DEGREE PROJECT IN COMMUNICATION SYSTEMS, SECOND LEVEL
STOCKHOLM, SWEDEN 2015

Enhancing user satisfaction in 5G networks using Network Coding

VICTOR JOHANSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Enhancing user satisfaction in 5G networks using Network Coding

Victor Johansson

2015-07-22

Master Thesis

Examiner and Academic adviser
Gerald Q. Maguire Jr.

Industrial advisers
Annikki Welin and Tomas Thyni

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

Network data rates are growing rapidly. The data rates provided to the customers by their network providers vary from Mbps to Gbps. However, rarely do users get the promised *peak* throughput.

In cellular networks, network conditions change based on obstacles, weather conditions between the client and the base stations, and even the movement of objects and people. As a result of the changes in the radio link, the data transfer rate can change rapidly, hence devices needs to adjust their communications based on the currently available data rate.

The Transmission Control Protocol (TCP) is widely used for reliable data transfer over networks. However, TCP was initially designed when link data rates were much lower than the link data rates commonly available today. As a result, TCP does not perform well at high data rates, despite some of the changes that have been made to the protocol to support high data rate links. Moreover, TCP has problems adapting to large changes in link bandwidth (not caused by congestion), resulting in a lower average throughput than the link could potentially deliver.

This thesis evaluates two different versions of the TCP protocol (e.g., TCP Reno and Cubic TCP) and proposes a network coding scheme to enhance users' experience when communicating over unstable radio links. The performance of the two TCP protocols and Random Linear Network Coding (RLNC) scheme were measured in an emulated network environment. The results of these measurements were analyzed and evaluated. The analysis shows that RLNC can provide a higher throughput than TCP over a network with high packet loss. However, RLNC is a UDP based solution and does not implement congestion control algorithms or reliability. A new solution is proposed that increases reliability and implements network adaptation in RLNC solutions.

The results obtained in this thesis can be used to develop a new protocol to increases the quality of users' experience in high loss networks.

Keywords

TCP, Cellular network, Throughput, Bandwidth, Network coding, UDP, Emulated network, Latency, Link data rate

Sammanfattning

Datahastigheter över nätverk ökar drastiskt. Datahastigheterna som ges tillgängliga till användare av deras respektive dataleverantör kan variera från Mbit/s till Gbit/s. Det är dock inte ofta användare får ut vad som har lovats.

I mobila nätverk kan nätverkets tillstånd ändras baserat på hinder, väderleksförhållanden mellan en klient och basstationerna, till och med beroende på förflyttning av objekt eller människor. På grund av detta så behöver användares utrustning anpassa dess kommunikation, baserat på den för närvarande tillgängliga datahastigheten.

Transmission Control Protocol (TCP) används i stor utsträckning vid behovet av tillförlitlig dataöverföring över nätverk. Däremot så designades TCP när länkdatahastigheterna var mycket lägre än vad som är vanligen tillgängligt idag. På grund av detta så presterar inte TCP över höga datahastigheter, trots ändringar som har gjorts i protokollet för att stödja höghastighets datalänkar.

Utöver det så har TCP svårt att anpassa sig efter stora ändringar i länkens bandbredd (som inte är orsakat av stockning), som resulterar i en mindre genomsnitts-dataström än vad länken potentiellt hade kunnat ge.

Detta examensarbete utvärderar två olika versioner av TCP (e.g., TCP Reno och Cubic TCP) och föreslår ett sätt att använda network coding för att öka användares upplevelse vid dataöverföring över instabila radio länkar. Prestationerna av de två TCP versionerna och Random Linear Network Coding (RLNC) metoden har blivit mätt i en emulerad nätverksmiljö. Resultaten från dessa mätningar blev analyserade och utvärderade. Analysen visar att RLNC kan ge en högre dataström än TCP över ett nätverk med hög risk för paketförluster. Däremot så är RLNC en User Datagram Protocol (UDP) baserad lösning, och därav inte implementerar trängselkontrolls-algoritmer eller tillförlitlighet. Ett förslag till en ny lösning som ökar tillförlitlighet och implementerar nätverksanpassning till RLNC lösningar har presenterats.

Resultaten från detta examensarbete kan användas till att utveckla nya protokoll för att öka kvalitén av användares upplevelse i nätverk med risk för hög paketförlust.

Nyckelord

TCP, mobila nätverk, dataströmmar, bandbredd, Network Coding, UDP, Emulerade nätverk, latens, länk datahastighet

Acknowledgments

I would like to thank the people that I have been working with, who has been supportive and helpful throughout the thesis project, namely:

Patrik Hagernäs and Qiaochu Chen, two fellow thesis workers at Ericsson of who I shared the testbed with. We cooperated in creating scripts for measurements in the testbed, and they helped me by discussing problems and provided me with useful ideas.

Professor Gerald Q. Maguire Jr., my academic adviser at KTH, for his continuous help and support throughout the thesis.

Annikki Welin, my supervisor at Ericsson, for providing me with relevant information, discussing problems, and keeping me on track during the thesis.

Tomas Thyni, my supervisor at Ericsson, for discussing problems and providing me with inspiration about how to solve different tasks.

Eric Andersson, an employee in Ericsson's Radio network department, for providing me with network statistics and for bringing forward relevant ideas about real network scenarios.

I would also like to thank Steinwurf APS for providing me with a license for the Kodo API.

I would like to thank Professor Luigi Rizzo at the University of Pisa, for his help by answering questions about problems I faced with his tool Dummynet, which was used in the thesis project.

Stockholm, July 2015
Victor Johansson

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

ACK	Acknowledgement
BIOS	basic input/output system
BS	Base Station
BWM-NG	Bandwidth Monitor-Next Generation
CDF	Cumulative Distribution Function
CSV	Comma Separated Values
CTCP	Coded Transmission Control Protocol
CPU	Central processing unit
Cwnd	Congestion window
DCCP	Datagram Congestion Control Protocol
GEO	Geostational Orbit
HDD	Hard disk drive
HSCP	Hybrid Secure Copy Protocol
HSPA	High Speed Packet Access
ICT	Information and Communication Technology
LTE	Long Term Evolution
MAC	Media Access and Control
ms	Milliseconds
MSS	Maximum Segment Size
OS	operating system
PEP	Performance Enhancing Proxy
RNC	Random Network Coding
RLNC	Random Linear Network Coding
RTO	Retransmission Timeout
RTT	Round Trip Time
SAN	Storage Area Network
SCP	Secure Copy Protocol
SCTP	Stream Control Transmission Protocol
SMSS	Sender Maximum Segment Size
ssthresh	slow start threshold
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE	User Equipment
WAN	Wide Area Network

1 Introduction

This chapter contains an introduction to the subject. It provides the reader with a general background of Transmission Control Protocol (TCP) and network coding. The problem that this thesis addresses will be defined and explained. Following this the purpose, goals, and methodology of the research are presented. Finally, the structure of the thesis is outlined.

1.1 Background

In 1981, TCP was defined in RFC 793 [1]. TCP is part of the foundation of the internet protocol suite, commonly known as TCP/IP. TCP remains the most widely used protocol for reliable transfer when sending a sequence of bytes between applications over a network. However, other protocols, such as the Stream Control Transmission Protocol (SCTP), have proven to be a potential competitor [2].

A lot has changed in networking technology since 1981. According to Nielsen's law of internet bandwidth "a high-end user's connection speed grows by 50% a year"[3]. Nielsen's law claims that in 1984, the throughput of a high-end user over a network link was only about 300 bps and in 2013 this had increased to about 100 Mbps. Users living in capital cities with fiber in their buildings could reach even higher throughputs of 1 Gbps. However, these rates were for wired networks. Wireless networks offer much lower data rates. The download data rate of my cellphone using my operator's Long Term Evolution (LTE) network averages 40 Mbps in 2015.

According to Ericsson's white paper "5G Radio Access" [4], in 2020 the first 5G networks will be deployed and these networks need to be able to support a traffic volume of 1000 times that of today, and peak data rates around 10 Gbps. Ericsson states that their 5G solution will not consist of a single technology, but rather will be an integrated combination of several radio-access technologies. Ericsson thinks that mobile-broadband technologies such as High Speed Packet Access (HSPA) and LTE will continue to be improved and will serve as a foundation for 5G. Although the network will support traffic of 1000 times of today, a typical user will still suffer from random packet loss errors and throughputs below the peak data rates. The latter is partly due to TCP and its weaknesses.

TCP was not specifically developed for wireless networks and struggles to adapt to rapid changes in the link's characteristics. The characteristics of a wireless link can change for several reasons. A large change in link throughput can occur due to a handover between different access network points, obstacles between the client and the access point, current weather conditions, and movement of objects and people. There are two different scenarios that can occur when the throughput of a link changes:

1. The first occurs when the link data rate changes from low to high. The sender will not notice that the available maximum throughput has changed and hence will not take advantage of this increased link data rate. This can happen when a user drives through a hotspot where the maximum link data rate is significantly higher than the macrocell via which the terminal was previously communicating.
2. The other occurs when the link changes from high data rate to low data rate. In this case the sender will attempt to send more data per unit time than the link can handle in its current state and not all of the packets can be successfully transmitted over the link, hence packets either will be buffered by the device or they will be lost. For example, this can occur when a user drives into a tunnel.

The link characteristics and competition for the link's capacity means that user equipment (UE) will, on many occasions, never achieve the maximum possible throughput of a given data link. The average throughput will be a much lower then what it could have been if a more adaptable protocol had been used.

T. Flach, et al. [5] examined the efficiency of TCP's loss recovery by examining traffic from Google's web services, excluding video traffic. They found that the time it took to deliver a file increased by a factor of 5 when there were packet losses in comparison to the file delivery time when there were no packet losses. The file delivery time was defined as the time from when the first byte was sent to when the last acknowledgement (ACK) was received. They also found that 77% of losses were recovered based upon TCP's timeout and not due to TCP's fast recovery algorithm. As TCP timeouts only happen after an idle period of time (to be explained farther in Section 2.1.2), the throughput of TCP decreases. As this packet loss was not caused by congestion, the decrease in throughput was *unnecessary*. For this and other reasons there is clearly a need to improve TCP's handling of losses.

The loss of a TCP segment due to causes other than congestion occurs frequently in wireless environments, due to the rapid changes in the link characteristics. Unfortunately, TCP handles each loss as if it were due to congestion. The usual reason stated for this is that the TCP nodes have no way of determining whether a segment was dropped due to a disturbance on a link or whether the segment was dropped because of congestion by a node in the network.

Network coding has been around since the year 2000. The basic idea is that bits in the information flow do *not* have to be delivered *independently*. Instead these bits can be mixed with other information flows, as long as the receiving host(s) has enough "evidence" or "clues" to sort out the original information [6]. There has been a lot of research in this area and *network coding* has proven to be a means to maximize information flow, especially in multicast networks.

1.2 Problem definition

An underlying assumption of TCP's congestion control algorithms is that *the risk of packet loss caused by damage is very small*. On wired networks the probability of a bit error for simple twisted pair wires is ~10^{-5}. Today, for fiber the probability of a bit error is generally under 10^{-15} (after decoding) [7].This assumption and the major concern of avoiding network congestion collapse [8] lead to TCP treating every packet loss as if it were caused by congestion in the network.

Unfortunately, the risk of packet loss caused by damage or disturbances in wireless environments is greater than the risk in wired environments. As noted earlier a disturbance can occur for many reasons. Many of these disturbances lead to a loss of packets for a period of time proportional to the velocity of the objects that are changing the radio propagation environment. However, a change in the link's characteristics does *not* necessary mean that the link's data rate changes, as it may be possible to overcome the impairments by increasing transmit power or by some other means. A disturbance caused by a handover between access points could cause the link data rate to change greatly, both due to the difference in available resources in the two different cells and because there might be a period of time during the handover when no frames are being transmitted. The *transmission* protocol needs to be able to recognize the difference between the different causes of losses and respond to these different causes in a more effective way than simply assuming that all errors are due to congestion or random bit errors.

Logically TCP keeps a timer for each segment in order to identify that a segment was lost. If an acknowledgement for the segment is not received from the receiver within a specific amount of time, a timeout occurs. If the connection is in the slow start phase, then the timeout triggers the congestion avoidance phase. If the timeout is triggered during the congestion avoidance phase, then the slow start algorithm starts over again. In both cases the algorithm threats the timeout as an indication of network congestion. Additionally, as described in K. Mansley's article "Tweaking TCP's Timers" [9], TCP implementations do not actually keep a timer per segment, but rather most implementation share a periodic timer. The purpose of using the different algorithms that have been

used in the different TCP implementations is for TCP to adapt to the data link's actual throughput in order to achieve as high a throughput as possible *without* causing network congesting.

1.3 Purpose

The aim of this thesis project is to propose a solution that will improve the performance of transmissions over wireless links, when there are potentially high error rates. Based upon my research and evaluation, I expect to increase my knowledge of both the TCP protocol and network coding. I also expect to learn how to apply my previously acquired knowledge of networking in practical environments.

The results of this research will guide other researchers and facilitate the future development of a reliable transport protocol for 5G networks.

Ericsson expects to benefit from the results of this thesis. Researchers at the company have been present during the research and they will directly gain information about the problems, proposed solutions, and research methodology used during this thesis project. A 10Gbit/s throughput emulation network is used to create a testbed for use in this thesis project. This testbed will be used within the company for further research.

The public will also benefit from this research, as it might lead to a solution that could increase a network user's effective link data rates in the future. Ericsson predicts *"beyond 2020, wireless communication systems will support more than 1,000 times today's traffic volume"* [4]. If this prediction is true, then congestion will probably decrease and more bandwidth should be available for users. If users can effectively utilize this greater bandwidth, then they will be more satisfied in terms of being able to download (or upload) files faster than they can do so today.

1.4 Goals

The goal of this project is to evaluate the performance of TCP and an implementation of network coding over a link with a high packet loss rate. The two different methods will be compared by measuring their performance in an emulated network environment. Finally, an improvement is to be proposed to increase user satisfaction (with regard to increased user data rates) in future wireless networks.

This goal has been divided into the following sub-goals:

1. Understand the TCP protocol and the effects of random error based packet losses.
2. Evaluate the effects of using Random Linear Network Coding (RLNC) from the Kodo library (these topics will be described in Sections 2.2.2 and 2.2.2.2, respectively).
3. Compare the results of using RLNC to the performance of TCP.
4. Draw conclusions based on these results in order to suggest how to increase the performance experienced by users communicating over high data rate but high packet loss rate wireless links.

1.5 Research Methodology

The research methodology used is to emulate a real network and to use implementations of protocols that are currently used by the public in realistic scenarios. The reason why this methodology was chosen opposed over simulation is that emulation will also expose problems that might exist in hardware implementations when implementing networks with this magnitude of data and error rates. This is of great importance in this thesis as network coding consumes computational resources at several nodes, which might be obscured in simulated environments.

Measuring how much load this computation will place on the network nodes in order to reach a throughput of 10 Gbit/s can be explored by using an emulation environment.

