have at least 200 seconds available for transmission, giving the link capacities presented in Table 3.4.

Table 3.4: Theoretical minimum radio-link capacity in 50% of satellite passes

Protocol	**Altitude**	**Duration**	$\mathbf{R_{eff}}$**bc**	**Capacity**
AX.25	500km	≈200s	8 350bit/s	$\approx 208.8kB$
NGHam	500km	≈200s	7 800bit/s	$\approx 195.0kB$

The capacities indicates the lower limit a transport layer protocol should be able to transmit reliably within one stream, while the maximum limit should be set so it does not add restraints to the transmissions.

3.3 NUTS payloads

Knowledge of the type, and especially the size, of the payload which will be transmitted through the NUTS communication network is of importance when designing a proprietary protocol. NUTS supervisor Roger Birkeland has stated that the biggest payload in terms of bytes will be the picture sent from the satellite to the GS[2]. The report *Digital processing system for a Cubesat camera* by Andreas Bertheussen states the raw picture of 2592x1952 pixels captured by the camera

expected to be used on the NUTS satellite to be ≈ 7.6MB[22]. Using Equation 3.1 it is calculated that the transmission of the picture would take approximately 2 hours, assuming an utility factor, u, of 100%. Even under these perfect conditions the transmission time largely exceeds the approximate of 200 seconds available at the longest satellite passes. As a result, Bakkebø et al. provides information about how the picture can be reduced to any requested size, using image compressor software such as the JPEG2000. This will necessarily produce a reduction in the quality of the image, but is unavoidable in order to transmit the image to earth over the slow radio-links described in Section 2.3.

$$T_{trans} = \frac{B_{tot}}{R_{eff} * u} \tag{3.1}$$

In Section 2.3 it was mentioned that the maximum payload size of the radio packets were 256 bytes for the UHF radio and 220 bytes for the VHF radio. The radio packets are in other words significantly lower than the maximum packet size, 65535 bytes, of a CSP packet. To avoid being dependent on link layer segmentation, this should be done in the transport layer, with packets not exceeding the lowest maximum payload of the two radio protocols. Hence, maximum packet size for the transport layer protocol must be 216 bytes, as the CSP header of 4 bytes is included in the radio link payload.

CHAPTER

4

DESIGN SPECIFICATIONS

There are many publicly defined protocols that could be used for as the transport layer protocol for NUTS, but they most often support more features than whats necessary for the requested functionality of the NTNU Test Satellite. In the attempt to minimize protocol overhead, and thus increasing channel utility, a protocol has been designed for the sole purpose of the NUTS CubeSat mission. The protocol has been given the name **NUTS Reliable Protocol (NRP)**.

4.1 NRP v.1.0 - overview

4.1.1 Header

The protocol header consists of 5 flags and 3 fields ranging over a total of 7 bytes, as illustrated in Figure 4.1.

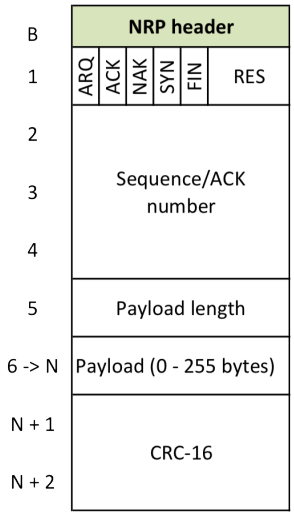


Figure 4.1: NRP header v.1.0

The structure of the protocol header is the same for every packet type in NRP, but the flags and field values change depending on the content and the type of packet being sent. The maximum payload size of a NRP packet is limited to 255 bytes, due to the limited payload size in the underlying link layer protocols AX.25 and NGHam, explained in Section 2.3.

4.1.2 Features

The following features are included in the NRP protocol;

- Segmentation and encapsulation of payloads
- Connection oriented service
- Client to server, half-duplex communication
- Error detection
- Go-Back-N ARQ functionality

4.1.3 Reliable data transfer

Figure 4.2 illustrates a timing diagram of a successful NRP transmission. Data is transmitted in a client to server direction, and acknowledgements are returned by the server as feedback.

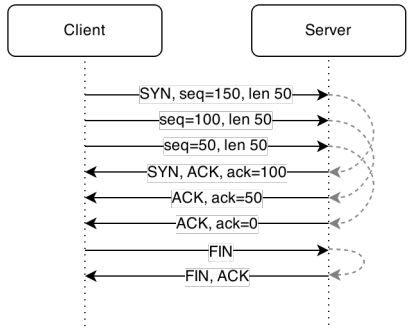


Figure 4.2: NRP transmission

The following four steps explains the NRP reliable transmission procedure in a simplified manner:

1. Segmentation and encapsulation of payload
2. Connection initiation
3. Transmit data using Go-Back-N ARQ
4. Connection termination

Step 1: Segmentation and encapsulation

Payloads larger than the maximum payload of one NRP packet are split into separate parts, each encapsulated by a NRP header and ready to be transmitted. Each header contains a sequence number indicating the order of the packets, and a payload length field indicating the size of the packet content. Sequence numbering is further explained in Section 4.4.

Step 2: Connection initiation

Next, a connection is requested by the client and acknowledged by the server. A reference to the connection details is stored in both the client and server throughout the transmission. A detailed explanation of NRP connections is given in Section 4.3.

Step 3: Transmit data

Data transmission begins immediately after requesting the connection, sending packets within a sender window, using the Go-Back-N ARQ protocol explained in Section 2.2. Transmission of data runs continuously until all packets has been received by the server.

Step 4: Connection termination

When all packets are sent from the client and acknowledged by the server, the client issues a connection termination request, which is acknowledged by the server. The transmission is successfully completed.

4.2 Packet types

The first byte of the NRP header contains 5 flag fields and a 3 bit reserved field. The combination of set[1] flags determines the type of the packet, and how it should be processed. The reserved field is currently unused except for its byte aligning purposes, making it available header space in a future revision of the protocol.

In total there are 9 different defined packet types, each explained briefly below.

SYN packet
Indicated by setting the ARQ and the SYN flags. It initiates a connection request between a client and a server, while also containing the first payload of the stream. This packet type is used in the first transmitted packet of a stream.

Direction: client to server

SYN/ACK packet
Indicated by setting the ARQ, the SYN and the ACK flags. It acknowledges a connection request sent from a client, letting the client know that the connection was created at the server.

Direction: server to client

SYN/NAK packet
Indicated by setting the ARQ, the SYN and the NAK flags. It is a negative acknowledgement to a connection request, letting the client know that the server was not able to create the connection, and that the transmission failed.

Direction: server to client

Reliable data packet
Is indicated by setting only the ARQ flag. The packet informs the receiver that

[1] A flag is said to be "set" if its value is '1'

the content of the packet is sent using the reliable protocol, thus requiring specific actions based on the header content.

Direction: client to server

ACK packet
Indicated by setting the ARQ and the ACK flags. The packet type is used to acknowledge received payload.

Direction: server to client

NAK packet
Indicated by setting the ARQ and the NAK flags. It requests the client of a connection to retransmit all packets from a specified sequence number.

Direction: server to client

FIN flag
Indicated by setting the ARQ and the FIN flags. It is sent as the last packet from client to server, requesting the server to release its connection, thus terminating the transmission.

Direction: client to server

FIN/ACK packet
Indicated by setting the ARQ, the FIN and the ACK flags. It acknowledges a connection termination request sent from a client, letting the client know that the connection was terminated at the server, and that the transmission was a success.

Direction: server to client

Unreliable data packet
The reliable features of NRP can be turned off by not setting the ARQ flag. This will result in the receiver not taking the header fields into account when processing a packet, thus providing unreliable, connectionless service. The header fields will still be present in the packet, which is useful because the NUTS implementation of the lower level CubeSat Space Protocol does not contain a field to specify the transport layer protocol.

Direction: client to server

4.3 Connection

A connection oriented transmission begins with a handshake between the two endpoints of the transmission, prior to sending any data[5]. During the handshake, each endpoint exchange information needed to transfer the following data, according to a specified transmission protocol. The connection is then preserved during

the entire time the transmission is active, and terminated once all data is successfully sent, or the transmission has failed. A connection oriented service facilitates several features important for a reliable transmission, such as error detection, acknowledgements, retransmission of lost packets, and correct order of data.

In NRP, the handshake is integrated with the first payload of a packet stream, thus slightly separating itself from the definition of a connected oriented service, while still facilitating the features for a reliable transmission. The separate handshake is generally useful to avoid sending unnecessary data through a larger network to a endpoint that may not exist, but is found unnecessary for the more predictable, and less congested, network of the NUTS satellite and GS.

The following subsections presents the design of the NRP connection handling. Figures that explains NRP behavior not specified in this section is available in Appendix A.2.

4.3.1 Setting up the connection

All reliable transmissions are based on a set connection between two endpoints, as illustrated in Figure 4.3. A client populates a connection structure from the connection pool with information about the transmission before sending a SYN packet to the server. The sequence number of the SYN packet is set to the size of the total payload, in bytes, making the server able to allocate memory for the content of the stream. The payload field is filled with the first part of the segmented payload, thus integrating the connection handshake and the first part of the payload within the same packet.

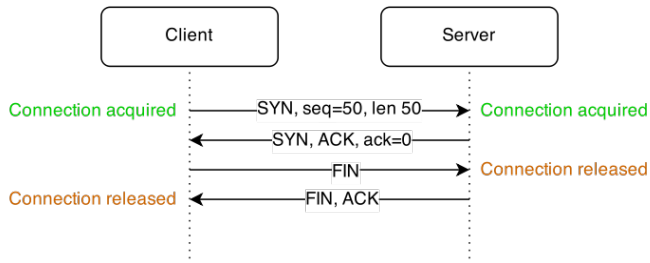


Figure 4.3: Setup and termination of a connection

A server receiving a SYN packet will request the population of an available connection structure and allocate a buffer for the payload of the stream. If successful, the payload of the SYN packet will be stored in the buffer, and a SYN/ACK packet will be sent in return. The SYN/ACK packet lets the client know the connection request was successful, and that the server is awaiting the next packet of the stream.

The client will retransmit its SYN packet if it does not receive a reply within a

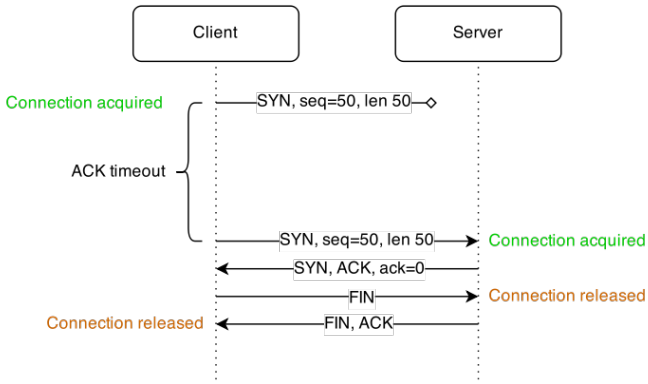


Figure 4.4: Packet loss during connection

set time, as illustrated in Figure 4.4. If the server does not reply within three retransmissions, the sending is canceled and the sending task is notified through an error code.

4.3.2 Refusing a connection attempt

Figure 4.5 illustrates the occurrence of a refused connection. When a server receives a SYN packet, but is not able to create a connection, it can respond back to the client with a SYN/NAK packet, refusing the connection attempt. The SYN/NAK packet carries an 8-bit error code in the payload field, explaining the reason for refusing the connection. Note that a SYN/NAK packet is a special case in the NRP design, as it is the only event in which payload is transmitted in a server to client direction.

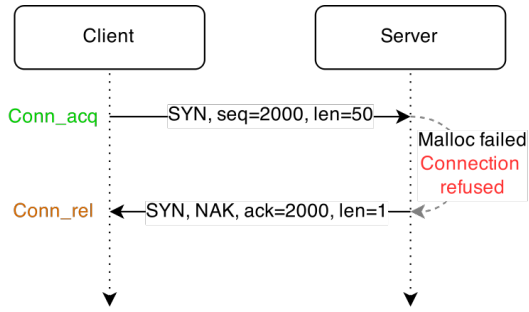


Figure 4.5: A connection is refused by the server

4.3.3 Closing the connection

Figure 4.3 also illustrates the termination of an active connection. When a FIN packet is received at the server, it releases its connection structure, and notifies the task listening to the specific port that a message has been received, with a pointer to the full payload of the stream. The server proceeds to acknowledge the FIN packet with a FIN/ACK packet, which makes the client release its connection structure. When the connections at both the server and the client has been released, the transmission is completed and and the sender task is notified whether or not the transmission was successful.

4.3.4 Connection timeout

The NRP protocol is designed to transmit information between a satellite and a GS. Hence, at any point, part of the communication link between two endpoints may disappear, due to the satellite being out of reach from the GS. If contact is lost before connection initiation, the connection attempt will result in a triple timeout. However, if the connection is already created between the end points, the loss of contact must be handled differently. After a triple ACK timeout, the client releases its connection and an error is returned. On server side, there are no indications of the transmission having failed, as it simply just stops receiving the expected packets from the client. Therefore, a connection timeout is used to avoid the connection to remain active in the server when contact is lost.

$$t_{conn} \geq (N_{retrans} + 1) * t_{ACK_{max}} \tag{4.1}$$

The connection timeout is given by Equation 4.1, where $t_{ACK_{max}}$ denotes the maximum timeout of acknowledgements in the network, and $N_{retrans}$ denotes the maximum number a packet is allowed to be retransmitted. The connection timer is necessarily reset each time a new packet is received, as this is an indication of the connection still being active. By exceeding the maximum client side timeout, a triple $t_{ACK_{max}}$, unwanted connection timeouts are prevented.

4.4 Sequence and acknowledgment number

Correct ordering and delivery of packets in a stream is handled using a sequence number, where each number represents one byte of the total stream. As illustrated in Figure 4.6, the initial sequence number of a transmission is set to the total payload of the stream in bytes, informing the server of the buffer size it has to allocate to receive the message. For each packet sent, the number is decreased by the size of the payload in that packet, supplying the receiving node with an indicator of the sequence number in the next expected packet.

