# Robust header compression for cellular IoT

Måns Ansgariusson and Axel Wihlborg-Rasmusen

Department of Electrical and Information Technology
Lund University

Supervisor: Stefan Höst and Steffen Malkowsky

Examiner: Pietro Andreani

February 19, 2019

# Table of Contents

# List of Figures

# Acronyms

**CID** Context Identifier. 24, 25, 27, 35, 36, 42, *Glossary:* Context Identifier

**CTCP** Compressed Transport Control Protocol. 15, *Glossary:* Service Data Unit

**DYN** RoHC package carrying the dynamic fields in the original eader(s).. 22, 24, 25, 28, 36, 45, 46, 53, 63, *Glossary:* Initialization & Refresh of the Dynamic chain

**FO** First Order Compression. 17, 25, 28, 39, 46, *Glossary:* First Order Compression

**FTP** File Transfer Protocol. 15, *Glossary:* File Transfer Protocol

**IoT** Internet of Things. ix, 1–3, 13, 41, 43, 44, 49, 50, 53–57, 59, *Glossary:* Internet of Things

**IP** Internet protocol. ix, 5, 6, 8, 10, 12, 15, 18, 22, 27, 28, 36, 39, 41, 45–47, 56, 57, 59, *Glossary:* Internet Protocol

**IR** Initialization & Refresh. 17, 23, 25, 27, 28, 36, 37, 39, 46, 53, 63, *Glossary:* Initialization & Refresh

**IR_DYN** Initialization & Refresh + Dynamic chain. 17, 22–25, 28, 36, 37, 39, 45, 46, 54, *Glossary:* Initialization & Refresh + Dynamic chain

**LAN** Local Area Network. 15, 64, *Glossary:* Local Area Network

**LPWAN** Low Powered Wide Area Network. 13, *Glossary:* Low Powered Wide Area Network

**LTE** Long-Term Evolution. 2, 10, 12–14, 35, 55, 64, 65, *Glossary:* Long-Term Evolution

**MAC** Medium Access Control. 12, *Glossary:* Medium Access Control

**NAS** Non Access Stratum. 10, *Glossary:* Non Access Stratum

**O-Mode** Bidirectional Optimistic Mode. 17, 18, 21, 22, 37, 39, 40, 43, 46, 47, 49, 54, *Glossary:* Bidirectional Optimistic Mode

**OSI-Model** Open Systems Interconnection model. 5, 8, 12, *Glossary:* Open Systems Interconnection model

**PDCP** Packet Data Convergence Protocol. 12, 18, 35, 36, 64, *Glossary:* Packet Data Convergence Protocol

**PDU** protocol data unit. 8, *Glossary:* Protocol Data Unit

**R-Mode** Bidirectional Reliable Mode. 17, 18, 21, 22, 25, 40, *Glossary:* Bidirectional Reliable Mode

**RLC** Radio Link Control Protocol. 12, 64, *Glossary:* Radio Link Control Protocol

**RoHC** Robust Header Compression. i, iii, ix, 1–3, 10, 12, 14, 15, 17, 21, 23, 26–28, 35–37, 39–48, 50, 51, 53–57, 59, *Glossary:* Robust Header Compression

**RRC** Radio Resource Control. 10, 12, 64, *Glossary:* Radio Resource Control

**RTP** Real-Time Transport Protocol. 6, 10, *Glossary:* Real-Time Transport Protocol

**RX** is the telegraph and radio abbreviation for receive, receiver, or reception.. 14, 44, 50, 55

**SDU** Service Data Unit. 8, 64, *Glossary:* Service Data Unit

**SO** Second Order Compression. 17, 18, 25, 26, 29, 39, 46, 51, *Glossary:* Second Order Compression

**TCP** Transmission Control Protocol. 5, 8, 10, 15, 17, 39, 43, *Glossary:* Transmission Control Protocol

**TX** is the telegraph and radio abbreviation for transmit, transmitter, or transmission.. 14, 44, 50, 54, 55

**U-Mode** Unidirectional Mode. 17, 21, 22, 25, 37, 39, 43, 46, 49, 53, 54, *Glossary:* Unidirectional Mode

**UDP** User Datagram Protocol. 5, 6, 8, 10, 15, 22, 27, 28, 36, 39, 41, 42, 45–47, 56, 57, 59, *Glossary:* User Datagram Protocol

**VoIP** Voice over IP. 2, 40, *Glossary:* Voice over IP

**WAN** Wide Area Network. 5, *Glossary:* Wide Area Network

# Abstract

In this Master Thesis we aim to evaluate the Robust Header Compression protocol in cellular IoT. The initial results show that RoHC may decrease the energy consumption significantly due to its reduction of the header size. The reduction in the header size decreases the required transmission window which in turn decreases the energy consumption. The result also show that Robust Header Compression is suitable for hardware acceleration but we are unsure if the hardware acceleration may be suitable for an IoT device since it would increase the required chip area.