1.6 Structure of the thesis

Chapter 1 presents a general introduction to the subject. This includes a short background, a definition of the problem, the purpose of the report, the aims and goals of the research, and a brief description of the research methodology.

Chapter 2 presents information and background about the TCP protocol in general, basic ideas behind network coding, and the Random Linear Network Coding Scheme. It also refers to other research that has been done to improve throughput either by improving TCP or by applying network coding schemes.

Chapter 3 presents the methodology used in this thesis project.

Chapter 4 presents the experimental testbed created for the project. The design, the tools that were used, and the limitations I faced are presented and discussed.

Chapter 5 describes the development of a system for transferring data using network coding. The different applications are described, as well as a TCP controller thread that was implemented to improve the system. Finally, potential improvements to the applications are presented, along with a summary of the system's design.

Chapter 6 presents the results and the analysis conducted in this project. Baseline results suggest how users might experience the throughputs of today are discussed. The performances of the different methods that have been evaluated are compared. Finally, the chapter ends with a discussion of the results.

Chapter 7 presents conclusions drawn from the analysis, as well as suggests future work to be done following this project. The chapter also discusses the limitations that affected the project and some reflections on how this research affects society.

2 Background

This chapter provides the reader with background information about TCP and network coding. The information provided will be sufficient to understand how TCP operates and the basic ideas behind network coding. The RLNC scheme used in this thesis project is explained in detail. The chapter also describes related work that has been carried out to improve throughputs by improving TCP, as well as applying network coding schemes.

2.1 Transmission Control Protocol

TCP is the most widely used protocol for reliable data transfer. Two peers using the protocol begin by setting up a connection between two TCP ports, one peer acting as a sender and the other as a receiver. These ports can be located on a single host or on two different hosts on the network. Logically, for each segment (a sequence of bytes) sent from the sender to the receiver, an acknowledgement is sent back to the sender to notify it that the segment reached its destination; hence the sender can send the next segment. If an acknowledgement is not received for a segment, then the sender believes that the segment was lost and resends it. In this way the sender ensures that every byte being sent over this TCP connection reaches its destination and is delivered to the peer TCP's application in order, hence TCP is a reliable byte stream transfer protocol.

While the description above describes the logical operation of TCP, in reality implementations have been optimized to reduce the amount of unnecessary traffic (for example, by delaying acknowledgements, hence multiple segments can be acknowledged at the same time) and to increase the throughput of the TCP connection. In the following subsections I will thoroughly examine the implementation called "TCP Reno", as this is the most widely known version of the protocol. The following subsections will give a more detailed description of how the protocol operates, and what improvements have been made in subsequent implementations, specifically: TCP NewReno and Cubic TCP.

This discussion begins by presenting the three-way handshake used to initiate a TCP connection. This is followed by a discussion of the data transfer phase. Following this I will introduce details of some of the newer TCP implementations.

2.1.1 Three-way Handshake

To set up a TCP connection, a three-way handshake is used. This handshake is initiated by the sender who sends a synchronization request (SYN) to the receiver. The receiver responds with an ACK to the sender along with its own SYN request. Note that the sender and receiver's SYN sequence numbers are both independent and preferably random 32 bit unsigned integers. The sender responds with an ACK of the receiver's SYN request, thus indicating that the TCP connection is established. This procedure has to be done every time a new TCP connection is established. An example of this procedure is shown in Figure 2-1.

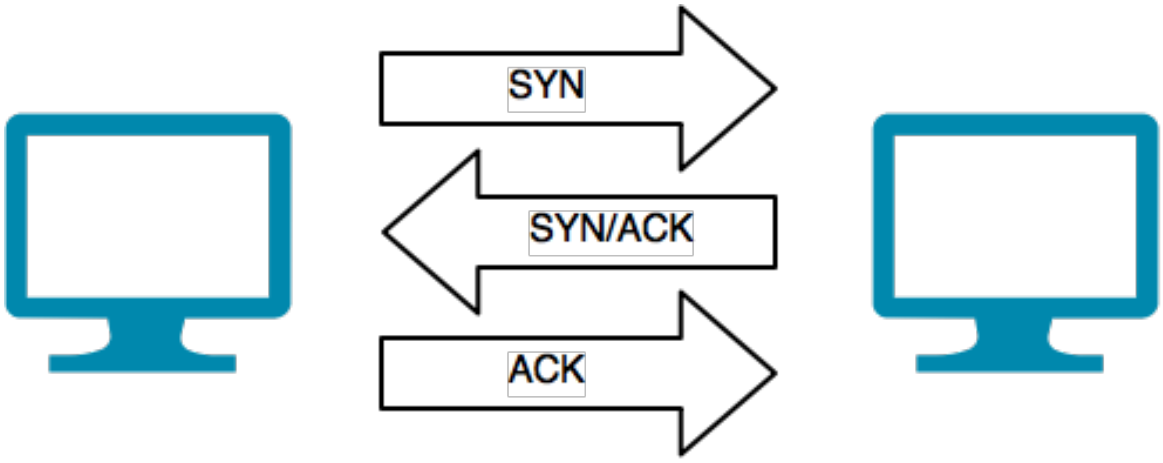


Figure 2-1: **The TCP three-way handshake procedure. The figure shows the packet flow to initialize a TCP connection between two hosts.**

2.1.2 Data transfer

TCP employs congestion control when sending bytes over a TCP connection. Congestion control is implemented by several algorithms that are designed to enable the transfer rate to reach as high a throughput as possible *without* overwhelming the network with more traffic than it can handle. The congestion control algorithms used by TCP Reno are: slow start, congestion avoidance, and fast recovery. In addition, TCP implements a sliding window scheme to prevent the sender from over running the buffer space that the receiver has, i.e., avoiding sending more data than the receiver is prepared to receive. However, with the increasing amount of memory that many receivers have, there is less and less need for this flow control. The flow control function is implemented at the receiving host of the TCP connection. It operates by informing the sender how many octets this receiver is ready to receive, starting from the last successfully received byte (offset). Note that the sequence number indicates the next byte (offset) that is expected.

When sending data, two parameters are maintained: (1) a congestion window (cwnd) and (2) a slow start threshold (ssthresh). The cwnd tells the sender how many bytes can be sent *without* the network becoming congested, while ssthresh indicates when TCP should begin to apply a congestion avoidance algorithm.

2.1.2.1 Slow Start

The sender begins with a cwnd of a small multiple of the Sender Maximum Segment Size (SMSS, the largest segment size that the sender is allowed to send). For each ACK received, the sender increases cwnd by one SMSS. This results in an exponential increase in the cwnd. This process continues until cwnd is greater than or equal to ssthresh. Initially, ssthresh is set to an arbitrarily high value. Slow start continues until the end-to-end capacity of the paths between the source and destination is reached and packets begin to be dropped, as this is viewed as a sign of network congestion. While this might also be due to reaching the limit of a bottleneck link, this is not the interpretation embodied in TCP's design.

TCP keeps track of when each segment is sent. If an ACK is not received within a specific period of time (when the sender expects to receive an ACK), then a timeout occurs and the segment has to be re-sent. The timeout is calculated by estimating the Round Trip Time (RTT) of the connection. Unfortunately, if the RTT of a TCP connection varies a lot, this will make it difficult to find a good timeout value. Algorithms have been developed to compute a good average value, thus enabling TCP to adapt to the current state of the connection. However, these algorithms utilize a linear model of

the behavior of the link delays and hence finding an optimal value when a link's bandwidth varies rapidly is still a problem.

When the first timeout occurs, ssthresh is reduced to half of the current cwnd or to a minimum of twice the SMSS.

2.1.2.2 Congestion avoidance

The congestion avoidance algorithm increases the sender's sending rate by increasing cwnd by one segment each round trip time (RTT), where RTT is the time it takes for a segment to be sent from the sender until the corresponding acknowledgement from the receiver arrives at the sender. The purpose of this congestion avoidance algorithm is to avoid causing excessive congestion in the network. It achieves this by changing from exponentially increasing the size of cwnd for each ACK that is received to only linearly increasing the size of cwnd for each RTT. When an isolated segment is lost, the receiver will notice a gap in the sequence numbers of the packets received. Since the basic TCP protocol only uses cumulative ACKs, the receiver will send a duplicate ACK (i.e., an ACK corresponding to the last acknowledged segment) to notify the sender of the sequence number of the last successfully received segment that had been received in order – but also indicating that an additional segment has been received, but that this segment is not the expected segment (hence there is one or more missing segments).

If three duplicate ACKs are received by the sender, then TCP initiates the fast recovery algorithm – since it knows that segments are getting through, but that the segment following the last acknowledged segment is missing.

2.1.2.3 Fast Recovery

The fast recovery algorithm has three steps:

1. First the ssthresh parameter is reduced to half of the current cwnd, but not less than two segments. Then the algorithm resends the missing segment using a *fast retransmit* which ignores the retransmission timer and immediately sends the missing segment. The cwnd is then set to ssthresh + three times the segment size. This is to take into account the segments that have been successfully received by the receiver, but that have not yet been acknowledged. This is done because the duplicate ACKs indication that the receiver has received some segments out of order (i.e., segments continue to be received – hence there is no network congestion, but rather there is simply a gap in the expected order of segments being received).
2. The sender waits for an ACK that acknowledges the newly sent data. The sender continues to increment cwnd by one segment for each duplicate ACK received, in order to acount up for the segments received by the receiver. This algorithm enables the sender to send another segment just as soon as it is allowed based upon the cwnd.
3. When the first ACK that acknowledges newly sent data is received, then the cwnd is set to ssthresh. This ACK acknowledges all the segments sent since the first duplicate ACK. After this, TCP returns to using the congestion avoidance algorithm.

If a timeout occurs for any segment during either congestion avoidance or fast recovery, cwnd is set to one and the slow start algorithm is initiated.

2.1.3 Improvements to TCP Reno

Newer versions of TCP have been developed to improve the algorithms of TCP Reno. Two of the best known versions are Cubic TCP and TCP NewReno. Each of these is described in the following paragraphs.

2.1.3.1 Cubic TCP

Cubic TCP [10] is widely implemented and has been the default congestion control algorithm in the Linux kernel since version 2.6.19. The algorithm is an improvement of the Binary Increase Congestion (BIC) congestion control algorithm, in order to improve TCP's friendliness and RTT-fairness.*

Apart from TCP Reno, Cubic TCP only uses one stage in its algorithm to handle congestion. The algorithm uses a cubic function to set cwnd. The function itself has two different parts: In the first part, window growth is a concave increase, ramping up to the cwnd prior to the observed congestion indication. In the second part, Cubic probes for higher throughput by increasing with an exponential growth, similar to TCP Reno's slow start phase.

2.1.3.2 TCP NewReno

TCP Reno's fast recovery algorithm acknowledges all the segments received at the receiver when the first ACK that acknowledges new data arrives. When this occurs, TCP Reno exits from the fast recovery algorithm and continues with the congestion avoidance algorithm. However, if multiple packets were dropped, then the sender would continue to receive duplicate ACKs and would once again initiate the fast recovery algorithm. In this case, the *cwnd* would be decreased once again and another fast retransmit performed. TCP NewReno improves TCP Reno's fast recovery algorithm by remembering the *last unacknowledged segment* sent when entering the fast recovery phase. The protocol increases cwnd by one for each duplicate ACK received and transmits a new segment whenever cwnd allows it. It uses *fast retransmit* to retransmit the next segment when an ACK that acknowledges new data is received. This procedure is repeated until the last unacknowledged segment is acknowledged. At this point, the protocol exits the fast recovery phase and returns to congestion avoidance.

2.2 Network Coding

This section introduces the concept of network coding and gives a detailed description of the RLNC scheme. The sub-sections provide information about network coding's role in wireless networks, as well as information about the network coding programming library used in this research.

2.2.1 What is network coding?

Network coding was first introduced by Ahlswede, et al. [11], who introduced the idea of a single node in the network combining incoming packets using linear combinations of these packets. One of the benefits of combining packets by their linear combinations is that you can send several independent packets over the same transmission link, with the same size as only one packet. This method can reduce the amount of data sent over network links, hence reducing the risk of congestion over a network node. However, when a combined packet arrives at the receiver, the receiver needs to have information about some of the other independent packets in the combination in order to extract the desired packet.

* Specifically to reduce the aggressiveness in the window reduction after losses and to reduce TCP's dependency on the RTT.

This idea was further evolved into developing network coding schemes to correct packet error bursts over a single transmission. Instead of having specific data in each packet, packets containing overall information about the whole file is transmitted. As long as the receiver receives enough overall information about the file, it can find out the original data by using the information acquired. The combining of several packets will be referred to as encoding the packets, while extracting a packet will be referred to as decoding a packet.

The encoding of packets can be performed in different ways, depending on what network coding scheme is being used. In addition to creating different kinds of combined packets, network coding schemes provides robustness against middle man attacks and packet sniffers. The combined packets do not make sense if you do not have information about the other packets used in the encoding and the scheme used in the encoding.

Network coding offers its greatest benefits in wireless environments, where packet losses due to disturbances on the network links happen more frequently that for wired networks. Network coding can also make use of the wireless network's broadcast nature to transmit packets to several hosts in order to distribute data or the network links can be utilized as a mesh network structure. Katti, et al. [12] applied this principle in their architecture named COPE. In their approach, broadcasts by hosts in wireless networks are stored packets in nearby nodes. These stored packets serve as potential evidence for future encoded packets.

2.2.2 Random Linear Network Coding

RLNC is a network coding scheme that uses random coefficients of a finite field to linearly combine packets. This technique operates by allowing nodes in the network (except for the receiving nodes) to perform random independent linear mappings of inputs onto each of their outgoing links. The receiver only needs to know the overall linear combination of source packets in order to decode a packet. The overall linear combinations of source packets can be passed on as a vector along with the data at each node. Failure in a transmission while using network coding will generally occur when the receiver does **not** receive enough evidence to be able to decode the desired data. Ho, et al. [13] presented success probabilities of their randomized coding scheme with different sizes of the finite field (these are shown in Table 2-1). The probability of successfully decoding a packet is shown to decrease, based on the distance to the receiving node. It also shows that the probability of successfully decoding packets is significantly higher when using a larger finite field.

Table 2-1: **Probabilities of success in a rectangular grid network containing 100 nodes. The receiver's positions are expressed in the form (row, column) relative to the sender's position at (1, 1). The bounds expressed in the first column of the table are the finite fields used while calculating the probabilities.**

Receiver position	**(3, 3)**	**(10, 10)**	**(2, 4)**	**(8, 10)**
F_{2^4} lower bound	0.597	0.098	0.597	0.126
F_{2^6} lower bound	0.882	0.567	0.882	0.604
F_{2^8} lower bound	0.969	0.868	0.969	0.882

This project focused on using RLNC to distribute data to several clients over error prone radio links. RLNC eliminates the requirement that *every* packet off data must be received at each client, by replacing it with the new requirement that *enough* **coded** packets are received at each client. As each coded packet is a linear combination representing all of the data, the client only needs to receive a sufficient number of coded packets to be able to solve the linear system of equations. This method also eliminates the need for the sender to keep track of which packets have been lost for

each independent client. The sender predicts the fraction of packets that will be lost and proactively sends extra coded packets to solve these expected packet loss issues.