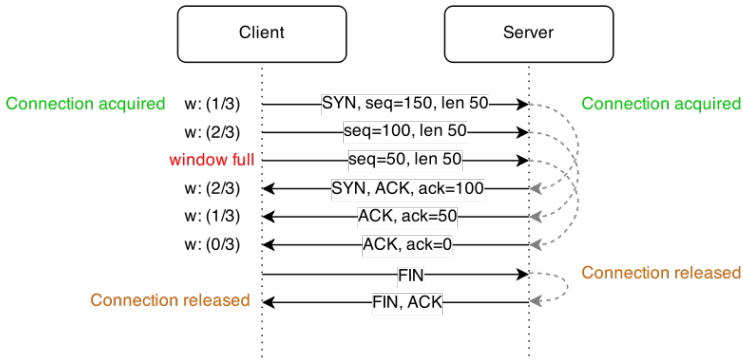


Figure 4.6: NRP streaming 150 bytes of payload over three packets

As packets are received at the server, their sequence numbers are checked against the next expected sequence number stored in the connection reference of the transmission. If the numbers match, an ACK packet is transmitted back to the client, and the sequence number field is used as an acknowledgement number field instead, informing the client that the server has received all bytes up to, but not included, the given ACK number. When the client receives an acknowledgement with ACK number 0, it knows that the entire payload of the stream has been transmitted successfully, and it closes the connection with a handshake.

Note that the three byte sequence field limits the total payload of the stream to 2^{24} bytes, or about 16.7MB, which is well within the requirements presented in Section 3.3

4.5 Cyclic Redundancy Check

A 16 bit Cyclic Redundancy Check (CRC) is computed and added after the payload to assure the packets validity. CRC is an error detection method commonly used in computer network protocols, popular because of its ability to detect multiple error scenarios at a low computational cost[23]. Kurose and Ross states that proper implementations of CRC are able to detect any b_e number of bit errors while $b_e \leq r + 1$, where r is the number of redundant bits added to the packet[5, p. 469]. It follows that for any bit error bursts of $b_e > r + 1$, the probability of detecting the packet to be erroneous becomes $P_{e\_detected} = 1 - (\frac{1}{2})^r$

The NUTS project has already implemented link layer error detection and correction for the radio links, through the AX.25 and NGHam protocols, yielding the 16 bit CRC to be a sufficient error detection technique for the NUTS Reliable Protocol.

4.6 Efficiency

4.6.1 Payload

In Section 2.3 we determined the best and worst case payload efficiencies of packets transmitted across the radio link, without a transport layer protocol. Tables 4.1 and 4.2 presents the updated payload efficiencies for the UHF and VHF radios when the redundant 7 byte NRP header is contained in the packet, and acknowledgements of the data is required. Note that the maximum payload of a NRP packet is set to 209 bytes for both AX.25 and NGHam, as the radio protocol is unknown from the transport layers point of view. The maximum packet payload, 220 bytes, of NGHam is reached with 209 bytes NRP payload, 7 bytes NRP header, and 4 bytes of CSP header.

Table 4.1: Theoretical payload efficiency with NRP using the AX.25 protocol

Case	$\mathbf{B_{pay}}$	$\mathbf{B_{tot}}$	$\mathbf{B_{ack_{pkt}}}$	$\mathbf{pl_{eff}}$
best	209	255	46	$\approx 69.4\%$
worst	1	47	46	$\approx 1.1\%$

Table 4.2: Theoretical payload efficiency with NRP using the NGHam protocol

Case	$\mathbf{B_{pay}}$	**Padding (B)**	$\mathbf{B_{tot}}$	$\mathbf{B_{ack_{pkt}}}$	$\mathbf{pl_{eff}}$
best	209	0	266	78	$\approx 60.8\%$
worst	1	27	78	78	$\approx 0.6\%$

This further allows estimations of the NUTS radios R_{eff} using the NRP protocol with a max packet payload size, pl_{max}, of 209 bytes, which is presented in Table 4.3.

Table 4.3: Theoretical effective throughput of the VHF and UHF radios with NRP

Protocol	**NRP** $\mathbf{pl_{max}}$	**R**	**estimated** $\mathbf{R_{eff}}$
AX.25 (UHF)	209B	$9600bit/s$	*b-c*: $\approx 6650bit/s$
NGHam (VHF)	209B	$9600bit/s$	*b-c*: $\approx 5850bit/s$

4.6.2 Transmission

Using the data in Table 4.3 and Equation 2.2 we are able to estimate the time it will take to send a picture through the slowest and most important NUTS communication link, the radio. Figure 4.7 illustrates a time comparison of the two radio link protocols when sending a stream of 30KB, using the NRP protocol. The figure shows that there is only a small difference between the UHF and the VHF radios when transmitting payloads using the NUTS Reliable Protocol.

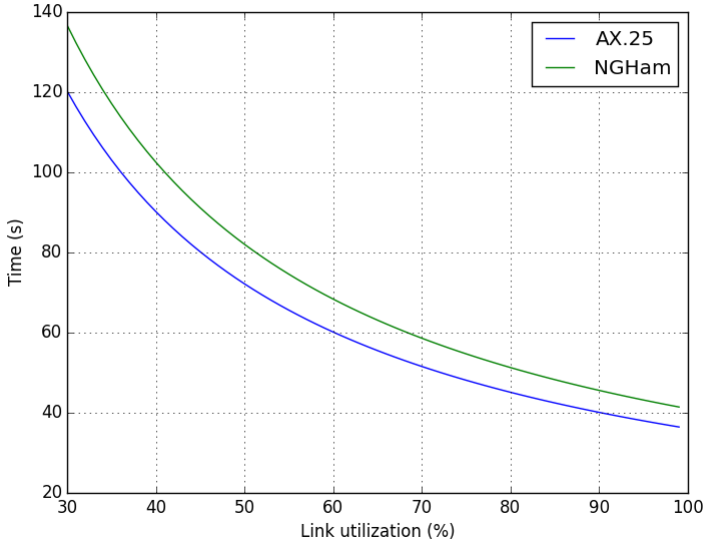


Figure 4.7: Theoretical time comparison of the UHF(AX.25) and VHF(NGHam) radios when sending 30KB using NRP

CHAPTER

5

IMPLEMENTATION

This chapter presents an implementation of NRP v.1.0 in the software repository of the NUTS CubeSat. The implementation is i written in C programming language, and builds on already implemented versions of FreeRTOS and CSP. All source code is available at NUTSs BitBucket git repository, and in the archive file attached to this report.

5.1 Overview

NRP is implemented on top of the already implemented CSP network layer protocol in a sub-modules internal communication structure, as illustrated in Figures 5.1 and 5.2. One FreeRTOS task, the NRP task, is responsible of receiving CSP packets from its underlying layer, where each packet is processed according to its NRP packet type. The task provides reliability by sending acknowledgements for received packets, and stores the data of a stream into an allocated buffer.

Multiple FreeRTOS tasks is used to listen to specific ports. When a stream has completed, its payload is sent to the task listening to the destination port.

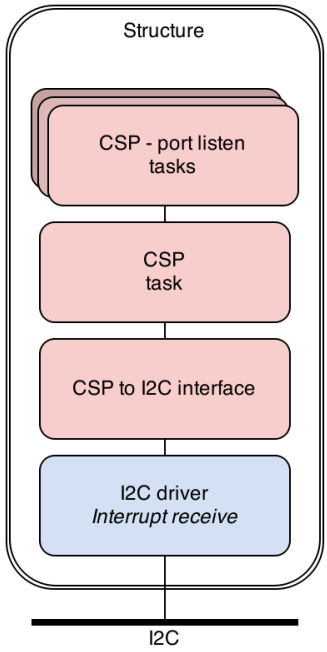


Figure 5.1: Communication structure before NRP implementation

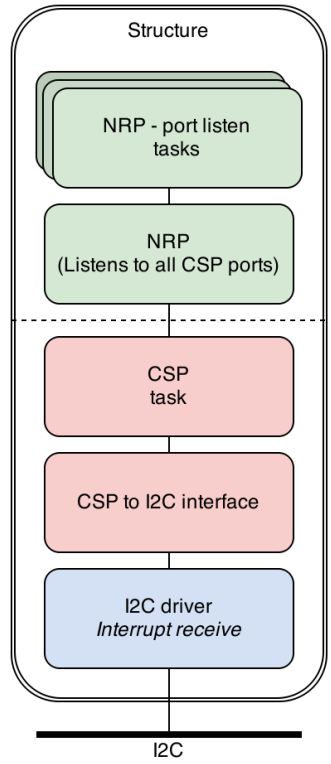


Figure 5.2: Communication structure after NRP implementation

5.1.1 FreeRTOS

FreeRTOS is an open source, cross platform, real time operating system (OS) widely used for micro-controllers and microprocessors[24], already implemented to the

NUTS software repository. The OS features creation of parallel tasks, semaphores and queues, which is used extensively in the protocol implementation.

5.2 Segmentation and encapsulation

For payloads longer than the maximum NRP packet payload size $pl_{pkt_{max}}$, where $pl_{pkt_{max}} \leq 255$, segmentation is necessary. Figure 5.3 illustrates the segmentation and encapsulation method of the NRP protocol. Here, the payload is divided into separate parts, each part being smaller or equal to $pl_{pkt_{max}}$, before setting header fields such as the sequence number and flags. Next, the NRP header and packet payload is used together with the CSP header to calculate a CRC of 16 bits. Finally, the packet content is encapsulated by the NRP header and the CRC is appended to the end of the payload.

In the NUTS implementation of NRP, $pl_{pkt_{max}}$ is set to a maximum of 209 bytes, as this is the maximum payload size available when using the NGHam protocol.

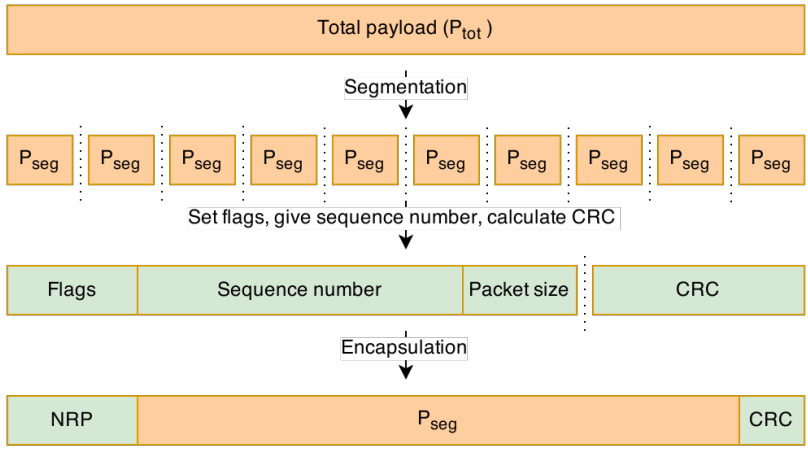


Figure 5.3: NRP segmentation and encapsulation

Segmentation is not necessary for payloads shorter than $pl_{pkt_{max}}$, thus leaving only the steps of filling header fields and encapsulate the packet.

5.3 Connection

The NRP connection pool consists of an array of *struct* data structures, storing information about each end point of a connection. The most important values of a NRP connection structure is presented in Table 5.1.

Table 5.1: Important values in a NRP connection structure

Variable	Description
client_socket_info	A struct containing the address and port of the client
server_socket_info	A struct containing the address and port of the server
window_avail	The number of unsent packets within the sender window, only used by the client
packets_recv	Server - Number of payload packets received. Client - Number of acknowledgements received
init_seq_num	The initial sequence number, the payload of the stream in bytes.
next_expected_seq_num	The expected sequence or acknowledgement number in the next packet to be received
init_timestamp	Connection initiation timestamp
last_updated_timestamp	Timestamp from last received packet
*recv_packet_t	pointer to the first byte of the payload

5.3.1 Mutual exclusion

Several of the variables in each connection *struct* is read and changed by multiple concurrently running FreeRTOS tasks, a concept described by Burns and Wellings as **shared variables**[25, p 137-139]. With shared variables, a problem arises in the event where a sender task and a receiving task performs overlapping memory instructions on the same connection variable at the same time, causing the stored values to become corrupt. To overcome this unwanted behavior, we introduce **critical sections**. These are sections where only one task can operate at the same time, thereby protecting the shared variables from corruption. In FreeRTOS, and many other OS's, critical sections can be implemented with a technique known as **mutual exclusion (mutex)**[26]. The FreeRTOS website describes the mutex functionality as follows;

> *"When used for mutual exclusion the mutex acts like a token that is used to guard a resource. When a task wishes to access the resource it must first obtain ('take') the token. When it has finished with the resource it must 'give' the token back - allowing other tasks the opportunity to access the same resource."*
>
> – FreeRTOS website[26]

In the NUTS NRP implementation, the entire connection pool is guarded by one single mutex. Before altering any information stored in one of the connection *structs*, the running task must obtain the connection pool mutex successfully. Once the mutex is taken, the task can change both the values in the variables of each

struct in the pool array, and their order in the array. The procedure of taking the mutex is illustrated in Figure 5.4

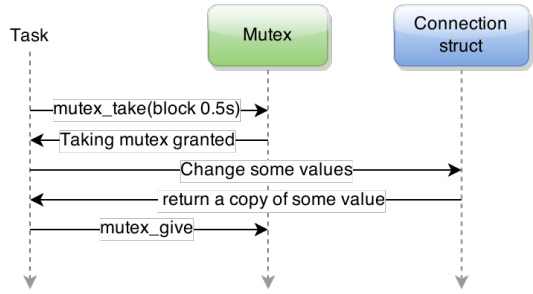


Figure 5.4: Obtaining the connection pool mutex to alter the connection values

5.3.2 Active first principle

The connection pool array is implemented as "active first", meaning that active connections are always indexed before inactive connections. All interactions with the connections involves an iteration through the connection pool array until the desired connection is found. By preserving the active first principle, less iterations are needed for each interaction, resulting in faster computation.