# Popular Science Summary

**Robust header compression is a well known protocol. It has been standardized for more than a decade, is used in almost all standard cellular communication standards yet not implemented in the smallest of devices. Companies are always trying to give their products an edge and one area to improve in is always battery time.**

Today almost everything and everyone are connected to the internet. There exists a great amount of connected devices and together they transmit a massive amount of data every day. As the number of Internet of Things (IoT) devices is expected to increase exponentially in the coming years, the competition on the market is increasing. IoT devices are expected to last years without supervision or maintenance while still being able to continuously transmit small amounts of data during their lifetime. These devices are a suitable match for applying RoHC.

RoHC exploits the redundancy in header fields. Some header fields like the destination and source address may be stored in the network vertexes. Most of the header fields do not change and are redundant if transmitted repeatedly. The receiver saves these variables and the transmitter only transmit header fields that change every transmission. That way, the receiver can put together a full Internet Protocol (IP) packet and send it to its final destination while the transmitter only sends a small part of the header.

The transmitter can reduce the header from 28 bytes to only 2 bytes. This means RoHC can reduce the data transmitted by up to 92%. This could potentially reduce the transmission windows as well as reduce the risk of residual bit-errors and retransmissions. The Worst-case scenario still point towards a reduction of a couple percent.

Originally the protocol was meant to reduce network congestion. At the time, network congestion was a problem while power existed in abundance. Today the reverse situation has occurred. The networks are robust and the IoT devices are restricted in battery life.

An IoT device uses the most energy while transmitting data. Thus, if less data is transmitted less power is used. On the other hand, the receiver has to do more computational work. Usually a receiver is a base station, a fixed installation with superfluous amounts of power.

This study show that a device may extend its battery time with up to a year by just implementing RoHC. Does this imply that RoHC will provide the edge that IoT hardware companies are looking for or are IoT devices already living longer than needed?

Chapter 1

# Introduction

As of today IoT is in its cradle. IoT applications usually send a very limited amount of data, rarely more than 10-20 bytes payload each transmission. This results in a relatively large overhead due to the size of the protocol headers in relation to the payload of the package. Header compression algorithms can be applied to reduce the overhead caused by the package headers.

## 1.1 Problem motivation

Robust header compression is not a new protocol. It has been used during the last 15 years and is standardized in cellular networks. It removes overhead and helps lower the traffic in the network. It was originally developed to not congest networks but has today been presented with a possible new use, cellular IoT. The demand of IoT devices has skyrocketed and companies today hope to optimize their products in order to keep the edge on the market. Can RoHC be a viable route to improve IoT products?

One motivation for this thesis is to further understand RoHC, its limitations and its strengths. It serves several different purposes. First, it might reduce congestion in cellular network, even though IoT devices rarely contributes to congestion. The cellular networks are robust and IoT devices do not transmit large amounts of data. More importantly there is a hope that RoHC can reduce power usage in IoT devices. This would in turn provide either more computations within the same battery life cycle or increase life time. Since users often are very fastidious, especially in regard to battery time, a longer battery time can increase return on a certain product. Thus, RoHC can serve a multipurpose function if it is found to improve IoT devices significantly.

## 1.2 Aim

This project aims to understand RoHC in order to give an accurate answer to our research question. RoHC shows a great potential to help improve IoT devices via pure study of the protocol. Our goal is to implement Robust header compression, analyse different aspects as described in the Section *Research Questions*1.3 and be able to reach a conclusion regarding the usefulness of RoHC in a cellular IoT device.

## 1.3 Research Questions

RoHC has been standardized since year 2001 and is a part of the Long-Term Evolution (LTE) radio stack. After the exponential increase in bandwidth and transmission rate the need for RoHC decreased.

Today as the development of IoT devices is increasing and the main difference between IoT suppliers is the energy consumption and chip area. To research the impact of RoHC when applied on IoT devices we aim to answer the following questions.

- What is the compression rate of RoHC?
- Would an IoT device benefit from using hardware accelerated RoHC?
- Can RoHC significantly increase the battery life of an IoT device?

## 1.4 Related Work

In the beginning of the 21-century internet bandwidth and transmission rate increased exponentially, which led the congestion rate to decrease substantially. This decrease has rendered RoHC less significant in a network setting [9].