RLNC uses User Datagram Protocol (UDP) as the underlying transport protocol, which unlike TCP is an unreliable protocol and does not implement any congestion control algorithms. However, as UDP does not treat packet losses as congestion, errors over the network link will not cause the UDP sender to reduce its sending rate.

Figure 2-2 shows an example of a RLNC transmission that suffers from a burst of packet losses due to some change on the network link condition.

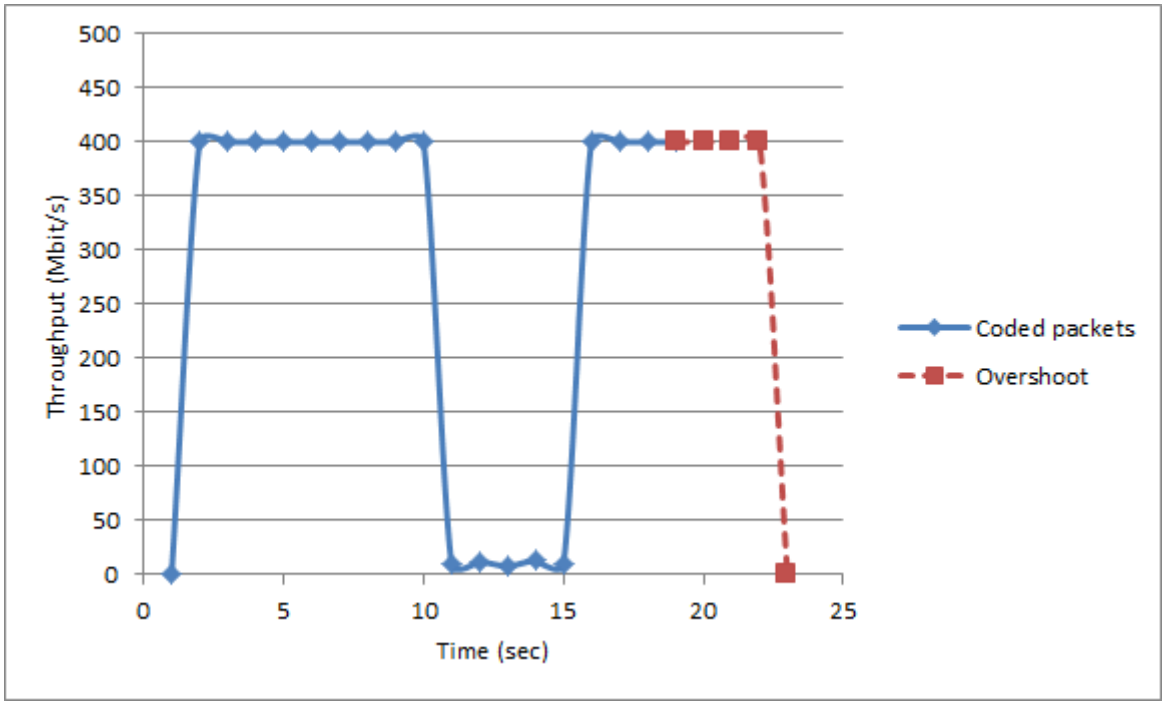


Figure 2-2: **An example of an RLNC transmission that suffers from a burst of packet losses. The sender has expected some losses and overshoots by sending extra packets to be used by the receiver to extract the original data.**

The send rate in Figure 2-2 is not affected by the losses as the transmission protocol used is UDP, which is not affected by the condition of the network. The overshoot graph resembles the amount of extra coded packets that are sent to make up for the lost data. The transmission of extra packets will be referred to as overshooting. By expecting the losses, the sender can send extra packets and let the receiver solve its packet loss problems independently. A benefit of using this method is when data is broadcasted out to several clients who can solve their own independent packet loss problems by using the same coded packets.

According to Steven Max Patterson's article [14], the company Code On released data that shows their implementations of RLNC to perform between 13% and 465% faster than the industry standard Reed-Solomon encoding in Storage Area Network (SAN) erasure application testing. Code On's RLNC implementation is tested against Intel's Reed-Solomon implementation called ISA-L, and an open source implementation of erasure correcting codes for storage applications named Jerasure. The results show that Code On's implementation outperforms both ISA-L and Jerasure on many of the test cases. It also shows a significantly faster performance on file sizes above 30MB. Patterson states that the increase in the performance of multicore processors and network parallelism has led to new possibilities to develop more advanced encoding schemes that utilize matrix operations and linear algebra.

2.2.2.1 Steinwurf APS

Steinwurf APS is a research company in Aalborg, Denmark. The company is a licensee of Code On, and was founded in 2011 by researchers from MIT and Aalborg University. Steinwurf's main research area is implementation of different kinds of coding solutions for reliable data transfer. Steinwurf works on fast implementations of network coding written in the programming language C++.

In order to promote network coding being used to solve network transmission problems, Steinwurf has created a library named Kodo [15]. Kodo is an open source network coding library created in C++. Its purpose is to enable the study of network coding algorithms in practical settings. I acquired a license for the Kodo library for this thesis project, for the purpose of creating an application to evaluate RLNC.

2.2.2.2 Kodo network coding library

The current version of Steinwurf's Kodo network coding library has been selected as it supports RLNC. This library was used to create the software used in this thesis project. This sub-section provides the reader with information about how the Kodo RLNC implementation works, especially in terms of how it was used for this thesis project. The parameters and the encoding and decoding procedure used in Kodo are explained.

The library provides several functions for RLNC, which can be used in special situations to improve performance. However, for this thesis project only the encoding and decoding functions were used in the evaluations that have been conducted.

Kodo uses Steinwurf's Finite Field Library (Fifi library)* for finite field arithmetic. The finite field used in the tests of this project is the binary field of size 2 - as this was the only field size implemented for the file encoder implemented in Kodo. A number of parameters need to be defined in order to encode and decode data, specifically: the symbol size and the generation size.

The symbol size determines the amount of data to be used for each encoded symbol. The data used for the symbol is expressed as a vector in the finite field. The size of the vector is increased by increasing the symbol's size. The size of this vector and the size of the symbol are preferably selected to fit (together with the transport header and IP header) within the MTU of the transmission path†, in order to reduce the risk of packet fragmentation.

The generation size defines how many symbols should be used for one generation. A generation is defined as the set of symbols used for one block of data. Computational cost increases when encoding larger blocks of data. The amount of data in a file is usually too much for network coding to be efficient, hence each file is divided into a set of generations.

A generation is seen as a matrix of vectors, expressing all the data in the generation. An example of a generation matrix in Kodo is shown in Equation 1.

* http://steinwurf.com/fifi/

† The MTU is usually 1500 bytes for most user networks. However, it can increase to, for example 9000 bytes, when using jumbo-frames on an Ethernet.

Equation 1: **The matrix of the generation size *g* number of symbols. The symbols are expressed as vectors of data in the finite field.**

$$M = \left[\vec{m}_1; \vec{m}_2; \ldots; \vec{m}_g\right], \text{where } \vec{m}_i \text{ is a vector in } F_2$$

2.2.2.2.1 RLNC encoding

An encoded symbol in RLNC consists of a random linear combination of all the symbols in a generation. It is computed by applying linear algebra to multiply the generation matrix with a randomly chosen coding vector. The coding vector consists of the generation size number of coefficients, with each coefficient representing a symbol in the generation. The encoded symbol is computed by multiplying the generation matrix with the generated coding vector, as shown in Equation 2.

Equation 2: **The computation of an encoded symbol between the generation matrix and the coding vector.**

$$\vec{x} = M * \vec{v}, \text{ where } \vec{v} = \left\{v_0, v_1, \ldots, v_{g-1}\right\}$$

The equation results in a vector of the same size as a symbol, but represents a combination of all the vectors in the generation. The coded symbol and the coding vector need to be known to the receiver in order for it to be able to decode the block of data. A *coded* packet, consisting of the encoded symbol and the coding vector, is sent to the receiver.

2.2.2.2.2 RLNC decoding

The receiver of the transmission gathers coded packets until a generation can be decoded. The vectors of a coded packet are stored in two different matrices, see Equation 3.

Equation 3: **The vectors of a coded packet are stored in two separate matrices.**

$$(\vec{x}, \vec{v}) \rightarrow \vec{X} = \left[\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_g\right], \vec{V} = \left[\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_g\right]$$

Decoding is done by solving a linear system of equations by applying linear algebra. After acquiring enough encoded symbols and information about the encodings, one can find the generation matrix using Equation 4.

Equation 4: **By applying linear algebra to solve the linear system of equations, the receiver can compute the generation matrix *M*.**

$$\vec{X} = M * \vec{V} \quad \rightarrow \quad \vec{X} * \vec{V}^{-1} = M$$

The requirement of receiving **all** symbols in a block changes into receiving **any** g number of encoded packets. For this reason, this method is robust to errors in symbols in an encoded packet.

2.2.2.3 Problems with network coding

One of the problems with RLNC is that it is dependent on UDP, which does not support congestion control. For this reason, RLNC based implementations need to support some form of congestion control, but no one has yet found an efficient way of doing so.

Unfortunately, combining network coding together with TCP is infeasible, as the receiver of a segment **cannot** send an ACK to the sender before the receiver has been able to decode the segment. Furthermore, this decoding **cannot** be done instantaneously when a packet arrives, as decoding can only be done when the receiver has received sufficient evidence to extract the correct contents of the encoded packet. This would cause delays in the different TCP connections passing through a network.

2.3 Related work

Network coding is not widely used in many current implementations. However, companies such as Steinwurf, among others, are putting a lot of effort into creating products and improving network coding schemes for future use. This section summarizes some of the other research that has been done in this area, as well as other research to reduce the impact of changes in the characteristics of network links on the network goodput, i.e., the application layer throughput.

2.3.1 XORs in the Air: Practical Wireless Network Coding

The paper "XORs in the Air: Practical Wireless Network Coding" introduced an architecture named COPE [12], which is rooted in the theory behind network coding. COPE was created to increase throughput in wireless mesh networks. COPE inserts a new layer in-between the Media Access and Control (MAC) sublayer and the IP layer. This new COPE layer detects opportunities for coding to combine packets. To be more specific, it utilizes three techniques: opportunistic listening, opportunistic coding, and learning neighbor state. These techniques are based on continuously finding new ways of encoding packets and making use of nodes close to the transmission's path. Additionally, COPE makes use of the broadcast nature of wireless networks to spread and store coded packets in neighboring nodes.

Their results shows that COPE can increase a network's throughput from just a few percent to several folds. The improvement depends on the amount of congestion in the network, the traffic pattern(s), and the transmission protocol used.

2.3.2 MATIN: A Random Network Coding Based Framework for High Quality Peer-to-Peer Live Video Streaming

MATIN [16] is a Random Network Coding (RNC) framework for efficient Peer-2-Peer (P2P) video streaming. The article addresses the problem that RNC adds high transmission overhead to the network, in the form of coding vectors. Another problem it addresses is the high computational complexity that must be implemented in the network hosts, due to the linear algebra computations.

MATIN proposes a solution to reduce the size of the coding vectors into only one coefficient, instead of one for each symbol in the generation. This reduces the overhead of the coded packets, as well as the computational costs of inverting the matrices of coding vectors. This proposed solution outperforms other RNC implementations that use Gauss Jordan elimination algorithms in the decoding procedure.

2.3.3 TCP Reaction to Rapid Changes of the Link Characteristics due to Handover in a mobile Environment

Ronquist [17] evaluates the effects of TCP's behavior when experiencing sudden changes in link characteristics. Problems are recognized when TCP experiences a handover from a fast link to a slower link. The investigation performed shows several spurious TCP timeouts, which trigger unneeded retransmissions. These result in a degradation of TCP's performance.

Problems also occur when handovers are performed from a slow link to a faster link. The problem is identified to be in the link layer buffer, which is not emptied fast enough and packets continue flowing on the slow link. This results in a underutilization of the newly available bandwidth.

The solution proposed for both situations are better adaptions to set an appropriate Retransmission Timeout (RTO) value. The conclusion of his research suggests that a small window size is favorable, as this can mitigate the negative effects from changes in the network link's characteristics.

2.3.4 RLNC throughput

Kim et al. [18] introduces Coded TCP (CTCP). CTCP is a protocol that combines RLNC with TCP as its transport protocol. The author points out TCPs problems to recognize packet losses due to interference on the link, which causes the protocol to think the network is congested and lowers the sending rate. CTCP eliminates that problem as the packets are encoded and the receiver does not need to know that **each** packet arrived, but that **a** packet arrived. For the case when the network is really congested, CTCP reacts to the TCP timeout instead and lowers the sending rate accordingly. However, CTCP does not cover the problem when a **burst** of packet losses occur to a change in network link characteristics. CTCP proves to significantly increase the throughput in networks with frequent single random packet losses.

Muhammad et al. [19] performs an investigation over several TCP versions when used in Performance Enhancing Proxy (PEP) architecture scenarios. The experiments were carried out by simulating a gateway connected to a Geostational Orbit (GEO)-repeater satellite to reach satellite terminals. The terminals are connected to end-host TCP clients. The throughput of the various TCP protocols was greatly reduced by the packet loss probability simulated over the satellite links. A solution was proposed to use RLNC together with the various TCP versions to reduce the impact of random packet losses. The results show that the proposed solution greatly increased the throughput of all of the TCP transmissions when single packet losses randomly occur.

2.3.5 UDP based file transfers

Several solutions have been developed to use UDP as the transport protocol to send files. Many of the solutions are based on using hybrid implementations of transport protocols, often by using TCP for control over the transmission and UDP for actual data transfer. Implementations like Hybrid Secure Copy Protocol (HSCP) and Tsunami UDP are examples of these hybrid protocols. For HSCP however, it should be noted that it is based on the original SCP protocol, with changing the data transfer into using UDP. A limitation of SCP is the internal buffer size of 1MB, which severely limits the performance over Wide Area Networks (WAN).[*]

Kumar et al. [20] explains how high speed reliable UDP can be achieved by sending bulk transfers of UDP packets. A bitmap can be sent via the TCP control connection to notify which packets were received. The packets that were not received are then re-sent, and an updated bitmap are sent via the TCP controller. This procedure is then repeated until all the packets are received at the receiver. Another important factor that needs to be considered is at which rate UDP is supposed to send. This rate is the bandwidth of the bottleneck link. The author proposes tools like Iperf and Netperf to measure the bottleneck bandwidth. However, unnecessary data are transmitted to the network by using these tools, which increases the possibility of network congestion. The network route can also change during the transmission, which adds the need to periodically check the bottleneck bandwidth. This is an aggressive method of checking the current state of the network.