5.3.3 Activation and termination

Upon connection activation, the connection pool array is iterated until an inactive connection is found. The connection is then activated if the connection pool is not full and no other reasons to reject the new connection is found. Terminating a connection will re-index the other active connections to fulfill the principle of active first.

5.3.4 Multiple simultaneous connections

Figure 5.5 illustrates how a sub-module is implemented to have multiple active connections with other sub-modules simultaneously. When a client-to-server packet is received, its source address and port number is extracted and compared to the *client_socket_info* field of the active connections in the connection pool. Consecutively, a server-to-client packet will compare the source fields to the *server_socket_info* of the connection pool. If the port and address of an active connection matches the received packet, its information is used to process the packet. The implementation is thus necessarily limited to only one simultaneous connection between two specific ports in the network at any time.

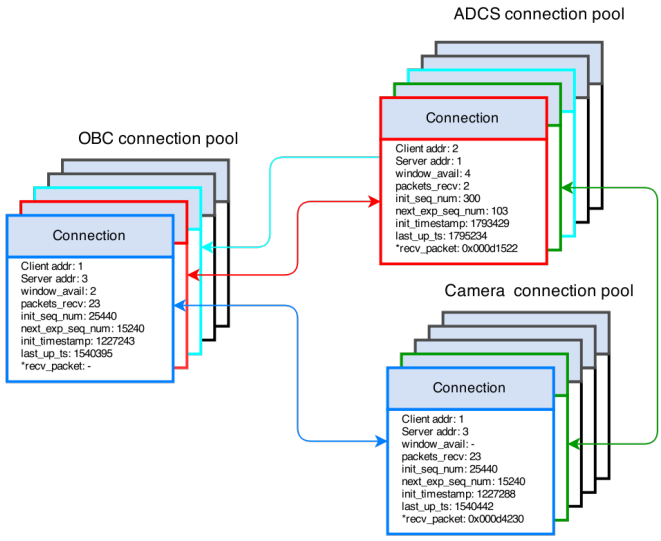


Figure 5.5: Multiple active connections within the connection pools of NUTS sub-modules.

5.4 CRC

The NRP implementation uses a publicly available CRC-16 implementation by Michael Barr to check the packets for bit-errors[27]. Barrs implementation features fast individual CRC computing due to its use of a pre-calculated look-up table, using 256 bytes of memory. The look-up table is recalculated every 10 minutes to resolve eventual corruption caused by cosmic radiation.

5.5 Data transfer

Reliable transfer of data is implemented abstractly for the user of NRP, through a simple API.

5.5.1 Sending

Once NRP is initialized, it can be used to send reliably with the *nrp_send* function shown in Listing 5.1. The parameters in the *nrp_send* function specifies the connection details, such as the address and ports of the source and destination, the payload length and a pointer to the data, and a priority. The priority is directly transferred to the CSP library, where buffered packets with higher priority are sent before lower prioritised packets.

```
int8_t nrp_send(const module_socket_t *dest,
                const module_socket_t *source,
                const uint8_t *payload,
                uint32_t payload_length,
                uint8_t priority);
```

Listing 5.1: NRP send function

A call to the *nrp_send* function initiates the procedure illustrated in Figure 5.6, which continues until the transmission is either finished with success, or stopped and an error code is returned. The *CSP dummy header* that is generated in the procedure is used to include the CSP header in the calculation of the CRC.

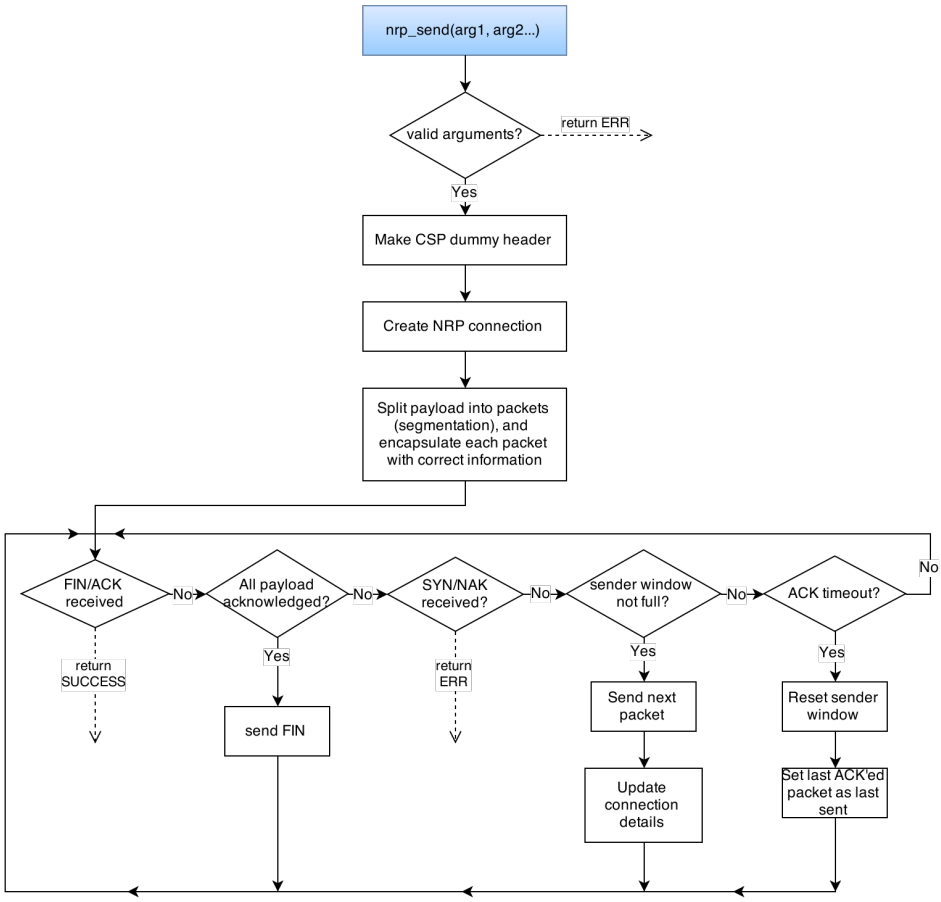


Figure 5.6: NRP send reliable data procedure

5.5.2 Receiving

A single FreeRTOS task is continously listening to all CSP ports, thus receiving all packets from the underlying network layer. When a packet is received, it is first checked for its validity by calculating the expected CRC. The packet is assumed valid if the calculated CRC value matches with the CRC appended to the packet. An invalid packet is discarded, while valid packets are processed depending on the value of the flag fields. Figure 5.7 illustrates the general receive procedure, while flag specific behavior is presented with figures in Appendix A.1.

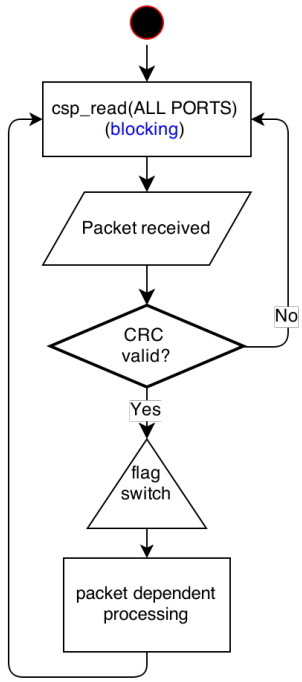


Figure 5.7: NRP receive procedure

5.5.3 Port listeners

Content is stored in the preallocated memory by subtracting the received packets sequence number from the initial sequence number of the transmission, as illustrated in Figure 5.8. When the the entire payload of a stream has been received, the receive task generates a *struct* containing all necessary information about the transmission and a pointer to the allocated data buffer. The *struct* is then added to a queue, to be received by the task listening to the port the stream was sent to. The queueing of fully received payload streams is illustrated in Appendix A.1.6.

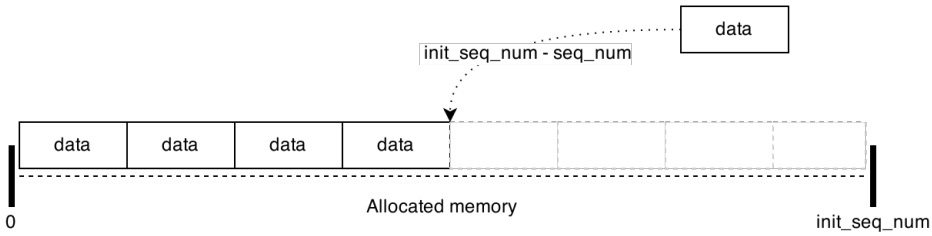


Figure 5.8: Storing data to reproduce original payload

5.6 Dynamic timeout

The Round-Trip-Time (RTT) is calculated by the client in the beginning of every transmission. It starts a timer right before sending the SYN packet to the server, and ends it immediately after receiving the SYN/ACK reply. This RTT then becomes the reference point for the ACK timeouts for all the following packets in the stream. ACK timeouts are simply obtained by a multiplication of the RTT, as shown in Equation 5.1, where t_{ACK} must be within the boundaries of $t_{ACK_{min}}$ and $t_{ACK_{max}}$. The triple RTT timeout provides an efficient and reasonable timeout for the ACKs, while remaining simple and affordable to implement.

$$t_{ACK} = RTT * 3 \tag{5.1}$$

Figure 5.9 illustrates the functionality of the t_{ACK}.

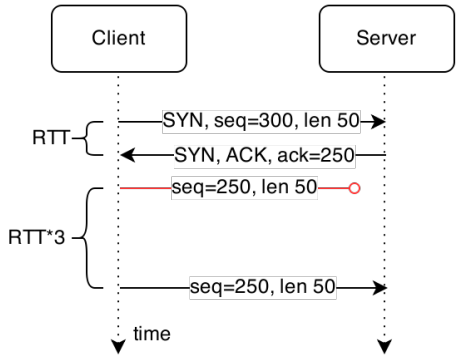


Figure 5.9: Calculating a timeout for each individual stream

CHAPTER

6

TESTING

To verify that the behavior of the NRP implementation is according to the protocol design described in Chapter 4, a series of tests has been created and executed. All tests are performed over a test setup with Inter-Intergrated Circuit (I^2C)-links. In short terms, the functionality tested for is listed below

- Segmentation and reproduction of original payload
- Robustness with CRC-16 error detection
- Go-Back-N ARQ functionality
- Receiving data streams from multiple senders simultaneously
- Effect of Go-Back-N ARQ window sizes
- Effect of dynamic ACK timeouts

NRPs underlying CSP protocol has been concluded to work for the NUTS mission by Jahren in the report *Implementing CSP over I2C for the new repository on the NTNU Test Satellite*, and will therefore not be specifically tested in this report[6].

6.1 Test setup

6.1.1 Hardware

The hardware setup is the same as in Jahrens test of CSP in 2014, using three Atmel UC3-A3256 development boards to represent different sub modules of the satellite, and an I^2C link as the communication channel[6]. An overview of the setup is illustrated in Figure 6.1. Here, the OBC, ADCS and camera sub-modules are represented, although their functionality is limited to NRP communication.

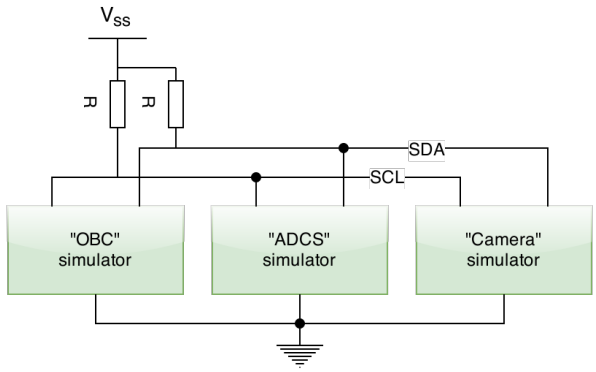


Figure 6.1: Schematics of the test setup with three Atmels UC3-A3 development boards connected through an I^2C link.

Each of the Atmel UC3-A3 development boards are connected to a computer through *UART to RS232* and *RS232 to Universal Serial Bus (USB)* adapters which enables serial communication through a **Command Line Interface (CLI)** implemented for the NUTS project by Thomas Nornes in 2014[28]. The CLI is ran by a separate FreeRTOS task and is used to tests and to setup the modules with different CSP addresses and ports.

6.2 Software

Data from the tests are collected by transmitting values from the sub-modules to a file on a computer through the serial communication. The data is then read and presented by various Python scripts using *matplotlib*, a python library for presentation of graphs and plots[29].

6.3 Heap limitations

The available internal Static Random Access Memory (SRAM) of the Atmel UC3-A3s is limited to 64KB[30]. Almost all of the memory is already used in the FreeRTOS, CSP and NRP implementations, leaving only the leftover memory available to be used in testing. Solving this was attempted in this thesis by trying to move the heap into an external SRAM, but it was never completed successfully. However, Birkeland has stated that an extension of the memory is expected to be implemented for some of the sub-modules in a later revision of the satellite, as this is needed in order to transmit large sized pictures[2].

Limited memory makes it impossible to test payloads in the size range of the pictures that will be sent between the satellite and the Ground Station. Instead, maximum NRP packet sizes has been adjusted considerably down in the following tests to simulate the many individual packets of a large payload stream.