When RoHC was created a possible usage was Voice over IP (VoIP) . Since VoIP sends small data packages where the header size is a large part of the package. In Performance evaluation of RoHC Reliable and Optimistic Mode for Voice over LTE by Andreas Maeder and Arne Felber one can read on how different RoHC modes affects the voice quality. In the study the authors concludes that there is no significant difference between Optimistic and Reliable mode when it comes to the voice quality over the 3G network. However their results show that the feedback overhead for Reliable and Optimistic mode is significantly different while the compression rate is marginally different due to the small amount of feedback messages sent in Optimistic mode [10].

A master thesis conducted by Amen Hussain from the Norwegian University of Science and Technology shows that a low powered IoT device energy consumption is largely based on the transmission time[8].

This lead us to question what is the best possible compression rate of RoHC and how it might affect the battery length of an IoT device and if the time spent implementing RoHC into an RoHC device can be motivated by the increase in battery life.

The thesis, *Hardware acceleration of the robust header compression (RoHC) algorithm* ( [11]), presents measurements of the increase in performance when implementing the RoHC algorithm in hardware and software. They found a mix between hardware and software implementation was the best match, however they didn't measure the amount chip area this would use. Since the size and battery length of an IoT device are the most competitive aspects between the hardware a small increase in the chip area may decrease the IoT hardwares ability to compete on the market.

## 1.5 Project Disposition

This thesis is meant to give a good understanding of how RoHC works and how it can be applied. Below is a short description of the different chapters of the report, what they contribute and contain.

Chapter 2 briefly describes important information in order to be able to comprehend different parts of the thesis. Background only reiterates known facts and does not add any technological advances.

The next chapter, Chapter 3, describes RoHC. It contains the most important information regarding RoHC; its key features, error handling and robustness. It is designed to give a good overview of our scope and our take of the protocol.

Chapter 4 introduces our implementations structure, design choices and frameworks. It also describes how we conducted our experiments and describes certain limitations.

Chapter 5 contains our findings in metrics, complete with graphs, Tables and descriptive texts.

The penultimate chapter 6 presents a discussion regarding the results, what conclusions we have drawn and how this might impact future IoT products. Firstly, the results are discussed, each to their own, and then connected in a conclusion. Secondly, certain sources of errors are brought up and discussed.

Lastly, chapter 7 provides an introduction to the next possible steps.

Chapter 2

# Background

## 2.1 Network Communication

The Internet is a communications network that spans across the globe and enables a quick means of communication between endpoints. Information that travels over the Internet does so via a variety of languages known as protocols. All endpoints share the same set of protocols to communicate their payload, or data, across the Internet. This is more commonly known as the TCP/IP suite since Transmission Control Protocol (TCP) and IP are the foundational protocols in the Internet communication suite [16].

The IP-stack describes how the different protocols cooperate to deliver the payload between different endpoints. Communication protocols encapsulate one another, where each layer have its own responsibility as can be seen in Figure 2.1. The layers are represented in the Open Systems Interconnection Model (OSI-Model) as can be seen in Figure 2.2.

- **Application Layer** - is the top layer. It is mainly concerned with human interaction and the implementation of software applications and related protocols.
- **Transport Layer** - Provide application-to-application communication services.
- **Internet Layer** - Provides information on how to relay network packages across network boundaries, also known as routing.
- **Link Layer** - Provides a communications protocol between physically connected devices in a local network.

The IP and Internet Protocol (UDP) are two of the fundamental Internet protocols. The IP handles how data packages are transported between Wide Area Network (WAN) boundaries. A network boundary is often described as a communication vertex between local networks, commonly known as routing. Local networks interconnect to wide area networks and Wide Area Networks interconnects to become "the Internet". The Internet refers to an arbitrary number of connected WAN, where multiple actors operate. This means that the IP is responsible for the transport between networks. There are two main versions of IP: IPv4 and IPv6. IPv4 is the legacy protocol that is by far the most used today while IPv6 is the protocol that is bound to replace it in the future due to lack of address space in IPv4 [16]. UDP is responsible for data integrity and port delivery. It does not guarantee the correct package order or data correctness. UDP utilises checksum just as IPv4 to determine if the data has been corrupted. Note: checksum can say whether a