[*] https://fasterdata.es.net/data-transfer-tools/say-no-to-scp/

2.3.6 DCCP

A problem when developing UDP based file transfer protocols is implementing suitable congestion control algorithms. Datagram Congestion Control Protocol (DCCP) was introduced by Kohler, et al. [21] as a solution to this problem.

DCCP aims to add the minimum mechanisms necessary to implement congestion control functionality. To obtain this, the Kohler, et al. identified the need for two mechanisms: sequence numbers to identify lost packets and a feedback channel to obtain congestion information. DCCP's congestion control implementation uses, much like TCP, the sender's congestion window to reduce unacknowledged data outstanding in the network. In contrast to TCP, unreliable transfers cannot use a cumulative acknowledgement in each packet to indicate which packets were received. Several approaches to implementing these mechanisms are presented, including transmitting acknowledgment information by a vector of ACKs to the sender. They present DCCP as a work in progress.

3 Methodology

This chapter contains a description of the methodology used in the project. The research in this project is divided into three different stages:

1. **Literature study** – The research began with a literature study to acquire necessary background information about the different topics. Information about TCP and its different implementations were acquired, as well as the known problems of TCP based on earlier research. The information was beneficial to create test scenarios in which TCP would have problems. Information about network coding was acquired to understand how it could work as a solution to the proposed problems. Based on the information gathered, the choice of evaluating the RLNC scheme and its benefits was made. Apart from the background information about TCP and network coding, information about how to set up a suitable testbed for this thesis was acquired. Selecting the appropriate operative systems to use and finding tools to emulate network links were part of this stage.
2. **Testbed implementation** – This stage consisted of implementing a testbed that could emulate the chosen network scenarios in an end to end network environment. It also included developing a system to transmit a file using RLNC over the emulated network. This step took the largest portion of time of this thesis project. It was difficult to detect and to understand the system properties that affected the network performance. A lot of time was spent optimizing the testbed to reach 10Gbit/s (due to limitations of the network card). It also took time to define suitable network scenarios for the measurements. Finally, measurements with the different TCP algorithms and RLNC were performed.
3. **Analysis and verification** – The final part of the project consisted of a comparison of the results from the measurement and a presentation of a proposed solution. The results of the measurements were analyzed to find and discuss the strengths and weaknesses of the two methods. Based on this analysis, a new solution was proposed. The limitations and the reliability of the measurements were commented on in terms of how they affected the results.

4 The testbed

This chapter gives a detailed explanation of the experimental testbed used for measurements and the software implemented within this project. Each of the different parts of the testbed is explained in terms of hardware, purpose, and the tools needed. At the end of the chapter the limitations of the testbed are discussed and evaluated, and the maximum possible transfer rate that can be realized with this testbed is measured.

4.1 Overview and design

An overview over the design of the testbed is presented in Figure 4-1. This testbed can be divided into three different parts: the client(s), the network, and the server. The testbed emulates an end-to-end connection between a client and a server over different transmission links. The values for packet loss and RTT over these transmission links are changed in the different tests. Switching between different transmissions links is used to emulate handover scenarios.

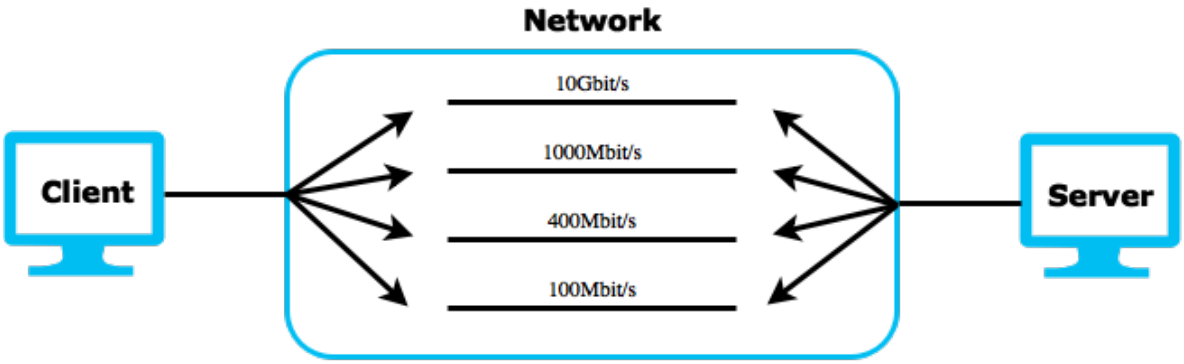


Figure 4-1: **Overview of the testbed design. The network contains four different network links with fixed maximum data rates.**

4.2 The equipment

The hardware used in the testbed was selected to provide a stable system for a maximum throughput of 10 Gbit/s. The client, network, and server parts are realized by three separate physical machines. These machines are connected by CAT6A snagless shielded 4 pair network cables, to form an end-to-end network. The machines were *Dell Precision tower 5810* computers. The specifications for the most significant hardware in the testbed are presented in Table 4-1.

Table 4-1: **The specifications for the hardware for each of the three machines used in the testbed.**

Computer make and model	Dell Precision Tower 5810
Central processing unit (CPU)	Intel® Xeon® Processor E5-1630 v3 (4 cores, 3.7 GHz, supports Intel Turbo Boost technology, hyperthreading, 10 MB of cache, 140 W)
Hard disk drive (HDD)	500 GB 7200 RPM
memory	64 GB 2133 MHz
Network interface	Myricom-10G PCIE2-8c2-2T

4.3 Client and Server Node configurations

The client and the server machines are running Ubuntu 14.04 with Linux kernel version 3.16.0-37-generic. The choice of using Ubuntu as the operating system was based on it offering a user friendly environment, the set of TCP congestion control algorithms implemented, and the possibilities to configure the system for best achievable performance based on hardware limitations.

The TCP congestion control algorithms implemented in the Linux kernel 3.16.0-37-generic are TCP Reno and Cubic TCP (see Sections 2.1.3.1 and 2.1.3.2). However, the default settings of the Linux kernel are not optimized to reach throughputs of 10 Gbit/s. For this reason the file */etc/sysctl.conf* was configured to change a number of system parameters. The final configuration of these parameters in sysctl.conf is shown in Figure 4-2. The net.core.* parameters are used to increase the socket buffer sizes, while the net.ipv4.* parameters are used to increase the buffers for the respective protocol. The vm.swappiness value is reduced from its default value of 60, to reduce the aggressiveness of memory swapping; hence, possibly reducing the rate of page faults. The system's performance increases if sufficient memory is available. The net.core.netdev_max_backlog parameter sets the maximum number of packets allowed to be queued when the interface receives packets faster than the kernel can process them (this parameter is related to the problem described in Section 4.6.1). Most of these parameters were recommended configurations from Myricom [22].

```
#UDP & TCP OPT
net.ipv4.udp_rmem_min = 8192
net.ipv4.udp_wmem_min = 8192
vm.swappiness = 10
net.core.optmem_max = 40960
net.core.wmem_default = 536870912
net.core.rmem_default = 536870912
net.core.rmem_max = 536870912
net.core.wmem_max = 536870912
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 87380 16777216
net.core.netdev_max_backlog = 250000
net.ipv4.udp_mem = 65536 16777216 16777216
```

Figure 4-2: The configured parameters in sysctl.conf on the Ubuntu hosts.

Configurations of the processors were made using each machine's basic input/output system (BIOS). The configurations were made to *disable* virtual cores (i.e. disabling hyperthreading), as the four physical cores are by default split up into eight virtual cores. To increase the reliability of CPU load measurements in the experiments, single thread per core performance needs to be ensured. As Intel explains in their article "Performance Insights to Intel® Hyper-Threading Technology" [23], hyperthreading trades thread processing latency for multitask throughput. A single process's performance can decrease as the available resources for a virtual core is less than of the underlying physical core.

CPU frequency scaling was *disabled* to increase the performance of each individual core. CPU frequency scaling means that the operating system increases/decreases the CPU frequency in order to save power. It is implemented and enabled by default in the Linux kernel. This was *disabled* to ensure the highest possible performance of the CPU at all times.

4.4 Configuration of the network part of the testbed

The computer that realized the emulated network is running FreeBSD 10.1 as its operating system. FreeBSD is an operating system which provides advanced networking features that were suitable for this project. One of the tools that influenced our choice of using FreeBSD was dummynet [24]. Dummynet is a tool added to the FreeBSD kernel. Dummynet is used in cooperation with the operating system's firewall ipfw [25] to emulate the different network links that we wanted to utilize in our test scenarios.

The network links are simulated by creating dummynet pipes. A dummynet pipe can be configured to have a limited bandwidth, a delay to simulate latency over the link, a packet loss ratio, and a queue size. Links are emulated by the pipe intercepting the packets in the network stack and trying to add them to a fixed size queue. This queue acts as a network node buffer, which is drained at rate based on the configured bandwidth. When a packet is removed from the queue, it is delayed for the configured amount of time before it is re-injected into the network stack. The packet loss ratio was set to drop packets based on non-congestion causes for packet loss. The pipes are added in the firewall's rules and are applied when a packet matches a specific rule. A packet is forwarded to the dummynet pipe when a match is found.

Several different pipes were implemented in the network. One pipe was used to simulate the connection between the server and the client. Handovers were emulated by swapping pipes during transmission. This was done by using the *set* property in ipfw, which can assign certain firewall rules to a particular set. Scripts were used to enable and disable sets after configured periods of time.

The values that were used for latency and packet loss ratio were provided by Ericsson from measurements done in networks in North America. The values were calculated from a Cumulative Distribution Function (CDF) based on their measurements of RTT and packet loss. The values used for the different pipes are presented in Table 6-1 on page 29.

4.5 Measurement tools

This section describes the different tools that were used to measure the data transmitted over the network. The section's purpose is to give the reader information about how the tools operate and possible errors the tools might generate.

The application that was used to send files over the network using the different TCP algorithms was the Secure Copy Protocol (SCP) [26]. SCP is used to copy files between network hosts. It begins by creating an SCP session between the hosts, and then opens a TCP connection to send the file.

The Bandwidth Monitor-Next Generation (bwm-ng) [27] application was used to measure the throughput to the client. Bwm-ng reads from the file /proc/net/dev in Ubuntu, which contains real time reports from the operating system (OS) about the amount incoming and outgoing of data on each network interface. Bwm-ng can export this data in *Comma Separated Values* (CSV) format. This data was subsequently used to generate graphs for further evaluation. However, the application was limited to reading the file every 0.5 seconds, which did not provide accurate results when examining a small time interval.

The iperf3 [28] tool was used to determine the maximum throughput of the testbed. It operates as an end-to-end application, with a sender and a receiver. The application generates random data to transmit over the network. The user can define which transmission protocol to use, as well as the amount of time that the transfer should last. The reason for using iperf3 to determine the maximum throughput was because of its ability to transmit both UDP and TCP traffic. However, the application had some problem that made it unsuitable for the experimental phase of this project.

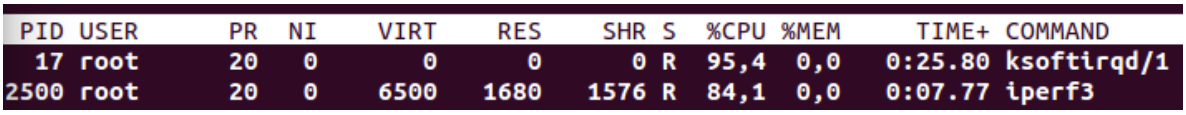
For example, the application did not show the correct throughput in its reports. In contrast, bwm-ng, reading the OS's counters associated with each of the interfaces, showed a higher throughput than what iperf3 reported. As iperf3 can also be used to transfer files using TCP, it was used to test the performance of a file transfer of a video clip where all the frame data was scrambled. TCP is a reliable protocol; hence, the resulting file should be correct after a transmission. This was checked by computing an MD5 checksum of the original and received files.

4.6 System limitations

Transmitting data at 10 Gbit/s puts a lot of load on a system. High quality hardware is required to support this data rate. A lot of time was spent during this thesis project on configuring the testbed to support as high a throughput as possible. The final configurations that were used had a throughput limitation of 9.8G bit/s for TCP connections and 7.3 Gbit/s for UDP connections.

4.6.1 Interrupt request handler

In Ubuntu, interrupt requests are issued to the OS when packets arrive at the network interface. The OS then interrupts what it was doing and starts to handle the interrupt. When too many interrupts are issued within a short period of time, the OS cannot keep up with these interrupts; hence a process to schedule these interrupts is invoked. This process is called *ksoftirqd* [29]. Figure 4-3 shows a snapshot of the CPU utilization when receiving a 7 Gbit/s stream of UDP packets.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
17	root	20	0	0	0	0	R	95,4	0,0	0:25.80	ksoftirqd/1
2500	root	20	0	6500	1680	1576	R	84,1	0,0	0:07.77	iperf3

Figure 4-3: **CPU utilization of the interrupt request scheduler and the receiving application iperf3 using the "top" command in Ubuntu when transmitting 7Gbit/s of UDP packets.**

The CPU usage of ksoftirqd reached over 95% when sending a UDP stream at 7 Gbit/s through the testbed. This means that the system could not handle all the interrupts in time and the system is under a heavy interrupt processing load. Some network interface can reduced this effect by not issuing an interrupt for each packet, but rather issuing interrupts for several packets at the same time*. This functionality was implemented for TCP in Myricom's network cards, but unfortunately not for UDP [30].

This limitation only affected UDP transmissions. By assigning the receiver process and the ksoftirqd to execute on different cores, a maximum throughput for UDP of 7.3 Gbit/s was achieved. Due to the use of interrupt coalescing TCP's throughput was not affected by interrupt request scheduling. A TCP connection reached a stable 9.8 Gbit/s throughput when there was no emulated delay or packet loss in the network. The maximum throughput achieved by both UDP and TCP are presented in Figure 4-4.

* This is called interrupt coalescing.