6.4 Bit-error function

In order to test the NRP implementation for its reliable features, it is necessary to have a function that can simulate the bit-flip errors present in space and radio transmissions. The created function randomly inserts a given number of bit-flips into a NRP packet, and has a set probability of being called right before the packet is transmitted. The CSP header is not included in the function, as this is processed in the network layer. The source code for the function is presented in Appendix C

6.5 Test 1: Segmentation and encapsulation

The test will prove that the NRP implementation is able to take payloads longer than the maximum payload of a NRP packet, split the payload into smaller chunks of data, encapsulate each chunk with an appropriate NRP header, before sending each packet individually over the network. On the receiving end, the original full payload will be reproduced by putting the chunks together in correct order. The segmentation and encapsulation is considered working successfully if it can segment data into

6.5.1 Procedure

A payload of 500 bytes is sent with a maximum NRP paylaod size, pl_{pkt}, of 1 to 10 bytes. Thus, in order to send the full payload it needs to be segmented into N chunks of data, where N is given by Equation 6.1. The number of chunks

Table 6.1: Stream specifications for the "Reliability with Go-Back-N" test

Specification	**Value**
Streams	$20 * 1$
pl_{tot}	500 bytes
pl_{pkt}	1 - 10 bytes

is printed out before sending the stream, indicating if the payload was correctly segmented.

$$N \geq \frac{pl_{tot}}{pl_{pkt}} \tag{6.1}$$

Each chunk of data is then encapsulated by a NRP header with values according to the protocol, creating a packet ready for transmission. The bit-wise representation of the packet is then printed to the CLI, where the values are checked for correctness.

6.5.2 Expectations

The results are expected to show payloads being segmented into a specific number of packets depending on the pl_{pkt} size, as shown in Table 6.2. It is further expected that every first packet of the stream will have set the SYN flag, indicating a connection request. The sequence number of the first packet should be 500 bytes, and decrease with pl_{pkt} for each packet. The last packet of the stream should not contain a FIN flag, as this is generated as a separate packet.

Table 6.2: Expected results for the "Segmentation and encapsulation" test

$\mathbf{pl_{pkt}}$	**N packets**	**Sequence numbers**
1	500	(500, 499,... ...,1)
2	250	(500, 498,... ...,2)
3	167	(500, 497,... ...,2)
↓	↓	↓
10	50	(500, 490,... ...,10)

6.5.3 Result

The segmentation and encapsulation test was successful, as all segmentations and encapsulations resulted in the expected values of Table 6.2. Every streams first packet had set the SYN flag, while the other packets had only the ARQ flag, indicating that the stream was using the NRP reliable functionality.

6.6 Test 2: Detection of bit-errors

The NRP CRC-16 generator is a published implementation by Michael Barr from 2000, and is considered to work properly[27]. However, it is important to make sure that it works as expected in the NRP implementation, in order to discard this source of error from other tests.

6.6.1 Procedure

Table 6.3: Stream specifications for the "Detection of bit-errors" test

Specification	**Value**
Streams	$9 * 5000$
pl_{tot}	500 bytes
pl_{pkt}	5, 48, 102 bytes
w	1
PER	0%, 5%, 20%

The test is performed by sending a total of 45 000 streams of 500 bytes with the different pl_{pkt} and PER shown in Table 6.3. The window size is set to 1 to avoid unnecessary retransmissions. The sending node, the ADCS, counts the number of packets it inserts errors in, and prints the result to the CLI at the end of each 5000 streams. The receiving node, the OBC, counts the number of packets it throws away due to CRC calculation not matching the CRC value appended to the packet. The OBC prints out the number of invalid packets it has detected after receiving all of the streams.

The test is also ran with different pl_{pkt} sizes to verify that packet sizes should be insignificant when detecting errors. The pl_{pkt} sizes is chosen randomly do give a range of payload sizes, as testing for all packet sizes between 1 and 255 is impractical.

6.6.2 Expectations

The I^2C link is very stable, and is thus not considered a source of error itself. This means that all detected errors should be caused by artificially inserted bit-flips by the ADCS, thus making the packet error count equal on each side of the transmission.

6.6.3 Result

The test was successful, with all corrupted packets being detected and discarded. There was no evidence of corrupt packets being accepted or valid packets being

discarded.

6.7 Test 3: Reliability with Go-Back-N

The test is designed to prove that the GBN ARQ protocol is working correctly, and that error codes are returned for unsuccessful transmissions. The GBN functionality is responsible of resending packets that has been introduced to errors, making sure that all packets of a stream is received successfully in the server of a connection.

6.7.1 Procedure

Table 6.4: Stream specifications for the "Reliability with Go-Back-N" test

Specification	**Value**
pl_{tot}	500 bytes
pl_{pkt}	5 bytes
w	5
PER	0% & 2%
t_{ACK}	$3 * RTT_{SYN}$

The camera sub-module takes 500 specific bytes of payload and segments and encapsulates it to a stream of 100 packets containing 5 bytes of data each, before sending the stream to the OBC using NRPs reliable functionality. The camera measures the time it takes for the entire stream to be transmitted successfully, and prints the result through the CLI. An unsuccessful stream transmission, which should cause an error to be returned to the sender task of the camera, will be printed with time 0. The OBC prints out error messages to the CLI, indicating if a SYN/NAK was sent, or if a successfully received stream contained corrupt data.

Each stream of 500 bytes is sent 20 000 times with 0% *PER*, and another 20 000 times with 2% *PER*, where each corrupt packet contains between 1 and 16 bit-flips.

6.7.2 Expectations

The result of the test is expected to show a large number of streams being sent successfully with Go-Back-N functionality, even with the packet error probability of 2%. The streams with 0% *PER* should have a close to constant stream transmission time, while the streams with 2% *PER* are expected to be spread over a larger time interval, due to retransmission of corrupted packets. The test with 2% *PER*

is also expected to show spikes ranging well above the average time, due to the probabilistic event where SYN or SYN/ACK packets are corrupted. These packets are sent before the dynamic acknowledgement timeout, t_{ACK} is set, resulting in a maximum ACK timeout of 3000ms before retransmission. The probability P_e of a packet or its acknowledgement being corrupted is 4%, giving the estimated occurrences, O_{est}, shown in Table 6.5. For the stream failure rate we estimate that each packet and ACK transaction has a probability of slightly more than 2%, as the packets can be acknowledged by the ACK belonging to the next packet of the window, thus almost removing the ACK-loss error source.

Table 6.5: Estimated occurrences of stream failures and spikes

Event in stream	$\mathbf{P_e}$	$\mathbf{O_{est}}$
Stream failure	$2\% < P_{e_{est}} < 2.5\%$	$16 < O_{est} < 32$
SYN or SYN/ACK loss once	4%	800
SYN or SYN/ACK loss twice	0.16%	32

For the test to be successful, there should be no occurrences of payload being received with errors after the entire stream is completed, and all unsuccessful streams should result in an error code being returned to the sending task.

6.7.3 Result

The test ran for a total of approximately 12 hours, without a single occurrence of received payload being corrupted after stream completion. All erroneous stream transmissions were registered by the sending module, and the test thus concludes the GBN functionality to be successfully implemented.

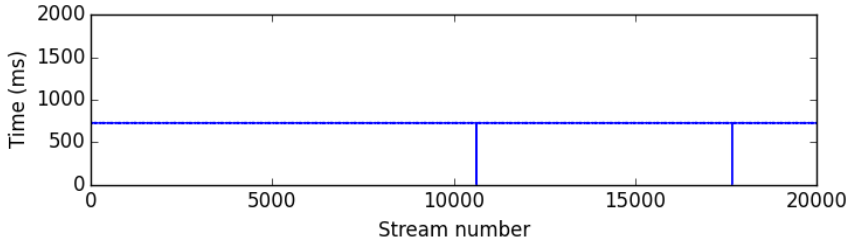


Figure 6.2: Measured stream transmission time for 0% *PER* with GBN

Figure 6.2 displays the time it took for each of the 20 000 streams of 100 packets to reliably be transmitted when no bit-errors were inserted. The figure shows that the time is fairly constant for all streams, except for two cases where the transmission failed, indicated by a time of 0. The OBC printed out two occurrences of SYN/-NAKs being sent, which explains the two transmission failures. The SYN/NAKs were caused by the OBC not being able to allocate 500 bytes for the payload.

The reason for the allocation errors is unknown, but a realistic scenario would be that it was caused by a delayed deallocation of a previous stream, leaving too little memory available for a new stream. This event could have happened, as there was not enough memory available to allocate multiple 500 byte payloads at the same time, and because the payload memory is allocated and deallocated by two different FreeRTOS tasks.

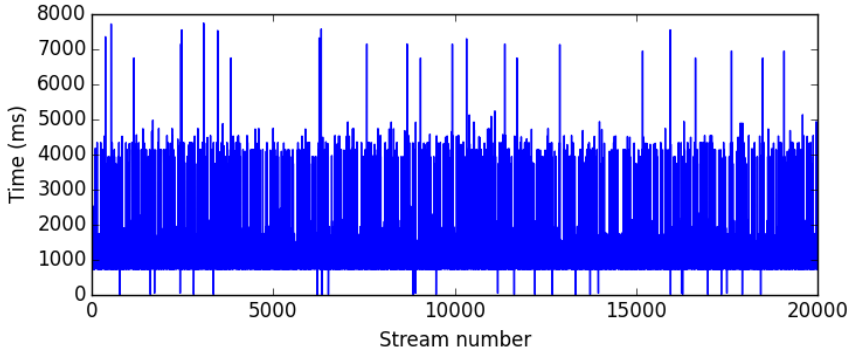


Figure 6.3: Measured stream transmission time for 2% *PER* with GBN

Figure 6.3 displays the same timing diagram for the 20 000 streams with 2% *PER*. The graph shows that these streams were highly time variant, ranging from 700ms to just below 8000ms.

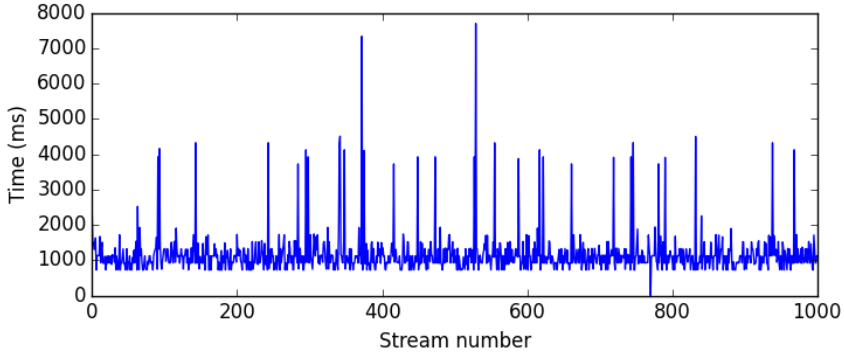


Figure 6.4: Closeup of the first 1000 streams in 2% *per* test

Figure 6.4 is a closeup of the first 1000 streams in Figure 6.3. The graph shows that most streams are transmitted within a range of 700 to 2000 milliseconds, while there are occasional occurrences of spikes and failures. The spikes can be divided into two groups, where the first group average around 4000ms and the second average around 7000ms. This is a clear indication of the expected 3000ms and

6000ms ACK timeouts for one and two SYN retransmissions, respectively.

Table 6.6: Measured occurrences of spikes and SYN retransmissions with 2% *PER*

Event	$\mathbf{L_{lower}}$ (ms)	$\mathbf{L_{upper}}$ (ms)	$\mathbf{O_{est}}$	$\mathbf{O_m}$
Fails	0	199	16-32	27
Normal	200	2999	19 150	19 206
One SYN retrans.	3000	5999	800	743
Two SYN retrans.	6000	∞	32	24

In Table 6.6, each measured stream time is placed within upper and lower limits, L, to compare the measured results with the estimated occurrences in Table 6.5. The measurements shows that the GBN functionality works as expected, with all measured occurrences, O_m, being within reasonable deviation of the estimations.

6.8 Test 4: Simultaneous streams

The test is designed to check whether or not the NRP implementation can handle multiple connections simultaneously, as stated in Section 5.3.4. To test this, it would have been best to transmit one very big payload stream from two sub-modules to one simultaneously, but this was not possible due to the heap limitations. Instead, many smaller streams are transmitted directly after each other from the two sub-modules, and the time of arrival for each stream is measured in the receiving module.

6.8.1 Procedure

Table 6.7: Specifications for the "Simultaneous streams" test

Specification	**Value**
streams	$2 * 1000$
pl_{stream}	500 bytes
pl_{pkt}	5 bytes
w	5
PER	0%
t_{ACK}	$3 * RTT_{SYN}$

The ADCS and camera sub-modules spams the OBC with streams having the specifications presented in Table 6.7, until they each have successfully sent 1000 streams. An unsuccessful stream transmission will result in the sender being inactive for slightly longer than the set connection timeout, before retransmitting the entire stream. Both the ADCS and the camera will be transmitting with the same CSP priority.

6.8.2 Expectation

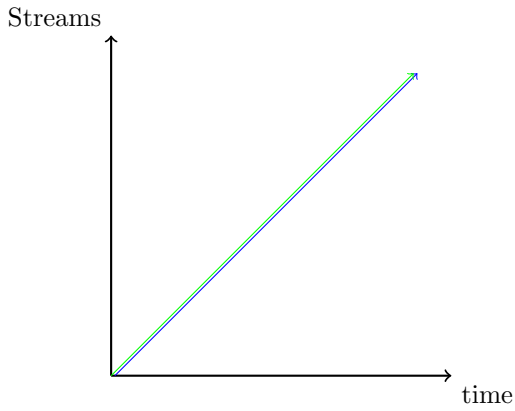


Figure 6.5: Expected behavior of simultaneous stream test

The receiver, the OBC, is expected to be able to receive equally from both the ADCS and camera, with a result close to the illustration in Figure 6.8.2. Here, the green and blue arrows represents the number of successful transmissions from the ADCS and camera respectively compared to the time it was received at the OBC.