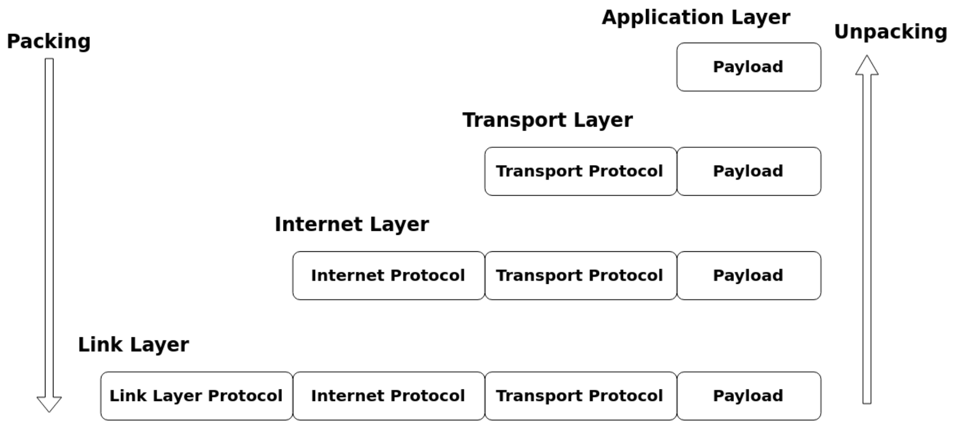


**Figure 2.1:** The TCP/IP Stack

package is corrupted but cannot guarantee correctness. The difference between corruption and correctness lies in the nature of the error checking method. Checksum is an error detection method that finds errors based on a 16-bit complement sum calculated over the data and IP headers (excluding the checksum). The checksum is expected to detect errors at a rate proportional to $1 - \frac{1}{2^{16}}$. It is worth noticing that the checksum also has one major limitation: the sum of a set of 16-bit values is the same, regardless of the order [16].

## 2.1.1 The Internet Protocol

The IP is responsible for addressing hosts, encapsulating data into datagrams and routing them from a source host to its destination across one or more network boundaries. Each package consists of a header and a payload, the header consists of a minimum of 20 bytes of data that specifies how the payload should be routed between network boundaries. The Internet package has two versions of headers that is in use today, IPv4 and IPv6. As seen in Figure 2.3 and Figure 2.4 the difference between the two protocols are many and the most prominent difference is the size of the source and destination addresses. The main reason for the creation of the IPv6 header was to deal with the problem of IPv4 address exhaustion. IPv6 does this by increasing the Source and Destination address fields from 4 to 16 bytes. It is an increase of possible address-space by a factor $2^{96}$. The IPv6 protocol are not backward compatible because of the fundamental differences in address space in the IPv4 and IPv6 header structures [16]. The IPv4 header can be seen in Figure 2.3 and the IPv6 header in Figure 2.4.

## 2.1.2 The User Datagram Protocol

The UDP is another of the core members of the Internet protocol suite. UDP is a connectionless transport protocol. It is a minimalistic protocol that transports payload without guarantee of delivery, ordering or duplication protection [16]. Seen in Figure 2.5 is the structure of an UDP header. UDP is commonly used with other protocols such as Real-Time Transport Protocol (RTP) .