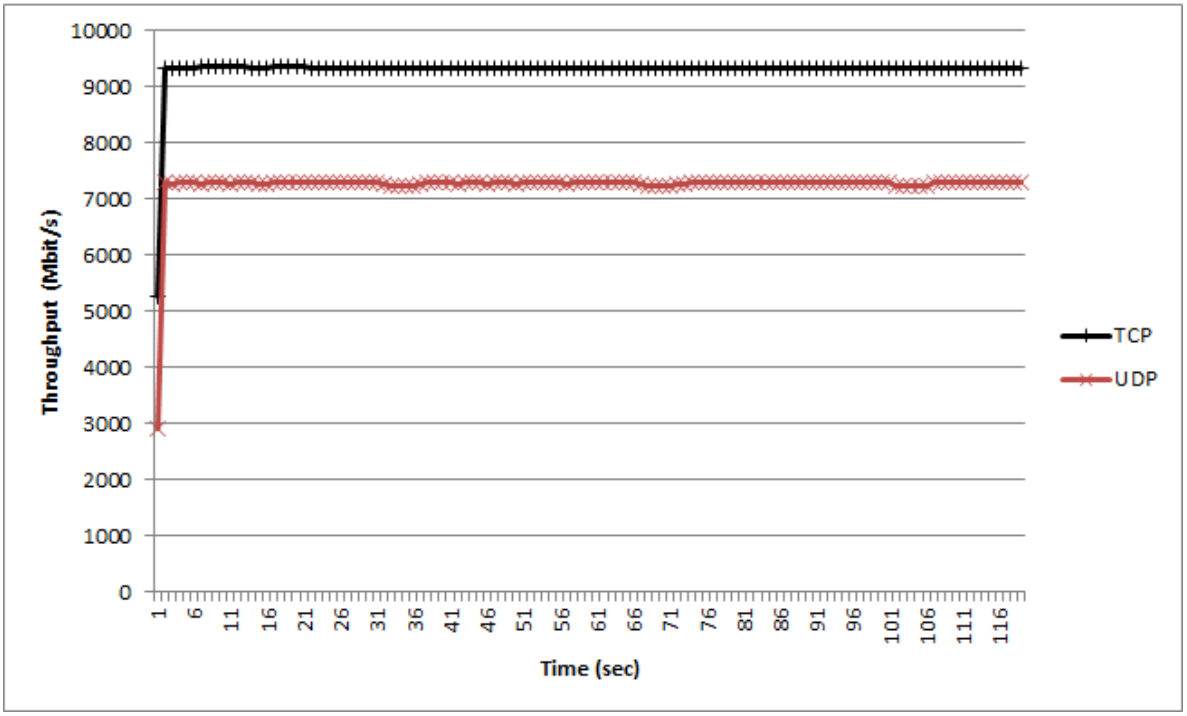


Figure 4-4: **The maximum TCP and UDP throughput measured on testbed hardware.**

4.6.2 Memory throughput

Another factor that is important to consider when transferring a file between computers is the memory read and write data rates. The maximum rate of transferring data to and from the main memory was measured as 1700 MB/s. This was measured by using the *dd* command in linux[31] to create a 10GB of random data on the main memory. The command provides information about how the file is written to the directory specified.

4.6.3 Disk throughput

The hard drives used in the testbed had a rotational rate of 7200 RPM, which limited their throughput when transferring files, due to the time it takes to write the blocks of the received file to the hard drive. To remove this limitation, RAM memory was allocated to store the files.

5 Implementation of a Network Coding application

This chapter contains a description of the software that was created to implement RLNC for testing in the testbed. The chapter starts with an overview of the implementation. Then the two end-host applications used in the final testing are described. Then a description is give of how a thread was implemented to maintain a TCP connection to provide the server with feedback from the client. The chapter ends with a summary of the implementation.

5.1 Overview of the implementation

A system to enable network coding was implemented for testing and evaluation of network coding, specifically RLNC. The system contains a “server” and a “client” application that act as a sender and a receiver. Kodo was used for encoding at the sender and for decoding at the receiver.

The purpose of the software was to send a video file through the network using RLNC. The video file’s format was of type YUV4MPEG2 [32], which will subsequently be abbreviated as Y4M. This format contains a header, followed by an amount of video frame data. The choice to use this kind of file was mainly because of the possibility to play the file even if part of the file is missing. This enables us to see how the packet losses affect the video after it had been transmitted. A video file of 1.1 GB size was chosen for the experiments. This file size is typical of a video file that a potential user would download today.

The system was able to use different settings for the encoding in terms of both symbol and generation size. By setting different values for these parameters, I could evaluate how different settings could affect the throughput and the load on the CPU. There are four different parameters that can affect the performance of the system:

1. The *field size* affects the computational complexity of the data representation. Using too large a field size might slow down the encoding and decoding. The system used the Kodos file encoder, which only supports a binary finite field of size 2; hence, the field size was not a problem.
2. The *symbol size* sets the amount of data used for each symbol in bytes. When used in network applications the size can be set to avoid the coded packets becoming larger than the MTU. As fragmentation will occur if the network packet is larger than the MTU; hence, the number of packet losses and packet handling operations will increase.
3. The *generation size* determines how many symbols will be used to represent a given amount of the files data, i.e. a block (see Section 5.2). Increasing the generation size increases the computational complexity of the encoding and decoding. However, it reduces the need for a larger field size and it reduces the protocol’s overall complexity by reducing the number of different generations needed for sending the whole file.

Before the server can start sending packets to the client, the system needs to come to a consensus about the file size, symbol size, and the generation size. These values were predefined in the first version of the system. Adding functionality to change these sizes is discussed in Section 5.4.

5.2 The server application

The server application acts as a distributor of data. The purpose of this application is to encode data and send it to a client using UDP. The server needs to inform the client about the encoder's settings in order for the receiver to initialize and build a suitable decoder. The settings have to be adapted to the client's hardware limitations in order to avoid overwhelming it with too high a computational load.

The file to be sent is divided into several blocks of data to reduce the computational effort required while encoding. The block size to use for a file is determined by three parameters: the symbol size, the generation size, and the field size. The server iterates through all the blocks of the file and for each block builds an encoder based on the current block. The data is transformed into vectors once the encoder for the block is built.

For each generation of the transmission, the server iterates through a loop that creates a randomly coded packet for the current generation and sends it over UDP to the client. The loop iterates for the expected number of coded packets that need to be sent to make up for the packet loss rate of the network. This value has to be set such that the receiver always receives a sufficient number of packets to be able to decode each block. Unfortunately, no feature yet exists to detect the actual packet loss rate; hence the server will have to set the value large enough to make up for a potentially high packet loss rate. This issue is further discussed in Section 6.5.

5.3 The client application

The client application acts as the receiving host by listening for UDP packets on a specified port. It gathers coded packets sent by the server, until a generation can be decoded. Decoding can be done when enough coded packets have been received for a generation. If not enough coded packets for a generation were received before a new generation is received, then the client skips the previous block of data and starts to gather information about the new one. As it is crucial that enough coded packets always be received for the client to decode every block of the file a future extension of this client and the server should indicate when the client has not received enough packets to decode the current generation, so that the sending application could send more encoded packets.

The client starts by initializing a decoder by using the same parameters as the server used when it initialized its encoder. The client reads the block number of each coded packet, which is sent as the first bytes of the coded packet. This information is required for the client to know when a new block is received. If the decoding is completed before a new block arrives, then the client simply waits until it reads the block number of a new block. Again a future extension would be to inform the sending application of this, so that it no longer needs to send encoded packets for this block.

This simple implementation works sufficiently well for testing network coding over a network link, based on predefined values at both the client and the server. However, additional implementations are needed to make it work as intended for use as a real application. The parameters of the encoding/decoding have to be decided on based upon the needs of the client. A TCP controller thread is introduced in the next section to enable further extension of this implementation.

5.4 TCP controller thread

A TCP controller thread maintains a TCP connection between the client and the server for as long as the transmission is ongoing. This connection is initiated by the client when it requests resources from the server. The server responds over this connection with parameters about the encoding to be used by the client.

The thread (and its associated TCP connection) enables the client to send feedback to the server about the transmission. It can be used to initially determine the parameters for the transmission. The main feature that was implemented was to send information about when a block was decoded. This prevents the server from spending unnecessary time on encoding additional packets when the block has been successfully decoded. This also reduces unnecessary traffic on the network. However, when testing the implementation in the testbed network, the delay of this feedback to the server was too high for the thread to give any benefit. This occurs because the server could encode and transmit packets much faster than the time it took for the notification to reach the server[*]. It resulted in all the coded packets of a block being sent even if the decoder was done.

The thread can be improved to increase efficiency, reliability, and to make the transmission adapt more to the network. Information that could be sent with the control thread includes:

1. *Blocks that were not decoded* – The client could send feedback about those blocks that were not able to be decoded because of lost packets. The server would encode a small number of extra packets from the missing block and send it in parallel with the current block. This would make it possible for the receiver to decode the missing blocks, hence, increase the reliability of the transmission.
2. *Estimated packet loss rate* – The client can use the number of received packets to inform the server about how much redundancy is needed. This would reduce the amount of overhead needed for the transmissions, as the sender can adapt to the state of the network.
3. *Hardware limitations* – As the decoding puts pressure on the CPU, too high a throughput could cause the client to become overwhelmed, leading to packets being dropped by the client.

5.5 Summary of the design

Throughout this chapter the design of the system used to implement network coding in this thesis work has been described. The reader has been given a description of how each part of the system works. A description of how the implementation can be improved in the future by using a TCP controller thread has been described. Subsequent chapters provide analyzes of the results of measurements of the implemented software's performance in the testbed. This software implementation was designed for testing purposes only, and is not suitable as a real application due to the lack of congestion control.

[*] The time to encode a new packet was xxx ms, while the delay from client to server was xxx ms.

6 Analysis

This chapter presents the results of measurement collected when carrying out experiments in the testbed. Baseline tests using TCP are presented to show how a connection can act in realistic scenarios. The designs of these scenarios used to test RLNC are described. Finally, measurements when using TCP and RLNC are presented and compared. Different settings were used to cover different scenarios, as well as to compare the different settings used in RLNC. These settings are described together with each of the scenarios.

The different network link characteristics used in the experiments conducted as part of this thesis project were chosen to resemble network links used in actual networks today, as well as a 5G network link. The 5G network link for this project was defined to have the highest possible throughput possible with our hardware*, i.e., 9.8 Gbit/s for TCP and 7.3 Gbit/s for UDP. The delay for the 5G network link used in this project is defined to be 20% lower than for the LTE links. The MTU used in all test cases was 1500 bytes. This value is based on the maximum size of IP datagrams in Ethernet 2 frames [33], which is the most common MTU value that a user would experience today. However, higher MTU values are often supported while transferring data between data centers and in gigabit Ethernet networks.

All the network link bandwidths and RTTs used in this analysis are summarized in Table 6-1. A number of different packet loss rates were test-based, together with the different tests listed in Table 6-2.

Table 6-1: **The different emulated network links used in the testbed. These links were emulated using dummynet.**

Network link	Bandwidth	RTT (ms)
10G	10Gbit/s	46
1G	1Gbit/s	56
400M	400Mbit/s	56
100M	100Mbit/s	56

Table 6-2: **The six TCP baseline test scenarios. The handover columns show which network characteristics were swapped during the test. Each test starts with "Handover 1" and stops with "Handover 3". The network link specifications were shown in Table 6-1.**

Test	Packet loss rate (%)	Handover 1	Handover 2	Handover 3
A1	0	10G	100M	-
B1	0	100M	10G	-
C1	0	400M	100M	1000M
A2	0.006	10G	100M	-
B2	0.006	100M	10G	-
C2	0.006	400M	100M	1000M

6.1 Baseline tests

A set of tests consisting of different link conditions and handovers were defined to establish a good baseline for the project. This baseline resembles the link characteristics and handovers that a potential user could experience while using today's LTE networks. All six of the TCP baseline tests are shown in Table 6-2.

* This is measured as a network with 0 ms delay and 0% packet loss rate. The hardware limitations are explained in chapter "4.6 System limitations".

6.2 Design of the experiments

This section discusses the design of the experiments. It discusses the thoughts behind these experiments and how they resemble realistic user scenarios. Following this there are arguments about the limitations of these experiments, as well as comments on the different test cases. Finally, the limitations of the design of the RLNC tests are explained and discussed.

The baseline design should resemble how the system looks before the proposed solution is applied. In this thesis, the baseline should reflect the user's experience when using TCP over today's radio links. The situations that have to be considered are packet losses due to disturbances on the radio link and handovers between different radio network links. Ericsson provided the project with data from measurements, carried out in the North America, to best resemble real users' experiences with respect to network throughput. These measurements were carried out by probing the network and sampling statistics at measurement point at a base station, see Figure 6-1. The measure point measured the round trip time and packet loss rate by probing both the wired network, and the radio network. The values used in the thesis were calculated as the average of each of the wired and the radio network measurements. The average values for the wired and the radio network were added together to resemble the complete path of a potential transmission path for a user.

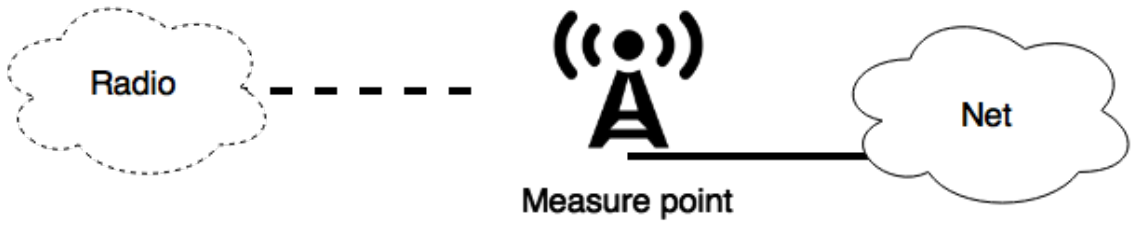


Figure 6-1: **An illustration of where the measure point was located. Data was taken from connections coming from both the network part and the radio part, to calculate CDF of RTT and packet losses.**

TCP treats all packet losses as indications of network congestion, but the Ericsson measurements were based on probing to identify the networks current states, in which most cases congestion was probably the reason for the packet loss. Therefore, this data did not provide accurate packet loss rate information.

Moreover, today's radio links use several functions in lower layers of the network stack to reduce the number of packets lost by retransmitting data so that the TCP peer correctly receives all of the segments. These functions are implemented to reduce the effect of lost packets due to random errors on the link's throughput. However, packet losses still occur as these functions cannot cover up all of the errors, but the rate of the observed errors is far lower than the actual error rate of the raw link. Unfortunately, these lower layer functions increases the delay of the transmission, while reducing the number of times that TCP thinks the network is congested.

A problem is that RLNC uses UDP to transmit packets and UDP has no congestion control implementation. Therefore, is unfair to compare TCP to RLNC, when TCP's low performance is mainly based on measurements where the network most likely was congested. Even attempting comparing TCP over different radio links is problematic, as the congestion based packet loss rate in the radio network is unknown. As a result the actual values for packet loss rates had to be estimated and, to some extent, these values do not match a real user's experience over a non-congested radio link.

The handover scenarios design were based on the proposed problem: TCP adapts slowly to changes in the characteristics of network links. Three different handover scenarios were defined:

1. From a high bandwidth to a lower bandwidth.
2. From a low bandwidth to a higher bandwidth.
3. From a medium bandwidth to a lower bandwidth, then changing to a high bandwidth.

These three scenarios are designed to uncover problems that might occur when a user experiences handovers.

As RLNC uses UDP as its transmission protocol, evaluating handovers with RLNC was impossible without further implementation work. As a result, RLNC was only tested *with* high packet loss rates, but *without* handovers. The packet loss rate chosen for tests with UDP was 1%. This value was chosen to show RLNC's performance in high packet loss network environments.

6.3 Baseline results

The following graphs show the results of the baseline tests as defined in Table 6-2. Handover are performed after 40 seconds intervals. In tests A and B, the handover is performed approximately at the 40 second marker. In test C, the handovers are performed at the 40 second marker and the 80 second marker.