6.8.3 Result

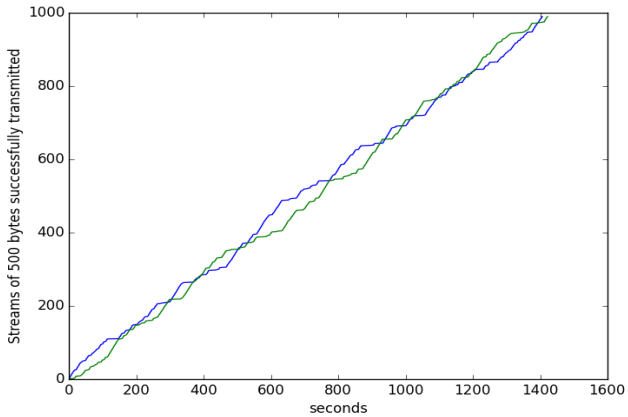


Figure 6.6: Test with multiple streams to same receiver

Figure 6.6 presents the time schedule of successfully arrived streams in the OBC, where the green and blue line represents the ADCS and camera respectively. The

graph shows results visually close to what was expected, with each sub-module managing to transmit all the 1000 streams within approximately the same time period. However, the close-up illustration in Figure 6.7 shows one of several examples in the data where one sub-module is on hold or unable to transmit an entire stream for a longer period of time, while the other sub-module is transmitting its streams successfully. This is most likely a result of one of the sub-modules failing a stream, and then being held back until the connection timeout has ran out on the receiver side.

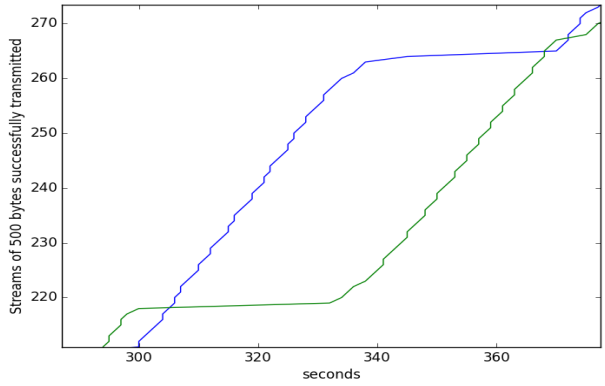


Figure 6.7: Closeup of test with multiple streams to same receiver

No explanation for the failing streams has been found, but it is assumed that it is caused by the OBC module getting too many packets simultaneously, thus not being able to process both streams. Once the ADCS has stopped transmitting due to a stream failure, the camera is able to transmit perfectly, and vica versa. The problem does not prevent the implementation from working reliably, but lowers the throughput due to tasks waiting for connection timeouts.

6.9 Test 5: Effect of window size

In Section 2.2.2 it was stated that a Go-Back-N protocol provided faster service than a simpler stop-and-wait ARQ implementation, due to a sliding windows higher utilization of the communication channel. To prove that this is the case for the NRP implementation presented in this report, we need to perform speed tests with different window sizes and compare the results.

It is important to note that a Go-Back-N protocol with window size, $w = 1$, is in fact a Stop-And-Wait protocol, as each packet sent will have to be acknowledged before transmitting the next packet of the stream.

6.9.1 Procedure

Table 6.8: Specifications for the "Effect of window size" test

Specification	Value
Streams	$36 * 2000$
pl_{stream}	500 bytes
pl_{pkt}	5 bytes
w	1,2,5,10
PER	0, 0.5, 1, 2, 5, 7.5, 10, 15, 20, 30 (%)
t_{ACK}	$3 * RTT_{SYN}$

2000 streams of 500 bytes segmented into 100 packets are transmitted from the ADCS to the OBC for each of the window sizes and *PER*s in Table 6.8, thus giving a total of 36 tests. The average time, number of failed stream transmissions and measured *PER* is printed through the CLI at the end of each test, and the results are saved.

6.9.2 Expectations

The test is expected to show improved stream transmission times when using GBN functionality, as opposed to SW, for all variants of packet error ratios.

For the stream failure rates, it is expected higher failure rates for lower values of w. This is expected due to ACK packets ability to acknowledge previous data packets in addition to the packet it was intended for, thus reducing the probability of a corrupted ACK causing retransmissions.

6.9.3 Result

Figure 6.8 shows the measured average speed of each stream, for all the different window sizes. The graph displays a clear improvement in speed when $w \geq 2$, with best results for $w = 5$. As the *PER* increases, the time it takes to transmit a stream increases exponentially for all $w > 2$ and proportionally for $w = 1$. A table with all the data from Figure 6.8 is given in Appendix D.

The stream fail rate is illustrated in Figure 6.9, showing close to expected results. The failure rate is highest for $w = 1$, almost equal for $w = 5$ and $w = 10$, while $w = 2$ has the lowest rate.

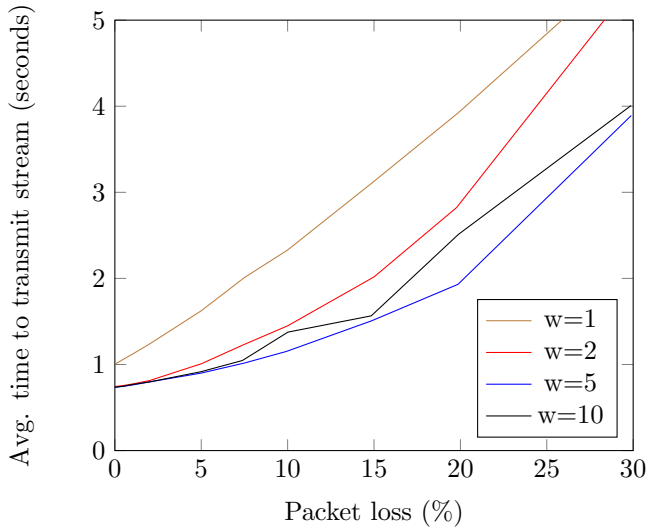


Figure 6.8: Graphical representation of stream transmission time for different *PERs*

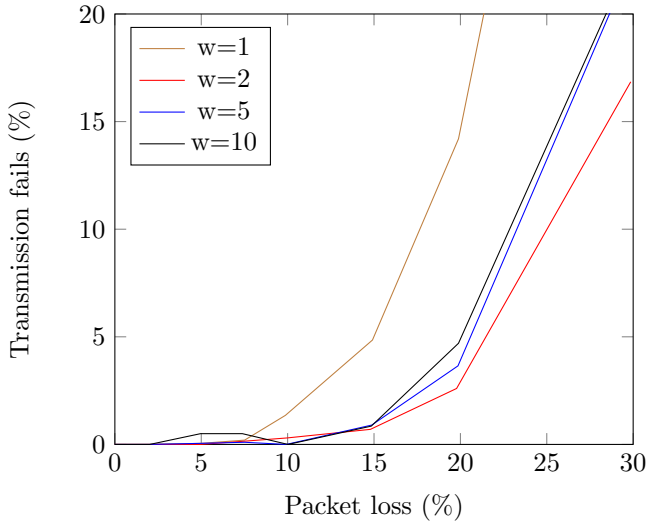


Figure 6.9: Graphical representation of the registered transmission fails

6.10 Test 6: Effect of Dynamic timeout

The Figure 6.8 from the previous test showed that stream transmission times decreased using GBN compared to SW. In this test, we want to investigate the importance of a dynamic timeout. As the results of the previous test showed transmission

times with a dynamic timeout, we will in this test use a static timeout, and look at the differences between the results.

6.10.1 Procedure

Table 6.9: Specifications for the "Effect of dynamic timeout" test

Specification	**Value**
Streams	$21 * 1000$
pl_{stream}	500 bytes
pl_{pkt}	5 bytes
w	1,5,10
PER	0%,0.5%,1%,2%,5%,10%,20%
t_{ACK}	1000ms

The procedure is almost identical to the "Effect of window size" test, with the exception of only testing for w's of 1,5 and 10, and with only 1000 streams for each individual test. The static timeout, t_{ACK}, is set to 1000milliseconds (ms), as shown in Table 6.9.

6.10.2 Expectations

The test is expected to show slower stream transmission times for all window sizes compared to the results from the "Effects of window size" test.

6.10.3 Result

The results of the test is presented in Figure 6.10. As expected, each of the measured streams has a much higher stream transmission time than for the dynamic timeout results in Figure 6.8 of the previous test. At 10% PER, the stream time has close to doubled for $w = 5$ and $w = 10$, and increased by a factor of 5 for $w = 1$.

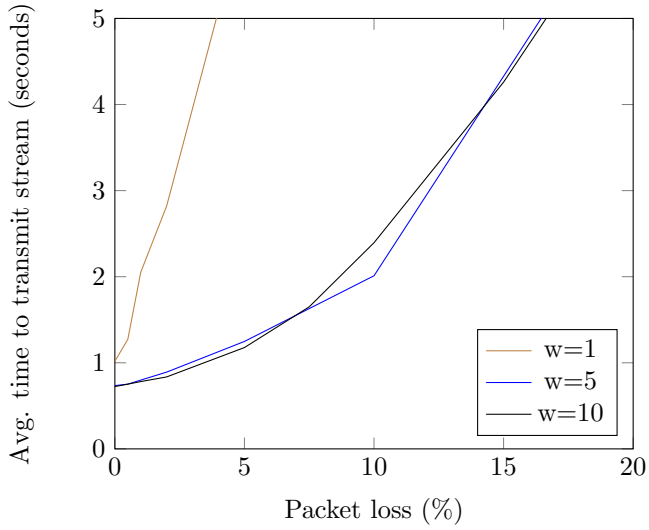


Figure 6.10: Graphical representation of the speed differences with static timeout at 1000ms

CHAPTER

7

DISCUSSION

The purpose of this chapter is to discuss the results from Chapter 6, determine if the protocol has been successfully implemented, and to investigate the importance of further development and testing of NRP.

7.1 Performance of the NRP implementation

The tests performed in Chapter 6 proved in many ways the NRP implementation to be successful. It was shown that the segmentation part of the implementation managed to split large payloads into smaller chunks of data, and that each chunk was encapsulated by correct header fields. Further it was proved that the implementation of CRC was working, with packets containing bit-errors being discarded, and valid packets being accepted. The GBN functionality test showed that the reliability feature of NRP was working as expected, with high probability of transferring payloads successfully over the I^2C link with a relatively high Packet Error Rate of 2%. All unsuccessful transmissions were reported and feedback was given to the task responsible of sending the data, thus offering reliable service. The effects of using a sliding window and dynamic timeout was also tested, and proved to significantly improve the transmission time and throughput of the protocol.

Only the test of multiple streams showed results that were different from the expectations, with a server node that was not capable of receiving payload from two client nodes simultaneously. This might have been the consequence of the server not being able to process all the packets quick enough, resulting in packets being thrown out of the receiving buffer prior to being processed, although this was never proven. It is therefore advised to further test the protocol implementation, to find and potentially correct the error source of this problem.

7.2 Real scenario testing

The NRP implementation presented in this report was only tested on an I^2C link, which will be the internal physical layer of the NUTS satellite. The results showed that the implementation works for this purpose, but should be tested further in order to conclude it as successful. In addition, NRP is also designed specifically for the UHF and VHF radio links between the satellite and the GS. Testing of the protocol over the radio links was considered impractical in this thesis, because neither the NUTS radios nor the CSP routing functionality were properly implemented at the time of the testing.

The biggest difference when testing over the radio links compared to I^2C , is the larger transmission delays due to the slow bit rate. It is unknown how this will affect the transmission time and the necessary window sizes needed to provide good utilization over the network, which is important in order to send larger payloads successfully between the satellite and the ground station.

One big concern when testing the NRP protocol over a network consisting of several links, with routing mechanisms to get the packets to their destination, is the possibility of reordering of packets. Packets arriving out of order in the destination node would result in many packets being discarded, as the NRP protocol implementation does not include a receiving window, thus only accepting ordered packets. This

could, however, be solved by upgrading NRP to include SR behavior, which should be possible without redesigning the entire protocol and implementation.

7.3 Go-Back-N versus Selective Repeat

The Go-Back-N protocol was chosen as the ARQ protocol for NRP due to its better performance compared to Stop-And-Wait, and its implementation simplicity compared to the Selective-Repeat. However, Section 4.6 illustrated the importance of a good utilization when sending bigger payloads reliably over the short time span available as the satellite passes the ground station. The achieved throughput is generally difficult to estimate as it depends on numerous parameters, such as *PER*, transmission and propagation delays, window size of the ARQ protocol and more. Still, it is a good idea to investigate and discuss the possible utilization improvement obtained by using a SR protocol instead of the GBN protocol.

Yang Qin and Lie-Liang Yang presents a calculation of utilization differences between the three ARQ protocols described in this thesis, as shown in Figure 7.1[14]. Their analysis is not based on satellite to GS radio links, but does provide general information about the differences in ARQ utilization, which is sufficient for the purpose of this discussion.

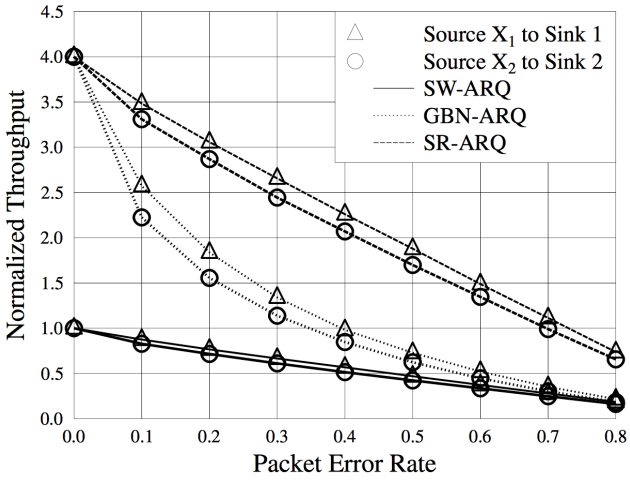


Figure 7.1: ARQ comparison by Yang Qin and Lie-Liang Yang

Qin and Liangs analysis shows a steep loss in utilization with GBN even for lower *PER* values, which indicates that NUTS would benefit from a SR ARQ protocol. However, the calculations should be compared with the expected *PER* in the NUTS network in order to present a reasonable expected improvement. If the expected *PER* is of size 10^{-2}, then the improvement factor becomes very small compared to

the implementation cost, while the utilization improvement factor with a *PER* of $\geq 10^{-1}$ would be significantly larger.