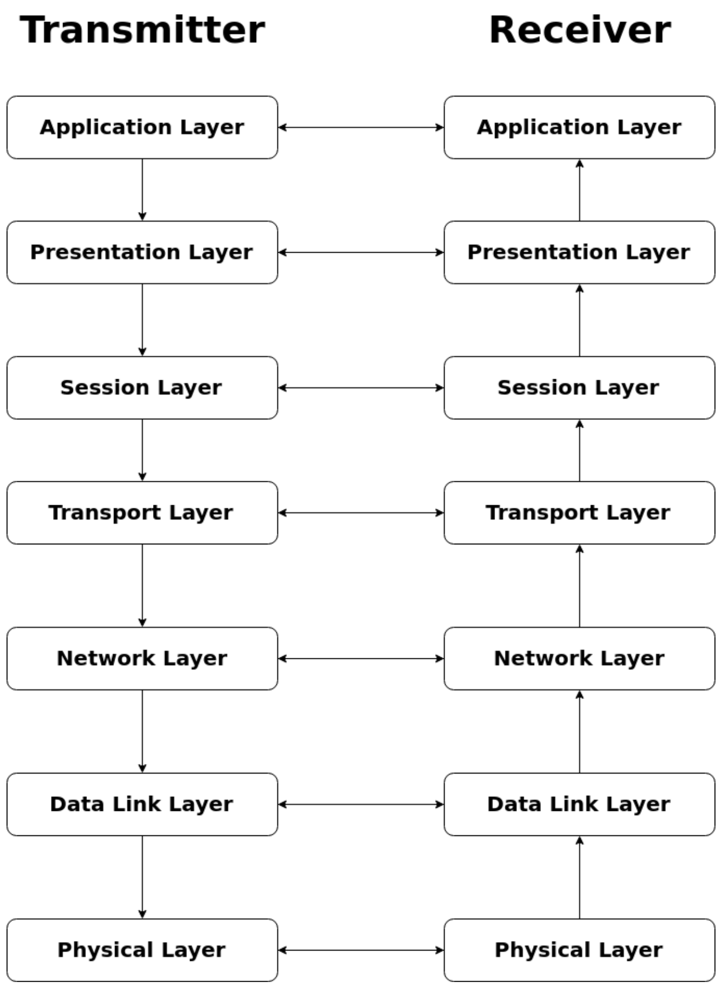


**Figure 2.2:** OSI-model

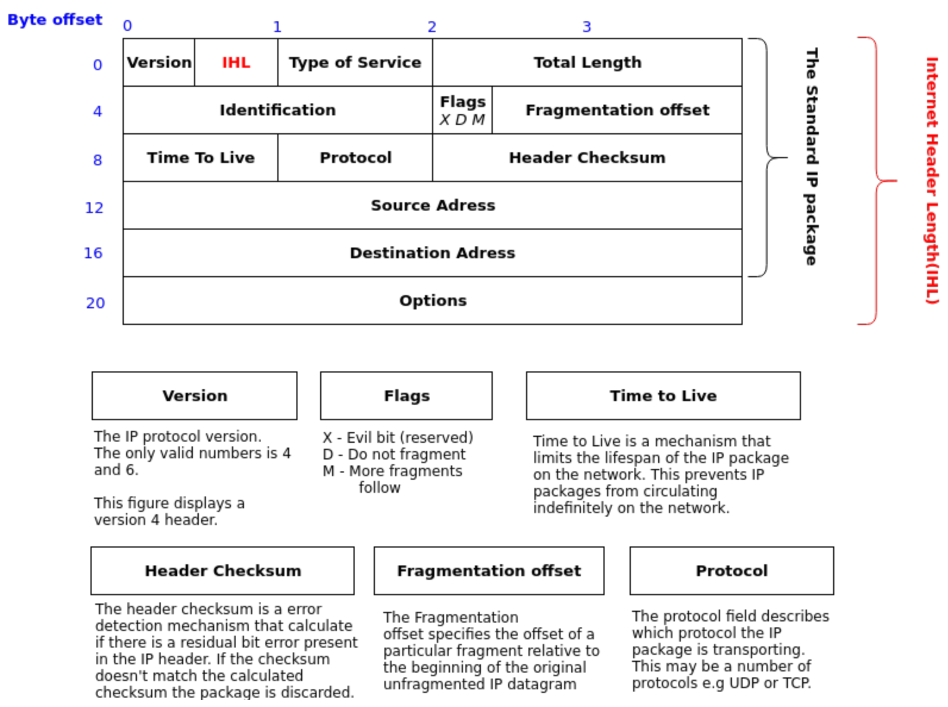

Byte offset
0
1
2
3
0
4
8
12
16
20
Version
IHL
Type of Service
Total Length
Identification
Flags
X D M
Fragmentation offset
Time To Live
Protocol
Header Checksum
Source Adress
Destination Adress
Options
The Standard IP package
Internet Header Length(IHL)
Version
The IP protocol version. The only valid numbers is 4 and 6.
This figure displays a version 4 header.
Flags
X - Evil bit (reserved)
D - Do not fragment
M - More fragments follow
Time to Live
Time to Live is a mechanism that limits the lifespan of the IP package on the network. This prevents IP packages from circulating indefinitely on the network.
Header Checksum
The header checksum is a error detection mechanism that calculate if there is a residual bit error present in the IP header. If the checksum doesn't match the calculated checksum the package is discarded.
Fragmentation offset
The Fragmentation offset specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram
Protocol
The protocol field describes which protocol the IP package is transporting. This may be a number of protocols e.g UDP or TCP.

**Figure 2.3:** The Header of a IPv4 package

### 2.1.3 Data encapsulation

Data encapsulation is key in network communication. In the OSI-Model, the higher layers attach their specific header and sends it further down. A package transmitted to a lower level consists of that layers header and the payload that is encapsulated by the header.