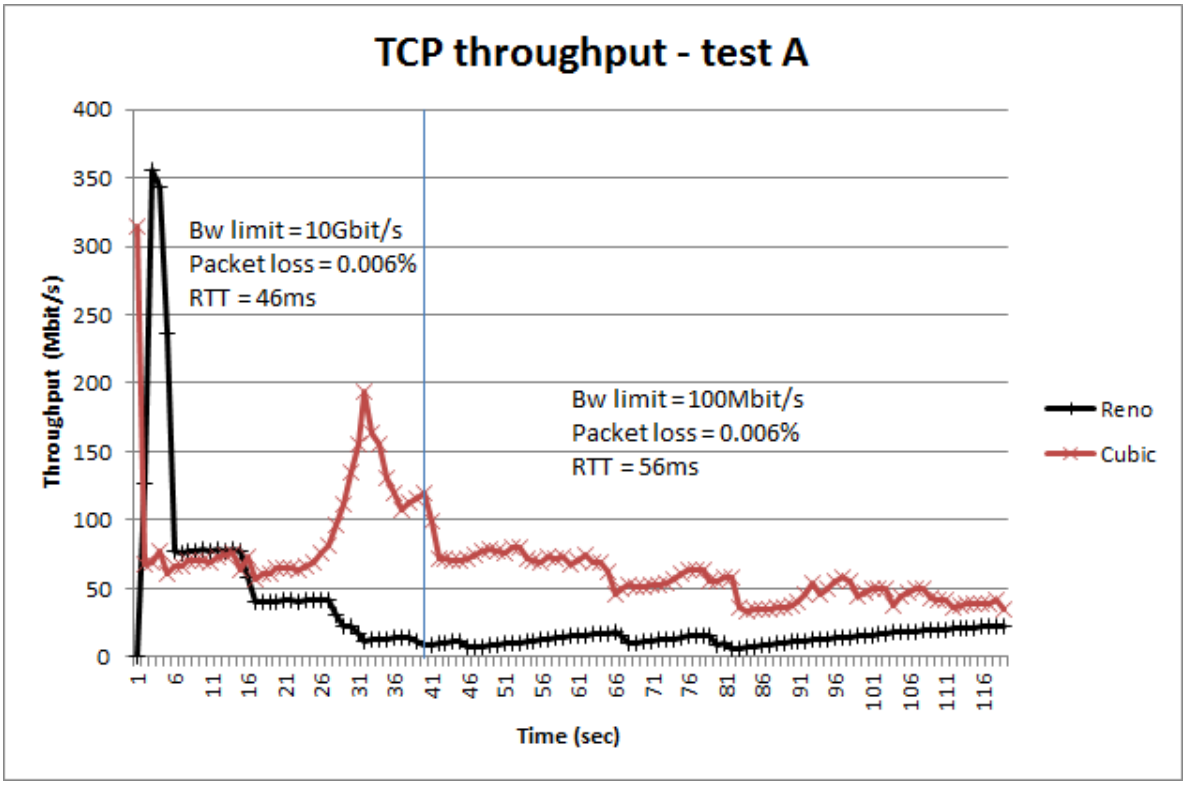


Figure 6-2: **Throughput as a function of time in baseline test A.**

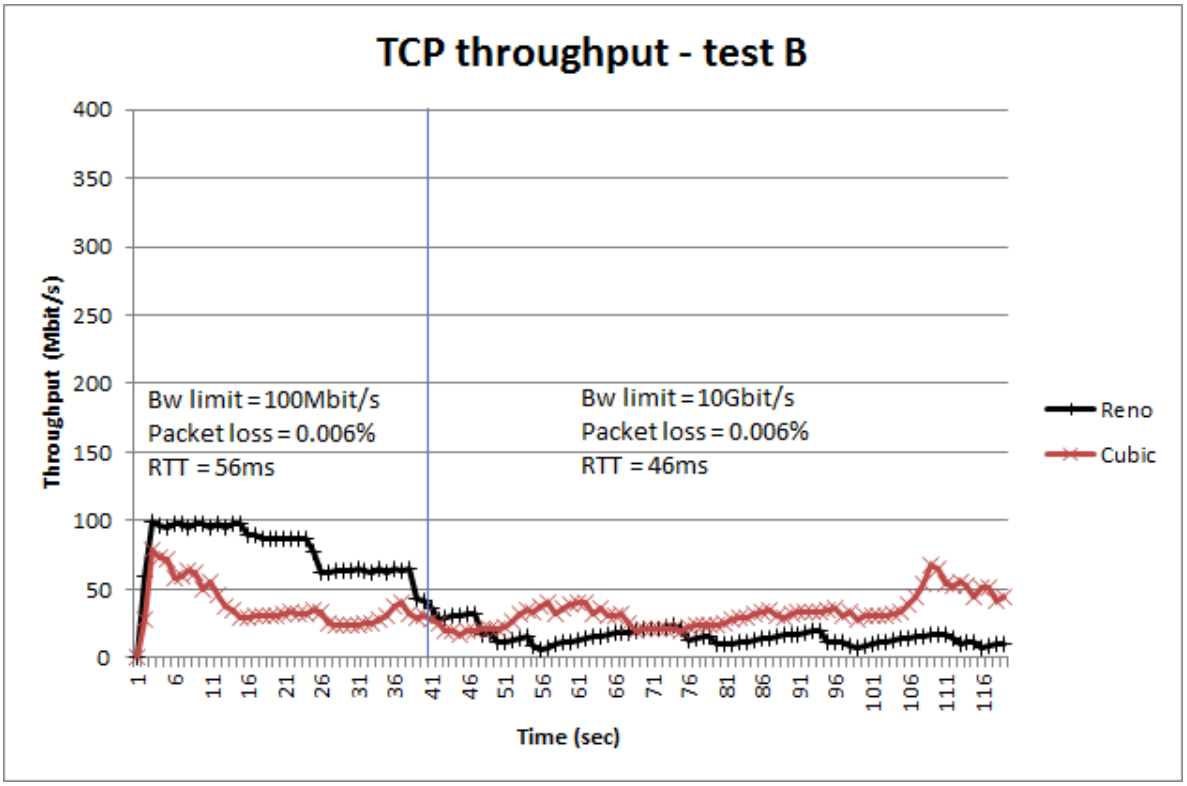


Figure 6-3: **Throughput as a function of time in baseline test B.**

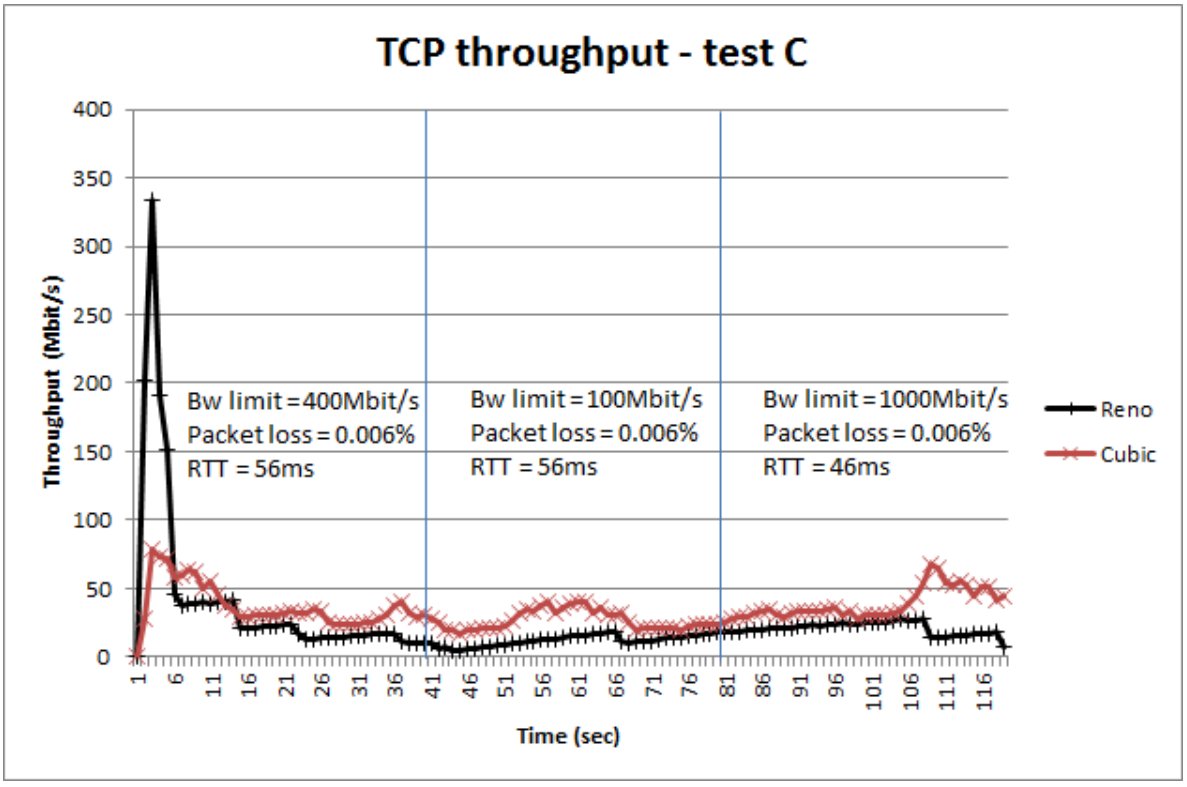


Figure 6-4: **Throughput as a function of time in baseline test C.**

6.3.1 Discussion of the baseline results for each of the test scenarios

In tests A and C, it is observed that TCP Reno reacts strongly to the packet losses when transferring a lot of packets over the high speed link. In test B, the reaction is less aggressive when less data is transferred in a short amount of time.

Test A shows the only scenario where any of the protocols throughput are high enough to react to the handover. Cubic TCP probes for a higher throughput after 26 seconds, and at the 40 second marker, packets get lost due to the reduction in bandwidth. The protocol adapts by going down to the previously highest throughput. In all tests it is observed that Cubic TCP serves with a more stable throughput than TCP Reno.

6.3.2 Summary of the baseline results

The measurements show that TCP's performance for average users with the estimated network conditions are often lower than 100 Mbit/s. However, this is most likely due to congestion issues in the network, and not entirely due to the packet error rate of the radio links. The average user utilizes too low a throughput to be affected by the high throughput handovers that were tested. However, the tests assume that the base stations participating in the handover are uncongested and can provide the user with the maximum available bandwidth.

6.4 Radio link measurements

To provide a fair comparison of TCP and RLNC, measurements emulating only the delays and losses associated with radio links were performed. The values used for TCP packet loss are based on values provided by Ericsson as measured at the radio side of the measurement point in Figure 6-1. These measurements were performed to understand a real user's potential experience, as the user might suffer from congestion in the radio network as well. Unfortunately, an estimate of how much packet loss was due to congestion was unavailable. As explained earlier the values used in the emulation may not be correct estimates of the radio network link's error rate, but they should be correct in terms of the user's experience of the radio networks when using TCP.

6.4.1 TCP performance over radio links

The following are graphs consider the same test scenarios as describe earlier, but with a packet loss rate that represents only the radio network's packet loss rate. These values do not account for congestion, but are based on what users currently experience in today's radio networks.