Marholm states the NUTS Bit Error Rate (*BER*) of the radio link to be 10^{-5}, giving a *PER* of around $2 * 10^{-2}$ for full sized packets[3]. The low error rate, together with the error detection and correction techniques in the radio link protocols, would result in only a small improvement in utilization by swithing to a SR protocol. However, the impact of bit-flips in the satellites hardware due to radiation is unknown, and might cause a much worse *PER* in end to end transmissions. The improvement factor and necessity of a revision of NRP to include SR should thus be further investigated before a conclusion can be made.

Upgrading NRP to include Selective-Repeat behavior should be possible without redesigning the entire protocol. In order to do this, it must be implemented a mechanism responsible of remembering the packets that has already been received. A mechanism for storing the data correctly ordered in the receive buffer has already been implemented, as described in Section 5.5.

7.4 Acknowledgement timeout calculation

The dynamic acknowledgement timeout described in Section 5.6 is implemented to avoid unnecessary large timeouts over parts of the NUTS communication network where the transmission and propagation delays are lower than for the radio, such as over the internal I^2C link in the satellite. The calculation of the dynamic timeout is currently based on a simple multiplication of the Round-Trip-Time, which works for its purpose. However, there are possibilities of improving the dynamic acknowledgement timeout and the maximum ACK timeout.

The maximum ACK timeout in the implementation presented in this thesis is set unreasonable high, at 3000ms, as the end-to-end RTT of a satellite to GS transmission is unknown. The value could easily be redefined in the implementation with better calculations of the theoretical RTT, thus providing faster transmissions.

It is also possible to improve the dynamic ACK timeout calculation by introducing more advanced techniques, e.g. where ACK-timeouts are based on an estimated RTT calculated from multiple measurements of the actual RTT. The improvement factor realised by such an implementation is unknown, but should be considered in a revision of the NRP protocol.

CHAPTER

8

FUTURE WORK

8.1 Further testing

As mentioned in Section 7.1, further testing of the NRP implementation is advised, as it has only been tested on an I^2C link with limited heap available. In addition, one of the tests performed in this report showed an anomaly from the expected behavior, which should be further investigated. At last, a complete test, including radio links and picture sized payloads, should be performed and verified before concluding to use the NRP protocol and implementation for the NUTS CubeSat mission.

8.2 Implementing NRP for the GS

This thesis has presented an implementation of NRP for FreeRTOS, the OS used on the sub-modules of the NUTS satellite. However, the transport layer protocol must also be implemented for the OS on the Ground Station in order to receive NRP packets from the satellite.

8.3 Implementation of CSP routing

The NUTS network consists of multiple modules and sub-modules divided by both radio links and I^2C links. Transmitting a packet from one of the sub-modules on the satellite to the GS would require a routing mechanism as all packets must be sent through the radio module. CSP offers a routing functionality which should be implemented to the software of each sub-module in the NUTS network.

8.4 Extending the heap

The internal memory of the Atmel UC3-A3256 microprocessors used on the OBC of the satellite is too small for the transmission of a picture, as most of the memory is already used by the FreeRTOS, CSP, and NRP implementations. Hence, the issue of moving the heap of the microprocessor into an external SRAM must be completed.

CHAPTER

9

CONCLUSION

This thesis has presented a proposed design and implementation of a reliable transport layer protocol, specifically intended for the NUTS CubeSat mission. The protocol, conveniently named the NUTS Reliable Protocol (NRP), features segmentation of payloads, error detection through cyclic redundancy checks, and reliability through Go-Back-N ARQ functionality.

NRP was designed based on the requirements, estimations and assumptions that has previously been presented for the NUTS CubeSat mission, and estimations that were presented in this thesis. A short study estimated that NRP was theoretically able to transmit reliably with close to 70% utilization of the radio links, which are the bottlenecks of the NUTS communication network.

The protocol was then implemented for the NUTS software repository, receiving data through the underlying network layer protocol, CSP. The implementation aimed at providing reliable service according to the designed protocol, using error detection and retransmissions of corrupt data.

The implementation was tested for its functionality, which showed results indicating that the implementation was mostly successful. Payload larger than the maximum NRP packet size was segmented into appropriate sized chunks, and encapsulated correctly according to the protocol design. Further it was shown that the implemented error detection successfully detected packets with inserted bit-errors. Tests of the Go-Back-N functionality proved that invalid packets were discarded, and retransmitted to assure that all the payload was received in correct order. One test showed a possible defect with the implementation, making it unstable when trying

to receive payload from multiple streams simultaneously, but the source of error was never found. The registered defect did not prevent the implementation form working reliably, but could have negative effects for the throughput.

The tests were only performed over an I^2C link, simulating the internal physical link of the NUTS satellite. However, the implementation should also be tested with routing mechanisms over multiple links simultaneously, including the NUTS radio link, as this is closer to the scenario the protocol will be used for.

The requested reliability feature of the transport layer protocol in NUTS was concentrated to the assurance of a client knowing when a payload has been fully received by a server, and when it has not. The tests have shown that this reliability is in fact present in this implementation of NRP, along with good results for throughput and utilization. The design and implementation of a reliable transport layer protocol for the NUTS CubeSat mission is thus concluded successful, but further testing and development is advised to maximise its performance.

REFERENCES

[1] NUTS. NTNU Test Satellite. [Online]. http://nuts.cubesat.no (accessed 27.03.2015).

[2] Roger Birkeland. NUTS coordinator and supervisor. Personal conversation, . E-mail: roger.birkeland@iet.ntnu.no.

[3] Sigvald Marholm. Antenna systems for NUTS. Master's thesis, NTNU, Trondheim, 2012.

[4] Leon Lantz II. Soft errors induced by alpha particles. IEEE Trans. Reliability, 1996.

[5] James F. Kurose and Keith W. Ross. *Computer Networking, A Top-Down Approach*. Pearson, Essex, England, sixth edition, 2013.

[6] Erlend Riis Jahren. Implementing CSP over I2C for the new repository on the NTNU Test Satellite. Technical report.

[7] GomSpace. CubeSat Space Protocol(CSP). [Online]. http://www.gomspace.com/documents/GS-CSP-1.1.pdf (accessed 26.01.2015), .

[8] GomSpace. GomSpace Profile. [Online]. http://www.gomspace.com/index.php?p=profile (accessed 01.04.2015), .

[9] GomSpace. libcsp. [Online]. https://github.com/GomSpace/libcsp (accessed 06.04.2015), .

[10] Wikipedia. CubeSat Space Protocol. [Online]. http://en.wikipedia.org/wiki/Cubesat_Space_Protocol (accessed 06.04.2015).

[11] Roger Birkeland. NUTS Backplane. [Online]. https://www.ntnu.no/wiki/display/nuts/Backplane (accessed 06.04.2015), .

[12] Andreas Giskeødegård. Implementing CSP over I2C on NTNU Test Satellite. Project report, NTNU, Trondheim, 2012.

[13] Blakkisrud J. Delabahan C. Munthe-Kaas N. Bjørnevik, A. and Ø. Sture. Testing of the CSP implementation on the NTNU Test Satellite. Experts in Teams, Project report, NTNU, Trondheim, 2014.

[14] Yang Qin and Lie-Liang Yang. Throughput Comparison of Automatic Repeat Request Assisted Butterfly Networks. Technical report, School of ECS, University of Southampton.

[15] Ekram Hossain Long B. Le and Michele Zorzi. Queueing Analysis for GBN and SR ARQ Protocols under Dynamic Radio Link Adaptation with Non-Zero Feedback Delay. [Online]. http://www.necphy-lab.com/pub/Queueing%20Analysis%20for%20GBN%20and%20SR%20ARQ%20Protocols%20under%20Dynamic%20Radio%20Link%20Adaptation%20with%20Non-Zero%20Feedback%20Delay.pdf (accessed 31.05.2015).

[16] Mathias Tømmer and Knut Aldrin Wikström. UHF Radio. [Online]. https://www.ntnu.no/wiki/display/nuts/UHF+Radio (accessed 04.04.2015).

[17] Roger Birkeland and Jon Petter Skagmo. OWL VHF Radio. [Online]. https://www.ntnu.no/wiki/display/nuts/Owl+VHF (accessed 04.04.2015).

[18] William A. Beech, Douglas E. Nielsen and Lee Knoper. AX.25 Amateur Packet-Radio Link-Layer Protocol. Publication, 1998.

[19] Marius Münch. Integration and verification of a keyed-hash of a message authentication scheme based on broadcast timestamps for NUTS. Master Thesis, NTNU, Trondheim, 2014.

[20] Jon Petter Skagmo. NGHam Protocol. [Online]. https://github.com/skagmo/ngham (accessed 20.05.2015).

[21] Flogard E. Gammelsæter M. Mork-H. Pignède A. Solberg S. Bakkebø, N. Image Compression. NUTS - Experts in Teams, Project report, [in Norwegian], NTNU, Trondheim, 2014.

[22] Andreas Bertheussen. Digital Processing System for a CubeSat Camera. Project report, NTNU, Trondheim, 2014.

[23] Terry Ritter. The Great CRC Mystery. [Online]. http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM (accessed 04.04.2015).

[24] FreeRTOS. FreeRTOS. [Online]. `http://www.freertos.org` (accessed 07.04.2015), .

[25] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages*. Pearson, Essex, England, fourth edition, 2009.

[26] FreeRTOS. FreeRTOS - Queues, Mutexes, Semaphores... [Online]. `http://www.freertos.org/Real-time-embedded-RTOS-mutexes.html` (accessed 20.05.2015), .

[27] Michael Barr. CRC Series, Part 3: CRC Implementation Code in C/C++. [Online]. `http://www.barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code` (accessed 02.06.2015).

[28] Thomas Hanssen Nornes. former NUTS member. Personal conversation.

[29] Matplotlib. Introduction. [Online]. `http://matplotlib.org` (accessed 02.06.2015).

[30] Atmel. AT32UC3A3/A4 Series Summary. [Online]. `http://www.atmel.com/Images/32072s.pdf` (accessed 02.06.2015).

ACRONYMS

I^{2}C Inter-Intergrated Circuit.

BER Bit Error Rate.

PER Packet Error Rate.

b-c best case.

w-c worst case.

ACK Acknowledgement.

ADCS Attitude Determination and Control System.

API Application Programming Interface.

ARQ Automatic Repeat Request.

CLI Command Line Interface.

CRC Cyclic Redundancy Check.

CSP CubeSat Space Protocol.

EiT Experts in Teams.

FEC Forward Error Correction.

GBN Go-Back-N.

GS Ground Station.

LGPL GNU Lesser General Public Licence.

ms milliseconds.

mutex mutual exclusion.

NAK Negative acknowledgement.

NAP NUTS Authentication Protocol.

NRP NUTS Reliable Protocol.

NUTS NTNU Test Satellite.

OBC On-Board-Computer.

OS operating system.

OSI Open System Interconnection.

RDP Reliable Datagram Protocol.

RTT Round-Trip-Time.

SR Selective-Repeat.

SRAM Static Random Access Memory.

SW Stop-And-Wait.

UHF Ultra High Frequency.

USB Universal Serial Bus.

VHF Very High Frequency.