A Protocol Data Unit (PDU) is a collective name of a single unit of information that is transmitted. Different kinds of PDUs are, but not limited to, UDP Datagram, TCP Segment or IP Packet [13]. A Service Data Unit (SDU) is the information carried by a PDU. An example is an IP-package as can be seen in Figure 2.6. The entire package is a PDU and the data inside the PDU, the payload, is referred to as a SDU. From a layers point of view, any data that comes to your level to be encapsulated will be denoted SDU and what is sent further down will be your PDU. This process continues all the way until reaching the physical layer, where the PDU is transmitted instead of encapsulated.

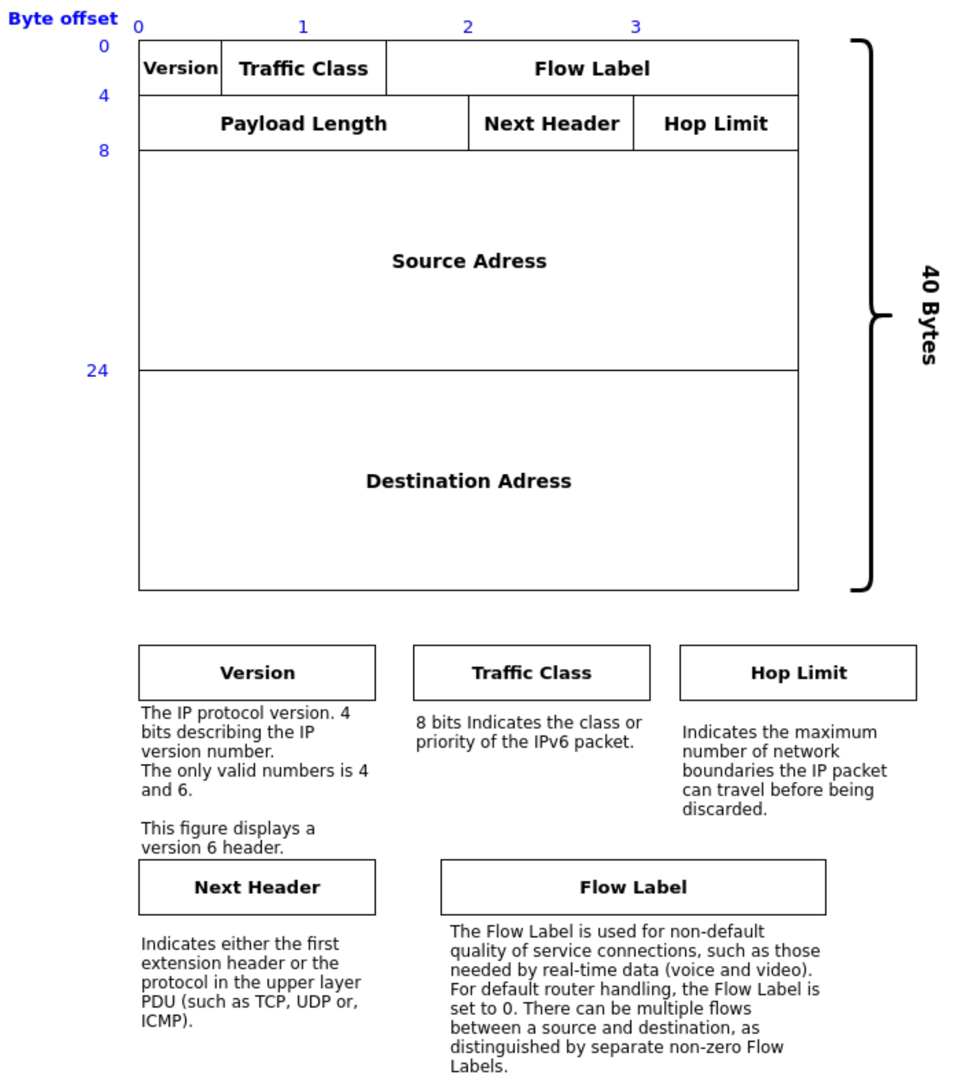

Byte offset
0
1
2
3
0
4
8
24
Version
Traffic Class
Flow Label
Payload Length
Next Header
Hop Limit
Source Adress
Destination Adress
40 Bytes
Version
The IP protocol version. 4 bits describing the IP version number. The only valid numbers is 4 and 6.
This figure displays a version 6 header.
Traffic Class
8 bits Indicates the class or priority of the IPv6 packet.
Hop Limit
Indicates the maximum number of network boundaries the IP packet can travel before being discarded.
Next Header
Indicates either the first extension header or the protocol in the upper layer PDU (such as TCP, UDP or, ICMP).
Flow Label
The Flow Label is used for non-default quality of service connections, such as those needed by real-time data (voice and video). For default router handling, the Flow Label is set to 0. There can be multiple flows between a source and destination, as distinguished by separate non-zero Flow Labels.