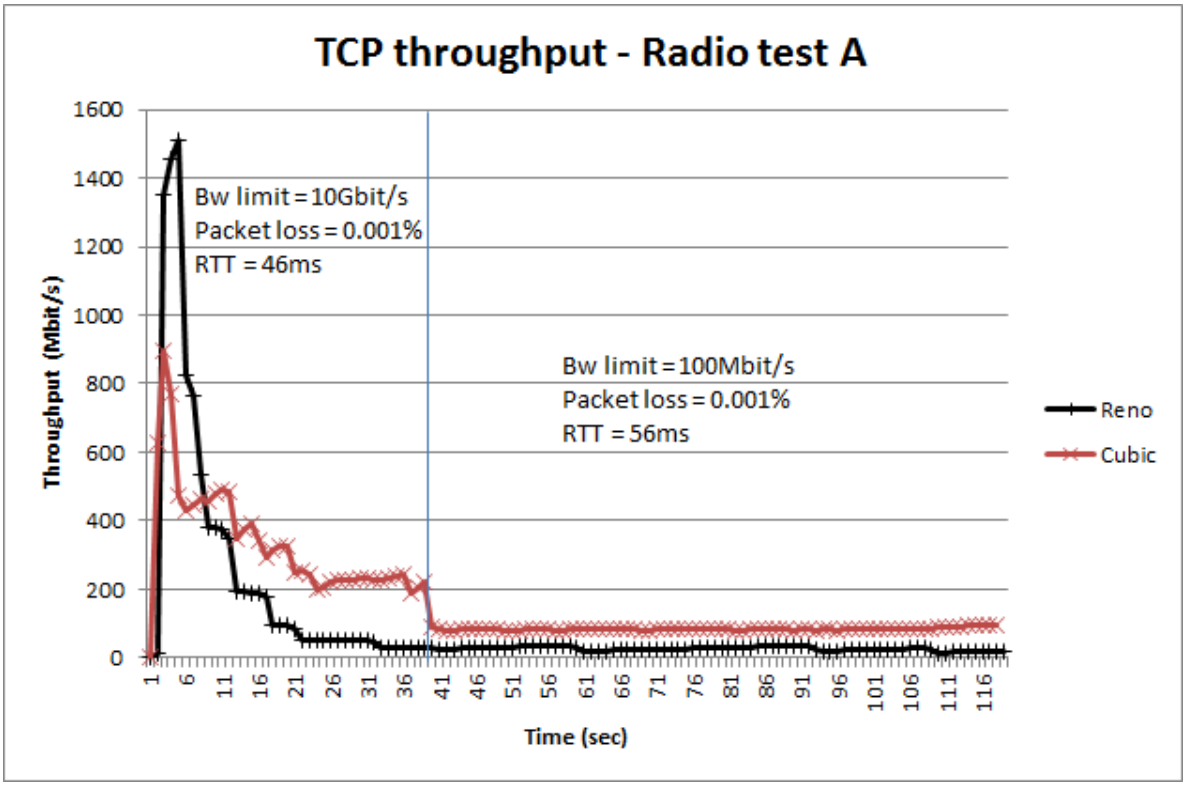


Figure 6-5: **Result from radio test A.**

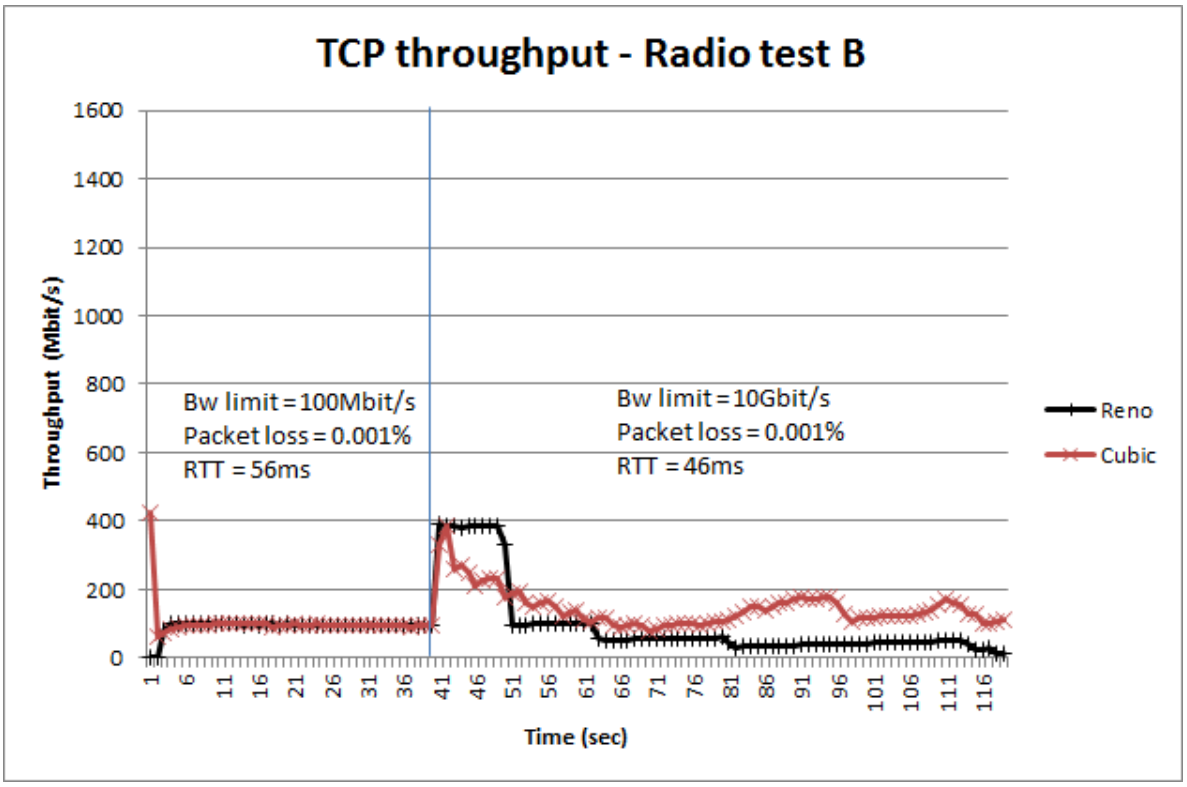


Figure 6-6: **Result from radio test B.**

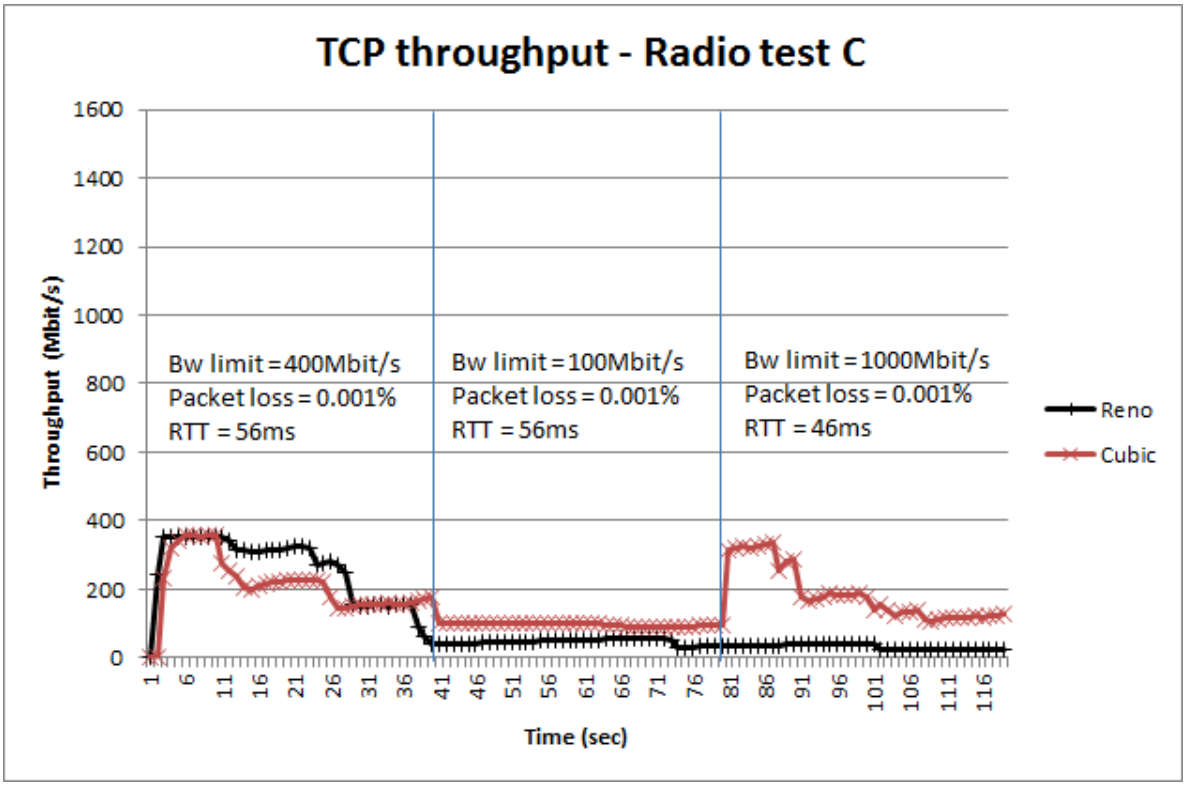


Figure 6-7: **Result from radio test C.**

6.4.2 Discussion of results from emulated radio link tests

It is observed that Cubic performs better than Reno in terms of adapting to the current available throughput. However, Cubic does not manage to fully utilize its probing mechanisms after a throughput reduction has occurred. In Figure 6-7, Reno never notices the handover after 80 seconds, resulting in a much lower throughput than Cubic.

6.4.3 RLNC performance over radio

RLNC was also evaluated in three tests over a hypothetical 5G network link, using three different generation sizes. The symbol size is optimized for each of the tests by adapting it to the MTU and the coding vector size required for the generations. The application specific data also had to be accounted for. The values used in the tests are presented in Table 6-3. The overshoot value for the tests was achieved when every block of the transmission could be decoded; hence, the whole file was received by the client.

Table 6-3: **Defines the three different tests used to evaluate RLNC. The "Extra size" is the size of the UDP header, the coding vector, and other potential data that the Kodo-encoder adds to the packet. The symbol size was chosen as large as possible for the generation size of the test, without letting the packet get fragmented on the link.**

Test number	Generation size	Symbol size	Extra size
1	200 symbols	1438 bytes	76 bytes
2	500 symbols	1400 bytes	114 bytes
3	1000 symbols	1338 bytes	176 bytes

The following results show measurements when using RLNC over the emulated radio network links. The packet loss rate value was increased to 1%, to provide a better evaluation of how RLNC would perform in high packet loss environments. The transferred file used in the RLNC experiments was described in Section 5.1. As UDP based transmissions do not adapt to the network, it was decided not to show throughput graphs of the RLNC tests in this chapter. RLNC throughput graphs can be seen in Appendix A.

Table 6-4: The average throughput, the number of extra packets required to decode every block, and the amount of CPU load required at the receiver running on a single core.

Test number	Average throughput	Overshoot	CPU load at receiver	Total data sent	Transmission time
1	5.57 Gbit/s	7%	83%	1.27 GB	1.799 sec
2	4.87 Gbit/s	3.8%	89%	1.27 GB	2.062 sec
3	4.45 Gbit/s	2.3%	92%	1.3 GB	2.325 sec

The results shown in Table 6-4 show a high possible throughput over a high loss network. However, the throughput does not consider congestion due to the different network nodes. These results include the transmission of the amount of extra data that was required to decode every block. The overshoot value is calculated by dividing the sent amount of coded packets by the generation size. The CPU load measured on a single core at the receiver is presented, as well as the total data sent and the exact time of the transmissions of the different tests. The total data sent are increased by the overhead required from the coding vectors and the amount of overshooting needed for the complete file to be transferred.

6.5 Discussion

The TCP results show that TCP has problems maintaining a high throughput even when the packet loss rate is low. The implementations of the protocol in Ubuntu can recognize handovers when a transmission does not suffer too much from random losses. However, Figure 6-7 shows a situation where TCP Reno has a problem recognizing a handover and as a result the throughput continues with a low linear increase in data rate. This results in low throughput for the user when they have made a handover to a base station that actually provides higher throughput.

The network coding experimental results show that RLNC based transmission methods can provide high throughput in networks with a high loss rate, as the transmission rate is not dependent upon the individual encoded packet losses. By combining the implementation used in the experiments with a proposed TCP controller thread, reliability can be provided by announcing the blocks that were not decoded to the sender. The sender can easily compute more coded packets for the un-decoded block and transmit them. The sender can implement additional functionality to increase the amount of overshooting while continuing the transmission. This solution might possibly be improved by using a UDP or Stream Control Transmission Protocol (SCTP) based control channel to increase the rate at which control information is received. Additionally, using RLNC for the control channel can increase the reliability of the control information while using UDP based solutions.

The RLNC results show that the CPU load increases when the generation size increases. The effect is a decrease in data rates as the computation time at the sender has increased. Problems can arise when the sender has encoded without considering the receiver's CPU performance. If the CPU load is too high for the receiver, then the decoding cannot be done efficiently and packets might be dropped due to buffering problems.

The RLNC implementation does not support any form of congestion control. Implementations for handling congestion are required in order for this approach to be applicable in real networks, as transmissions might otherwise overwhelm the networks. If Ericsson's predictions (as discussed in Section 1.1) are true, then congestion-based network control might not be necessary - as congestion might not actually be a major problem when 5G has been implemented. Higher data rates might be desired and UDP based methods such as RLNC might be a solution to avoid the throughput limitations due to burst losses by changes in the network link characteristics.

7 Conclusions and Future work

This chapter presents the conclusions drawn from this project, as well as future work to be conducted in the area. The chapter also describes the limitations that affected the project, and how this research affects the society.

7.1 Conclusions

The first two goals of the thesis project were to investigate transport protocols and the characteristics of the transport protocols. These goals were met by investigating several versions of TCP. Theoretical investigations were made of older versions of the protocol, as well as two newer versions. Experiments using TCP Reno and Cubic TCP over an emulated network with different test scenarios were performed and the behavior was analyzed.

The next goals were to implement and evaluate a proposed network coding solution and to demonstrate its capabilities. This was achieved by implementing a system that uses RLNC from the Kodo library and evaluated by running tests over emulated network links with high packet loss. The addition of a TCP control connection to announce when decoding was completed at the receiver proved to not have any effect.

The last goal was to propose improvements to the concept and the protocols. This goal was met by proposing further implementations where the TCP control connection could be useful. Discussions of where the different methods might still have problems were provided.

Some insights that I have gained is that even if the network can provide higher data rates, this puts a lot of pressure on the end- user devices. Packet handling puts pressure on the CPU when transferring at high link data rates. UDP packet handling did not perform well as the functionality to improve it was not implemented in the device driver. Another insight that I have gained is that TCP was more dependent upon packet losses than expected. When sending more data, more packet losses occur and the transmission is heavily affected. Random packet losses can be crucial to TCP's behavior, ending in low effective throughputs.

From my experience in this thesis project, I would suggest researchers emulate network scenarios instead of performing simulations. Simulations might hide hardware problems that can emerge while actually transmitting data between different hosts via real networks. I would also suggest as further research combining TCP transmissions with UDP based data transmission. I think future solutions can be found in separating logistic information to a TCP flow, while the actual data transfer utilizes a UDP flow.

If I had to do the thesis project again, I would have tried to realize my hardware limitations sooner. I feel that I spent more time setting up the testbed than was needed. I should have tried to understand the problems that were observed, instead of relying on what other sources claimed that the performance should be.

7.2 Limitations

The limitations of my results are based upon the limitations of the testbed. The testbed created for the project was unable to implement radio link protocols that are used in radio connections to reduce the packet losses of transmissions. This limitation made it difficult to make the testbed act as a real radio access network. Another limitation was the values that were used to define the characteristics of the network links. These values were measured in real networks where congestion happens frequently, and most of the packet losses were due to congestion and not from random link errors. As RLNC does not implement congestion control, it is unfair to TCP to use these values to

compare its performance against RLNC's performance. An assumption that was made when measuring TCPs radio performance was that the packet loss value was from errors in the radio environment, and not from congestion on the nodes. This assumption can affect the reliability of the comparison of the two methods. However, the final values used were based on average measures from real network scenarios. By using average values I was able to see how TCP would act in the environment under most of the user cases. Special cases where the RTT and packet loss are very low resembles a perfect connection, and cases where the values are very high could resemble a situation where the network are broken, or under heavy load. Using special case values does not resemble most of the user's experience, and should not be considered in this thesis project.

The maximum UDP throughput was lower than the maximum TCP throughput. This limitation was due to the network cards, which did not implement functionality for interrupt coalescing when using UDP. This problem was explained in Section 4.6.1.

The result of these limitations made the comparison between TCP and RLNC less accurate, and the effect of network congestion needs to be considered when implementing UDP based methods.

7.3 Future work

Future work should implement a way of determining the data rate that the sender should transmit. This data rate should be based on the receiver's CPU performance, the current network congestion, and the current packet loss rate.

A need exists to implement better UDP packet handling in hardware to increase the link's data rates when using UDP based methods. One of the limitations of this thesis was the CPU load when handling UDP packets.

7.4 Reflections

The advantages of developing systems to implement a possible RLNC solution would not be limited to economic aspects. Implementing it on the server side can be done by pre-encoding files with different settings and storing them. When a connection to a user is initialized, the server would retrieve the coded packets for the correct settings based on the receiver and then simply transmit the packets. It would not require much computational power from the server except when encoding new files that are to be stored. However, more storage would be required as the server would need to store more coded packets than the actual data, as well as several different coded packets for different transmission settings. The receiving side might suffer from economic aspects due to the bad packet handling while using UDP (explained in Section 4.6.1). The OS could not keep up with the high throughput packet handling and a lot of CPU power was used to schedule the interrupts. This might be a decrease in performance for certain devices and might have to be implemented to reach higher throughput.

This solution would increase the user's effective data rate over high loss network links and provide users and companies with higher throughput. Files would be transferred quicker, without the network link throughput suffering from non-congestion related packet losses. This would help network providers to reach, or to increase their goals with 5G networks in urban environments.

The environment could be affected in a positive way by being able to increase network throughput through radio. The need for long distance transmission cables would decrease if higher throughput were enabled through long distance radio transmissions. The environment would not be hurt by the installation of these cables. A negative effect on the environment could be that the power consumption increases based on the need for more computational power at nodes. By increasing the

computational requirements from RLNC, the need for less extra coded packets to decode are needed, and the overall traffic would decrease.