APPENDIX

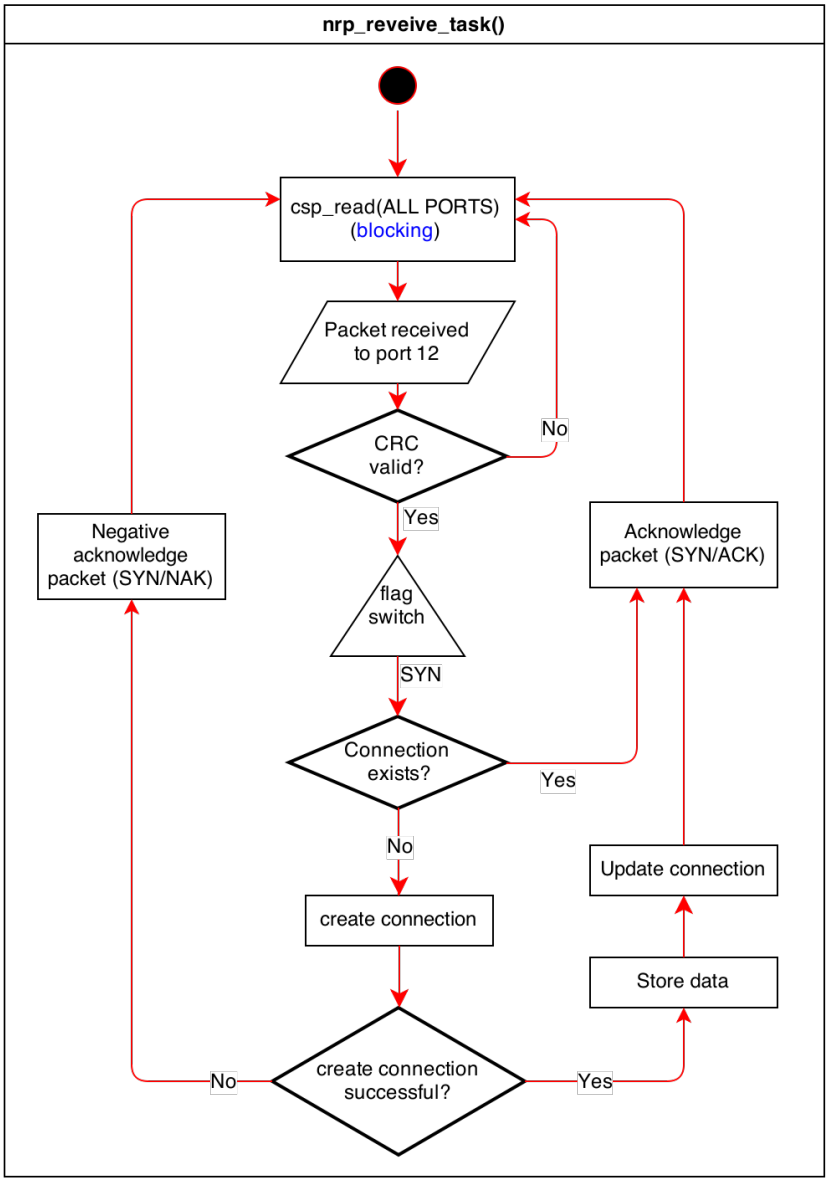
A

NRP DESIGN

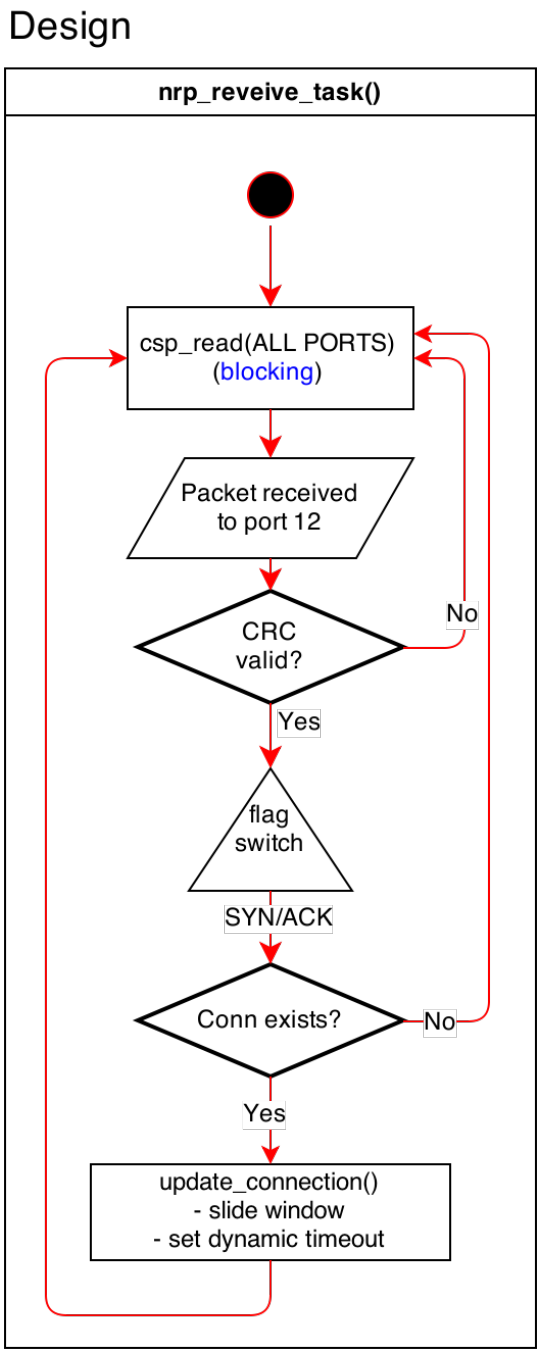
The figures in this appendix chapter illustrates NRP behavior that was excluded from the main chapters of this report.

A.1 Receive packet procedures

A.1.1 SYN packet

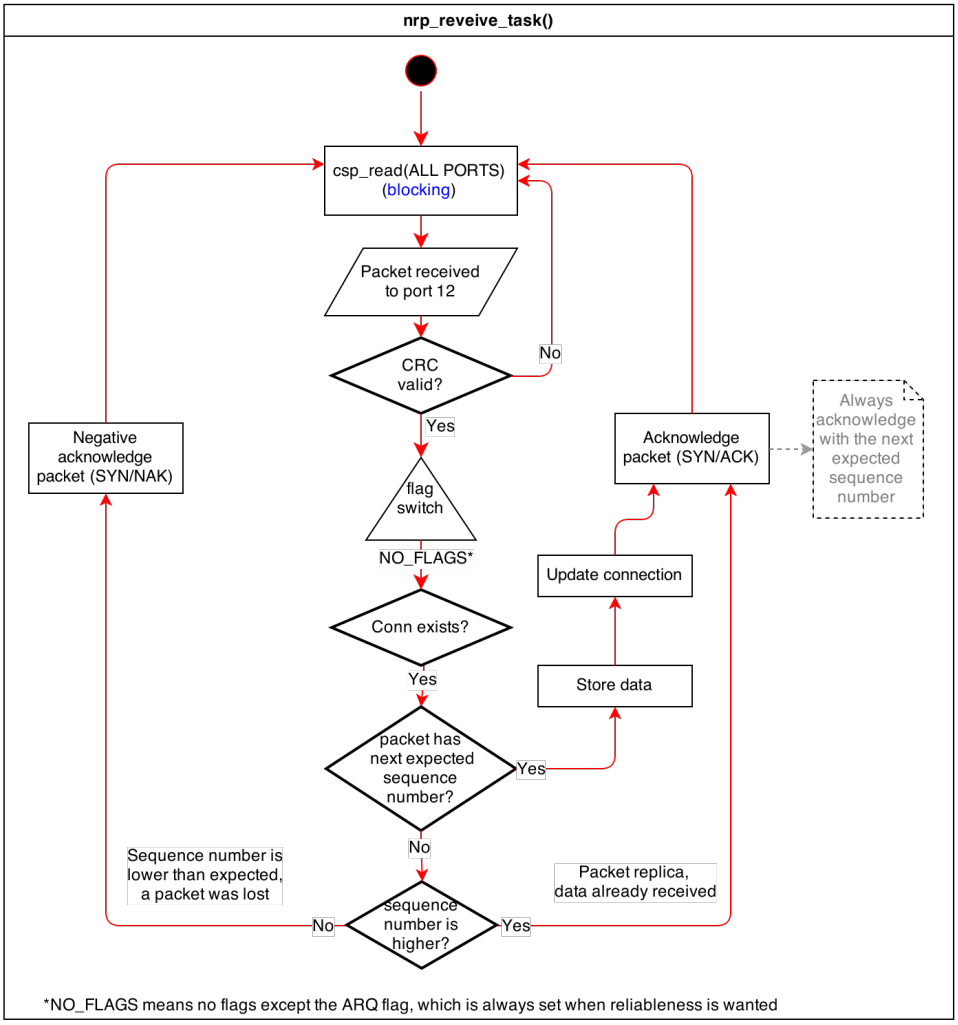


A.1.2 SYN/ACK packet

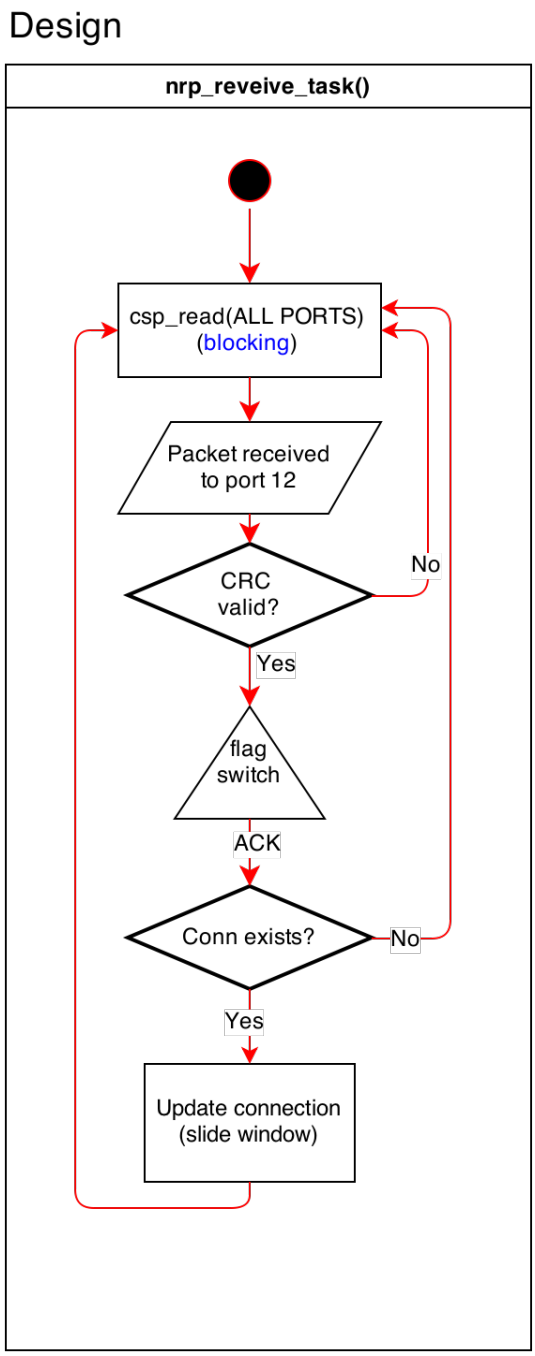


A.1.3 Content packet (No flags)