**Figure 2.4:** The Header of a IPv6 package

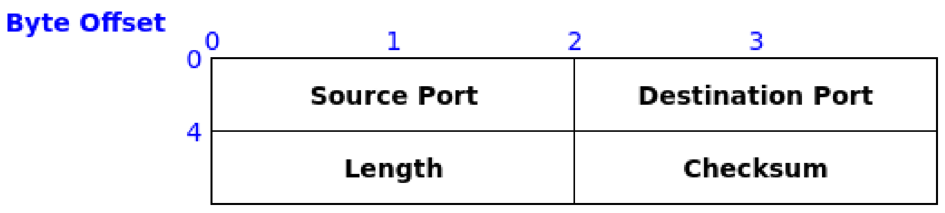


**Figure 2.5:** The Header of a UDP package

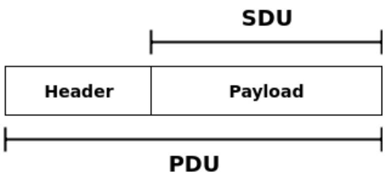


**Figure 2.6:** Overview of PDU and SDU

## 2.2 Radio stack

The Radio Protocol Stack as seen in Figure 2.7, also known as E-Utran Stack, LTE Protocol Stack or LTE Stack, is a stack to communicate over the cellular network. The LTE stack provides Link Layer capabilities similar to Ethernet. Ethernet is a standardized method for sending data over cables while LTE provides methods for communication in the cellular network. Both protocols have the capability to send IP-packets (with payloads such as UDP/TCP/RTP). Even though, they both enable communication but over different links. The LTE stack can provide different services than Ethernet, such as RoHC. This is because how cellular connections are established and how other methods, rather than IP, are used to identify a cellular user; context identification numbers. This implies, the LTE stack have other possibilities to alter and facilitate different overlaying protocols and still maintaining functionality within the cellular network. If the packet are to be sent to an end-user outside the cellular network, standard protocols are to be used.

### 2.2.1 Layer 3

Layer 3 consists of two different kind of packages, IP or Non Access Stratum (NAS) and Radio Resource Control (RRC). These are two different routes with a simple difference. If the packet is user made, i.e. an application, IP will be used. If the packet is made from user equipment e.g. call control management or identity management, NAS / RRC will be used.

NAS together with RRC is the backbone in mobile communication. The main usage is to establishing and maintaining connections with base stations, listen after area updates and paging updates. Simplified, paging is a notification such that the RRC must open a downlink to the base station because of an incoming package. If you are starting up your phone and want to send a data package, the RRC will set up connection such that there is an uplink ready to use. RRC handles these requests, modifies the user equipments behaviors after current status and maintaining connections with the base stations.

The IP/UDP protocols in the LTE Stack is identical to its normal use together with another link layer protocol. As previously mentioned, there is a difference in how IP packets work within cellular networks. This means that IP and subsequent protocols in layers above does not need to be changed in any way to be used with the LTE stack.

Layer 3 have responsibility to establish, maintain and close connections while keeping user equipment on a reasonable activity level. The other half consists of IP, meant to transfer and receive information needed by certain applications.