References

[1] J. Postel, 'Transmission Control Protocol', *Internet Req. Comments*, vol. RFC 793 (INTERNET STANDARD), Sep. 1981 [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[2] J.-S. Ha, S.-T. Kim, and S. J. Koh, 'Performance comparison of SCTP and TCP over Linux platform', in *Advances in Intelligent Computing*, Springer, 2005, pp. 396–404 [Online]. Available: http://link.springer.com/chapter/10.1007/11538356_41. [Accessed: 30-Jun-2015]

[3] 'Nielsen's Law of Internet Bandwidth'. [Online]. Available: http://www.nngroup.com/articles/law-of-bandwidth/. [Accessed: 22-Jun-2015]

[4] E. Dahlman, G. Mildh, S. Parkvall, J. Peisa, J. Sachs, and Y. Selén, '5G radio access', *Ericsson Rev.*, vol. 6, pp. 2–7, 2014.

[5] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, 'Reducing Web Latency: The Virtue of Gentle Aggression', in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, New York, NY, USA, 2013, pp. 159–170 [Online]. DOI: 10.1145/2486001.2486014

[6] B. Li and Y. Wu, 'Network Coding [Scanning the issue]', *Proc. IEEE*, vol. 99, no. 3, pp. 363–365, Mar. 2011. DOI: 10.1109/JPROC.2010.2096251

[7] T. Fehenberger, A. Alvarado, P. Bayvel, and N. Hanik, 'On achievable rates for long-haul fiber-optic communications', *Opt. Express*, vol. 23, no. 7, pp. 9183–9191, 2015.

[8] J. Nagle, 'Congestion Control in IP/TCP Internetworks', *Internet Req. Comments*, vol. RFC 896, Jan. 1984 [Online]. Available: http://www.rfc-editor.org/rfc/rfc896.txt

[9] K. Mansley, *Tweaking TCP's Timers*. University of Cambridge, Department of Engineering, 2004 [Online]. Available: http://www.cl.cam.ac.uk/research/dtg/lce-pub/public/kjm25/CUED_F-INFENG_TR487.pdf. [Accessed: 30-Jun-2015]

[10] S. Ha, I. Rhee, and L. Xu, 'CUBIC: a new TCP-friendly high-speed TCP variant', *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008. DOI: 10.1145/1400097.1400105

[11] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, 'Network information flow', *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000. DOI: 10.1109/18.850663

[12] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, 'XORs in the Air: Practical Wireless Network Coding', in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2006, pp. 243–254 [Online]. DOI: 10.1145/1159913.1159942

[13] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, 'The benefits of coding over routing in a randomized setting', in *IEEE International Symposium on Information Theory, 2003. Proceedings*, 2003, p. 442–. DOI: 10.1109/ISIT.2003.1228459

[14] S. Patterson, 'How MIT and Caltech's coding breakthrough could accelerate mobile network speeds', *Network World*, 29-May-2014. [Online]. Available: http://www.networkworld.com/article/2342846/data-breach/how-mit-and-caltech-s-coding-breakthrough-could-accelerate-mobile-network-speeds.html. [Accessed: 19-Jul-2015]

[15] M. V. Pedersen, J. Heide, and F. H. P. Fitzek, 'Kodo: An Open and Research Oriented Network Coding Library', in *NETWORKING 2011 Workshops*, vol. 6827, V. Casares-Giner, P. Manzoni, and A. Pont, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 145–152 [Online]. Available: http://link.springer.com/10.1007/978-3-642-23041-7_15. [Accessed: 23-Jun-2015]

[16] B. Barekatain, D. Khezrimotlagh, M. Aizaini Maarof, H. R. Ghaeini, S. Salleh, A. A. Quintana, B. Akbari, and A. T. Cabrera, 'MATIN: A Random Network Coding Based Framework for High Quality Peer-to-Peer Live Video Streaming', *PLoS ONE*, vol. 8, no. 8, p. e69844, Aug. 2013. DOI: 10.1371/journal.pone.0069844

[17] M. Ronquist, 'TCP Reaction to Rapid Changes of the Link Characteristics due to Handover in a Mobile Environment', Royal Institute of Technology, 1999 [Online]. Available: http://people.kth.se/~maguire/DEGREE-PROJECT-REPORTS/990805-Mattias_Ronquist-final-report.pdf. [Accessed: 30-Jun-2015]

[18] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Médard, 'Network coded tcp (ctcp)', *ArXiv Prepr. ArXiv12122291*, 2012 [Online]. Available: http://arxiv.org/abs/1212.2291. [Accessed: 20-Jul-2015]

[19] M. Muhammad, M. Berioli, and T. de Cola, 'A simulation study of network-coding-enhanced PEP for TCP flows in GEO satellite networks', in *Communications (ICC), 2014 IEEE International Conference on*, 2014, pp. 3588–3593 [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6883878. [Accessed: 19-Jul-2015]

[20] K. A. Kumar, S. Vemuri, and R. Suresh, 'Normalized Transfer of Bulk Data By Using UDP In Dedicated Networks' [Online]. Available: http://www.idc-online.com/technical_references/pdfs/data_communications/Normalized%20Transfer.pdf. [Accessed: 20-Jul-2015]

[21] E. Kohler, M. Handley, and S. Floyd, 'Designing DCCP: congestion control without reliability', 2006, p. 27 [Online]. DOI: 10.1145/1159913.1159918

[22] 'How do I troubleshoot slow Myri10GE performance?'. [Online]. Available: https://www.myricom.com/software/myri10ge/392-how-do-i-troubleshoot-slow-myri10ge-or-mx-10g-performance.html#linux. [Accessed: 02-Jul-2015]

[23] 'Performance Insights to Intel® Hyper-Threading Technology | Intel® Developer Zone'. [Online]. Available: https://software.intel.com/en-us/articles/performance-insights-to-intel-hyper-threading-technology. [Accessed: 03-Jul-2015]

[24] M. Carbone and L. Rizzo, 'Dummynet revisited', *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, 2010.

[25] '30.4. IPFW'. [Online]. Available: https://www.freebsd.org/doc/handbook/firewalls-ipfw.html. [Accessed: 25-Jun-2015]

[26] 'scp(1): secure copy - Linux man page'. [Online]. Available: http://linux.die.net/man/1/scp. [Accessed: 25-Jun-2015]

[27] 'bwm-ng(1) - Linux man page'. [Online]. Available: http://linux.die.net/man/1/bwm-ng. [Accessed: 25-Jun-2015]

[28] 'iperf3 — iperf3 3.0.11 documentation'. [Online]. Available: http://software.es.net/iperf/. [Accessed: 25-Jun-2015]

[29] 'Ubuntu Manpage: ksoftirqd - Softirq daemon'. [Online]. Available: http://manpages.ubuntu.com/manpages/dapper/man9/ksoftirqd.9.html. [Accessed: 12-Jun-2015]

[30] 'Why is my 1500-byte MTU UDP performance less than my TCP performance?'. [Online]. Available: https://www.myricom.com/software/myri10ge/441-why-is-my-1500-byte-mtu-udp-performance-less-than-my-tcp-performance.html. [Accessed: 12-Jun-2015]

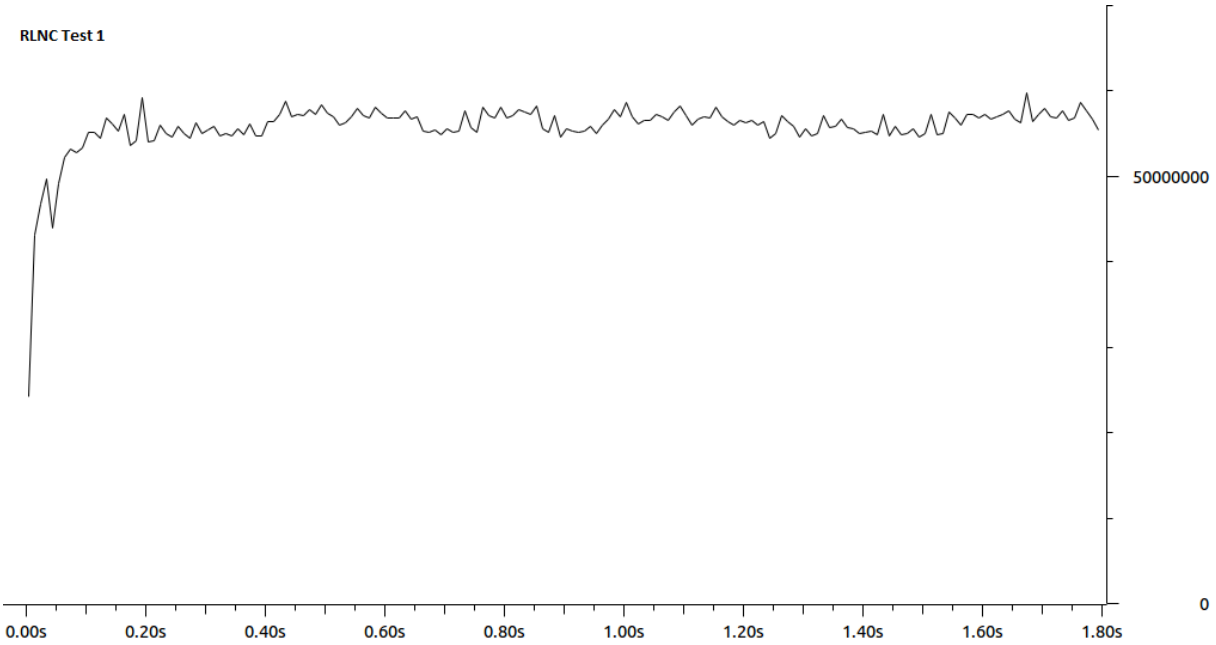
[31] 'dd(1): convert/copy file - Linux man page'. [Online]. Available: http://linux.die.net/man/1/dd. [Accessed: 18-Jul-2015]

[32] 'yuv4mpeg(5) - Linux man page'. [Online]. Available: http://linux.die.net/man/5/yuv4mpeg. [Accessed: 12-Jun-2015]

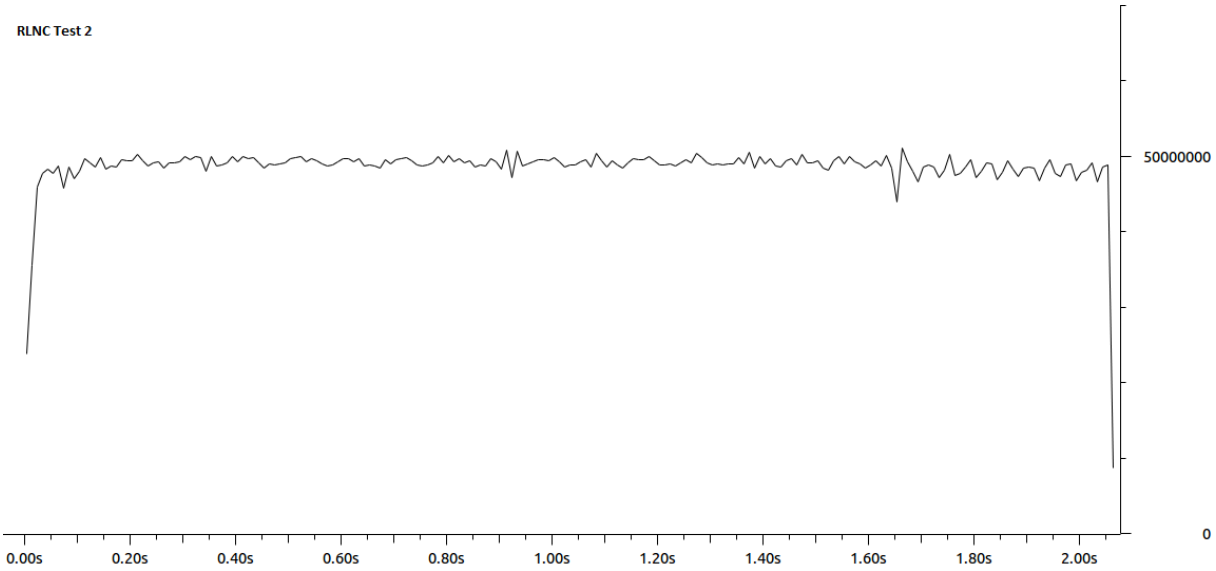
[33] C. Hornig, 'A Standard for the Transmission of IP Datagrams over Ethernet Networks', *Internet Req. Comments*, vol. RFC 894 (INTERNET STANDARD), Apr. 1984 [Online]. Available: http://www.rfc-editor.org/rfc/rfc894.txt

Appendix A:

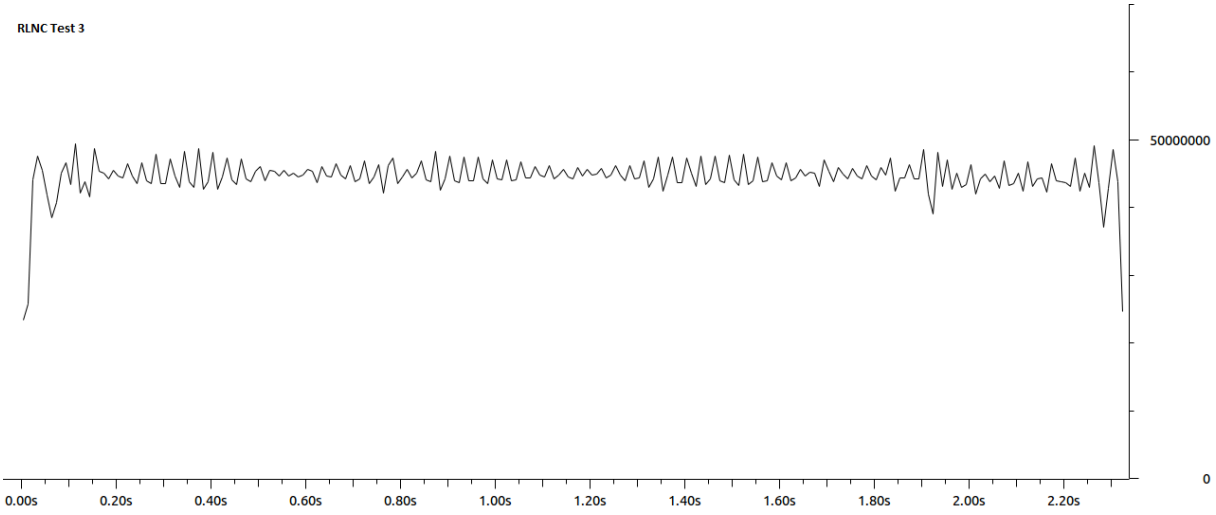
The following graphs show the throughput received at the receiver of the three RLNC tests presented in Section 6.4.3.



A-1: **Throughput graph of the results of test 1 presented in Table 6-4. The Y-axis shows the number of bits/0.1 second.**



A-2: **Throughput graph of the results of test 2 presented in Table 6-4. The Y-axis shows the number of bits/0.1 second.**



A-3: **Throughput graph of the results of test 3 presented in Table 6-4. The Y-axis shows the number of bits/0.1 second.**

TRITA-ICT-EX-2015:178