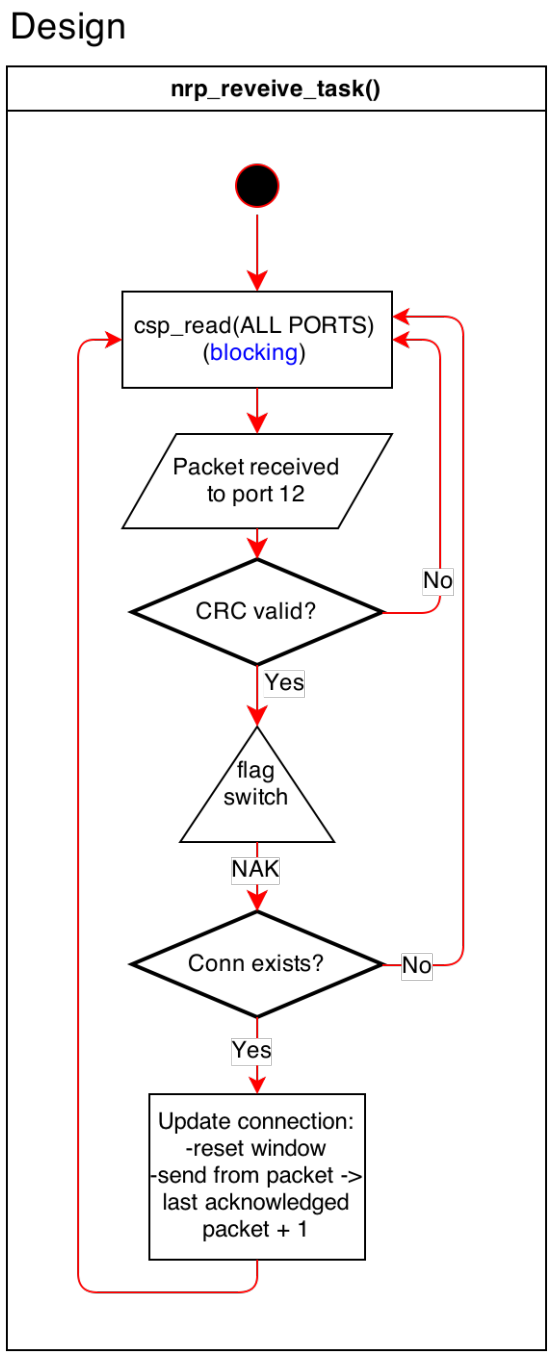
Design



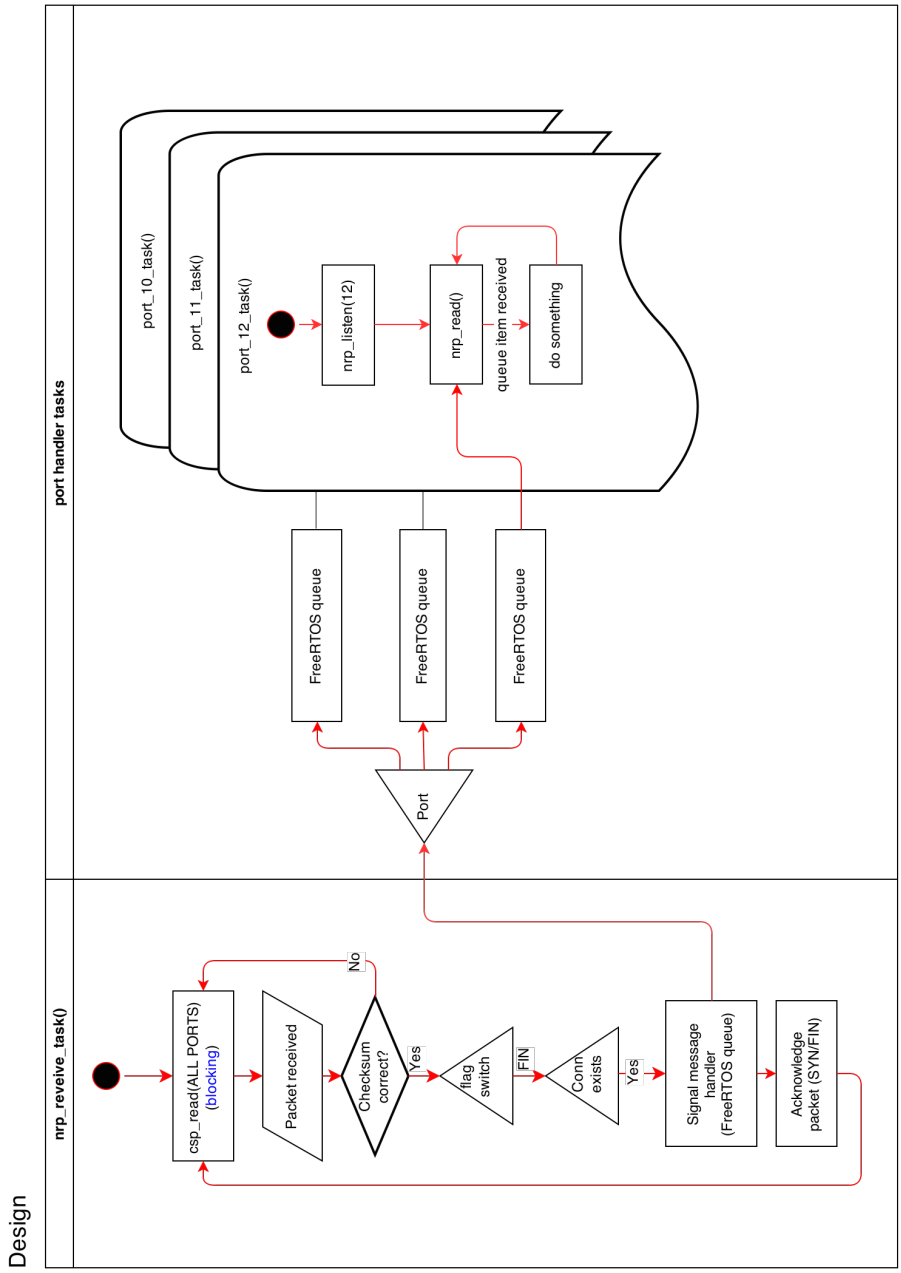
A.1.4 ACK packet



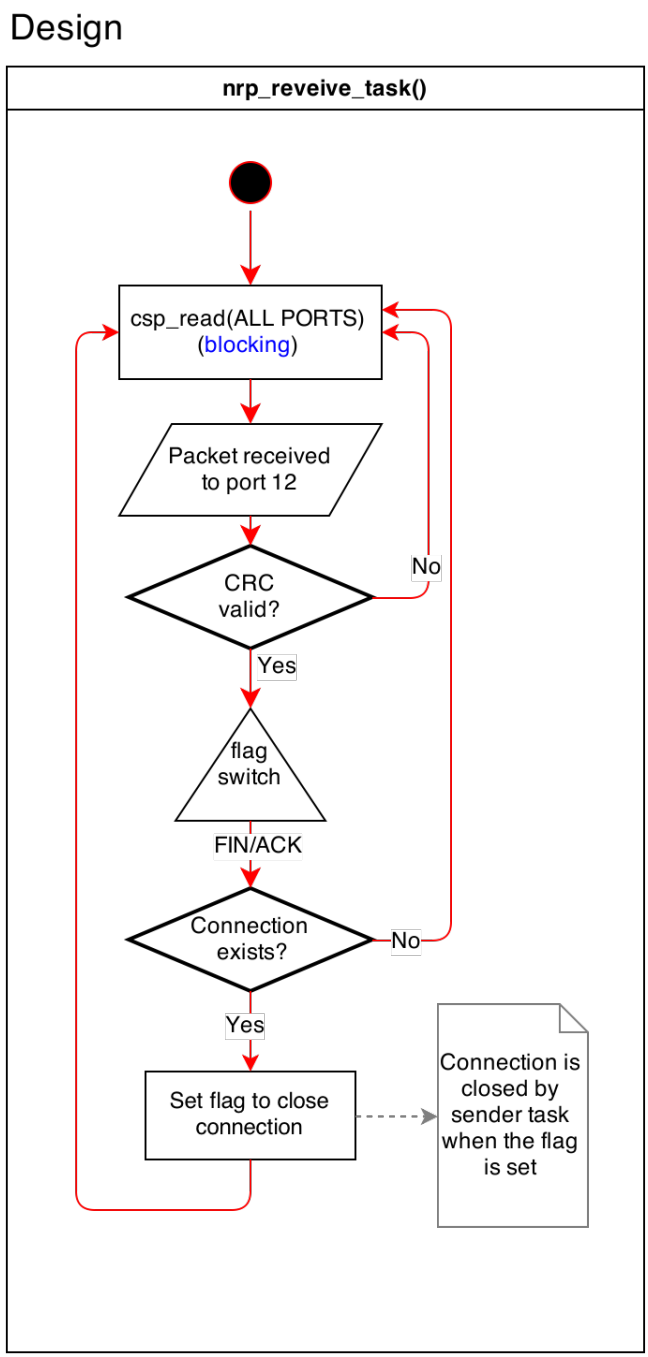
A.1.5 NAK packet



A.1.6 FIN packet



A.1.7 FIN/ACK packet



A.2 Connection events

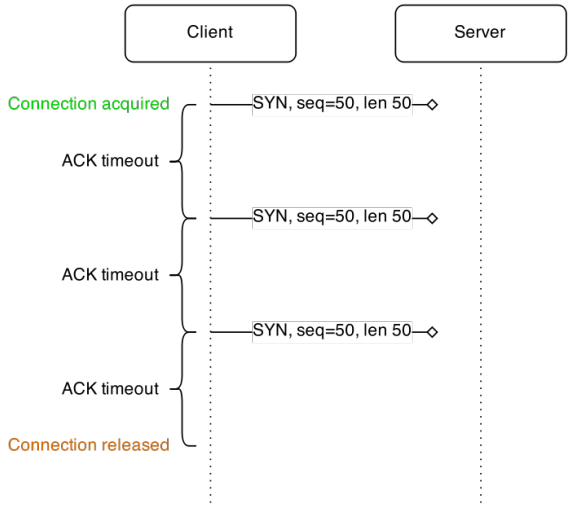


Figure A.1: Triple timeout

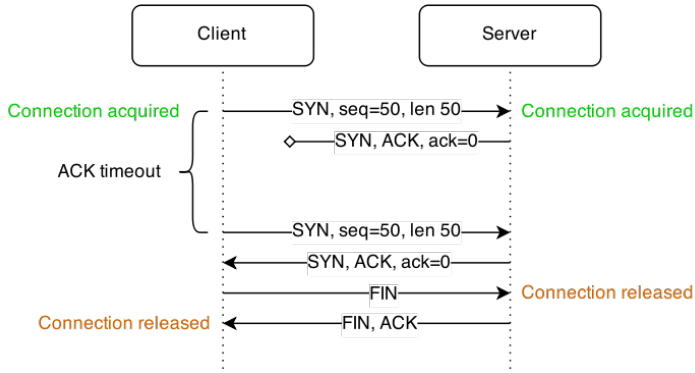


Figure A.2: ACK lost during connection setup

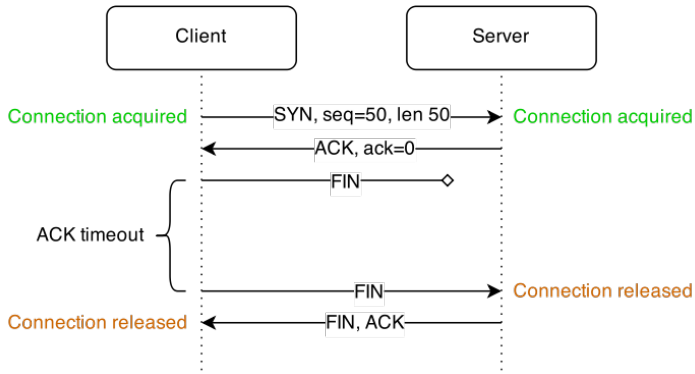


Figure A.3: FIN lost during connection termination

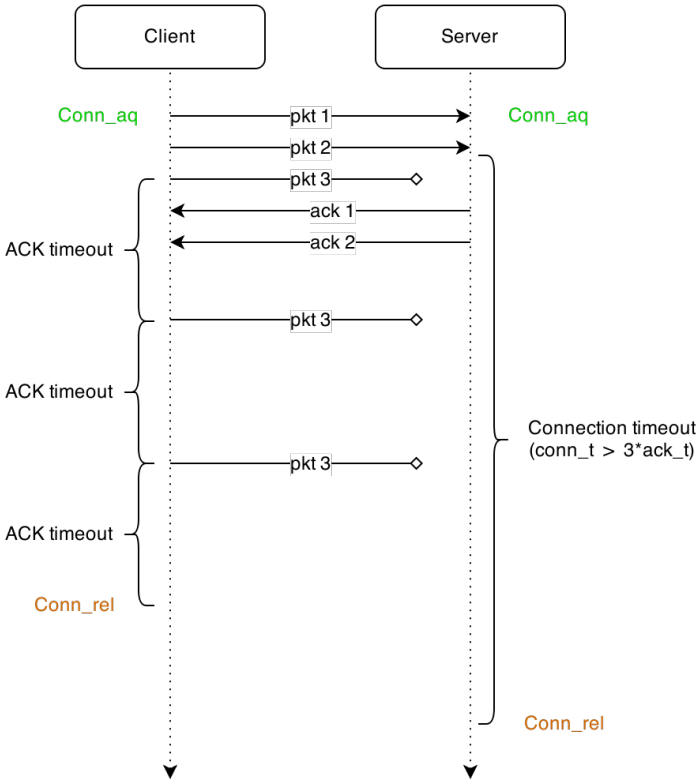


Figure A.4: Connection timeout when radio contact is lost during transmission

APPENDIX

B

TRANSMISSION CAPACITY GRAPHS

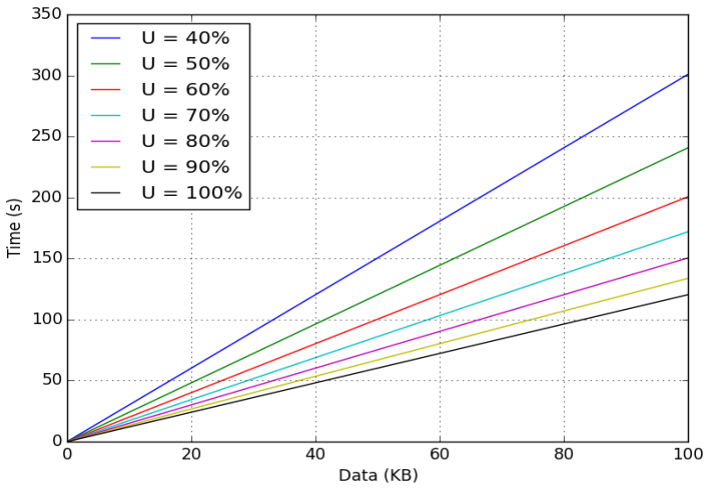


Figure B.1: Theoretic data versus time comparison for different utilizations on the NUTS radio link using AX.25

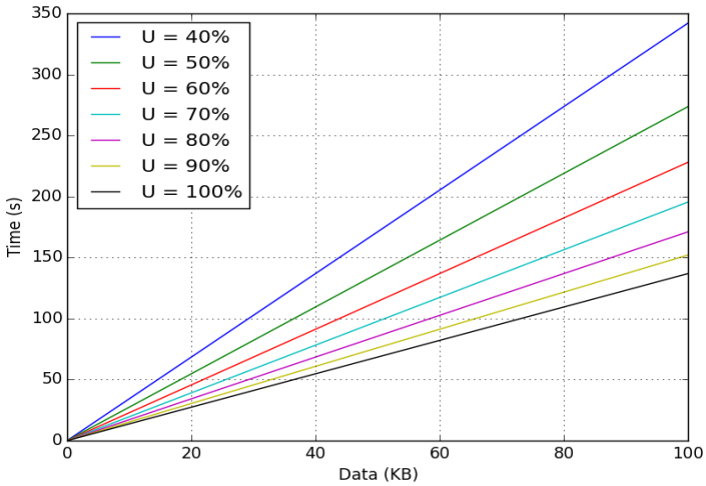


Figure B.2: Theoretic data versus time comparison for different utilizations on the NUTS radio link using NGHam

APPENDIX

C

BIT-FLIP FUNCTION

```
typedef struct {
  uint8_t byte;
  uint8_t bit;
}test_bit_flip_t;

void flip_random_bits(uint8_t *packet, uint8_t number_of_bitflips){
  uint8_t length = NRP_HEADER_SIZE + packet[NRP_PAYLOAD_LENGTH_BYTE];
  test_bit_flip_t flipped[number_of_bitflips];
  for (int i = 0; i < number_of_bitflips; i++){
      uint8_t bit = rand() % 8;
    uint8_t in_byte = rand() % length;

    //check that the bit is not already flipped
    Bool already_flipped = false;
    for (int j = 0; j < i; j++){
      if (flipped[j].byte == in_byte && flipped[j].bit == bit){
        i--;
        already_flipped = true;
        break;
      }
    }

    //flip bit with XOR
    if (!already_flipped){
      packet[in_byte] ^= 1 << bit;
      flipped[i].byte = in_byte;
      flipped[i].bit = bit;
    }
  }
}
```

Listing C.1: Bit-error inserting test function

APPENDIX

D

SPEED TEST

w: window size
P_e: Packet error probability
avg time: Average time in ms
P_{loss}: Measured packet loss
s. fails: Stream fails

w	P_e	avg time	P_{loss}	s. fails
1	0.0%	1003	0.00%	0.00%
1	0.5%	1061	0.49%	0.00%
1	1.0%	1118	0.98%	0.00%
1	2.0%	1237	1.95%	0.00%
1	5.0%	1624	4.98%	0.05%
1	7.5%	2010	7.51%	0.20%
1	10.0%	2330	9.87%	1.35%
1	15.0%	3113	14.91%	4.85%
1	20.0%	3926	19.89%	14.20%
1	30.0%	5683	29.69%	53.00%
2	0.0%	743	0.00%	0.00%
2	0.5%	756	0.48%	0.00%
2	1.0%	773	1.00%	0.00%
2	2.0%	813	1.96%	0.00%
2	5.0%	1008	4.97%	0.00%
2	7.5%	1234	7.49%	0.15%
2	10.0%	1449	9.95%	0.30%
2	15.0%	2017	14.79%	0.70%
2	20.0%	2823	19.78%	2.60%
2	30.0%	5390	29.86%	16.85%
5	0.0%	735	0.00%	0.00%
5	0.5%	747	0,49%	0.00%
5	1.0%	761	1,00%	0.00%
5	2.0%	797	1,98%	0.00%
5	5.0%	899	4,97%	0,05%
5	7.5%	1017	7,50%	0,10%
5	10.0%	1151	9,92%	0.00%
5	15.0%	1508	14,87%	0,90%
5	20.0%	1931	19,86%	3,65%
5	30.0%	3892	29,89%	22,35 %
10	0.0%	737	0.00%	0.00%
10	0.5%	747	0,50%	0.00%
10	1.0%	767	1,00%	0.00%
10	2.0%	798	2,02%	0.00%
10	5.0%	915	4,96%	0,05%
10	7.5%	1046	7,38%	0,05%
10	10.0%	1376	10,02%	0.00%
10	15.0%	1565	14,84%	0,85%
10	20.0%	2512	19,89%	4,70%
10	30.0%	4008	29,88%	22,60%

Table D.1: Data points for the speed test described in Section 6.9.