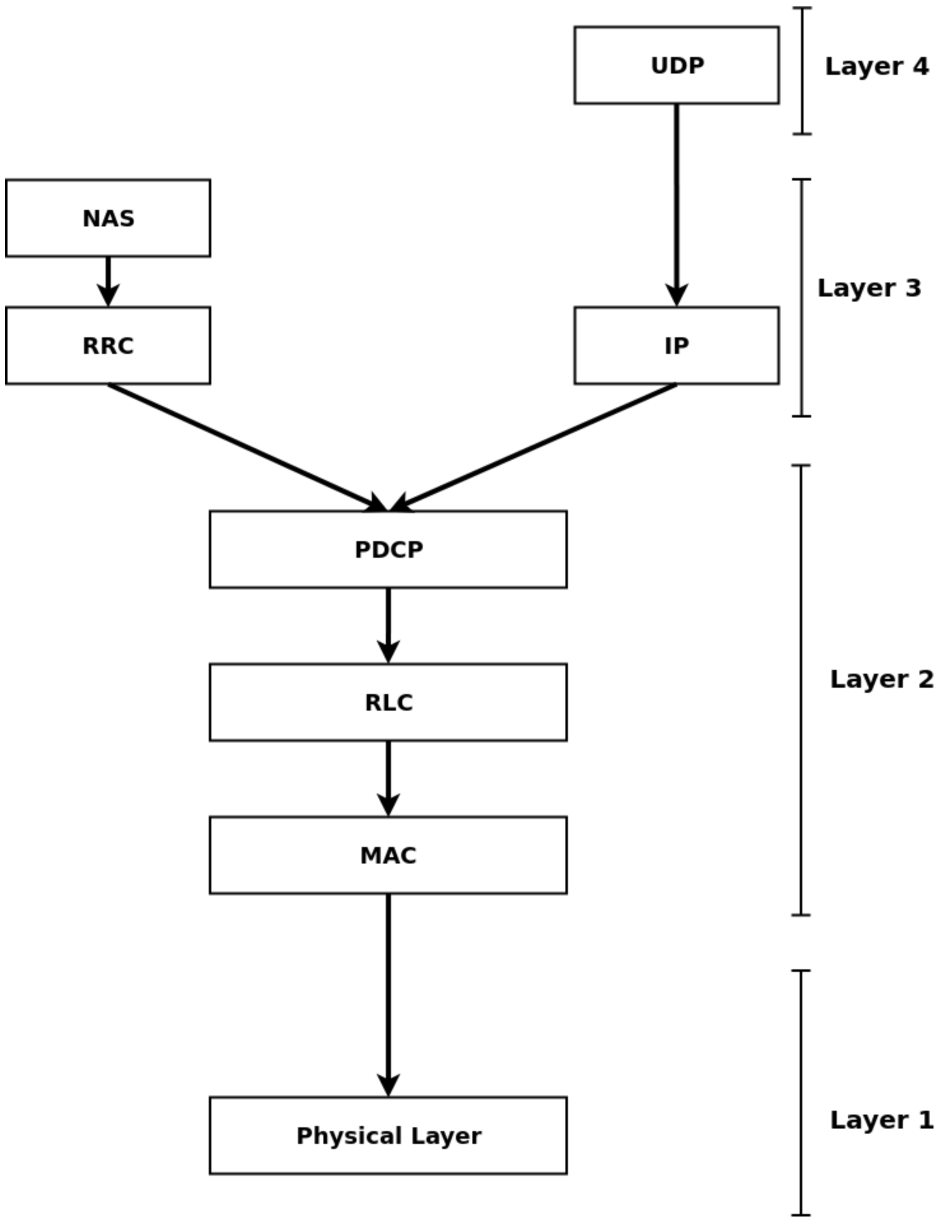


**Figure 2.7:** The Radio Protocol Stack

### 2.2.2 Layer 2

#### Radio Link Control Protocol - RLC

Radio Link Control Protocol (RLC) is a Control protocol with three different states, Transparent Mode (TM), Acknowledge Mode (AM) and Unacknowledged Mode (UM). Depending on RLC state, this will enable or disable certain functions in the Packet Data Convergence Protocol (PDCP) layer, such as segmentation and RoHC.

RLC's main responsibilities can be viewed in the list below.

- Transfer of upper layer Protocol Data Units in one of the three modes.
- Re-segmentation of RLC data PDUs (AM)
- Reordering of RLC data PDUs (UM and AM);
- Duplicate detection (UM and AM);
- RLC SDU discard (UM and AM)
- RLC re-establishment
- Protocol error detection and recovery
- Error correction (only for AM data transfer)
- Concatenation, segmentation and reassembly of RLC SDUs (UM and AM)

RLC is at the same level as the logical link control (LLC) in the OSI-Model and thus have very similar uses.

#### Packet Data Convergence Protocol

PDCP is a support protocol in the LTE stack, tasked with header compression, integrity protection and ciphering. It is the only protocol to link with the IP stack and provides RRC with key functions. It is a versatile protocol, able to provide many support functions. While glancing over it, much of PDCP's functionality seems simplistic. But, as seen in this thesis with RoHC, much of the functionality PDCP uses can be extremely complex. It keeps track of more or less the entire communication, including specific packages, package data and certain timestamps. PDCP provides ciphering and deciphering of data, packet discards based on a timer, duplication or malformation, fragmentation and data transfers to other layers.

As aforementioned PDCP does contain the RoHC scheme and is responsible to provide certain data to the scheme. It passes on which header compression profiles are able to be used, max context identifier values and, segmentation information. The latter are not applicable in the NB-IoT standard provided by 3GPP. Since RoHC resides within PDCP, PDCP is responsible to provide RoHC with data to be compressed/decompressed and handle the output.

#### Medium Access Control

The Medium Access Control (MAC) layer is the first protocol in the LTE-stack and receives bytes from Layer 1. It is responsible for mapping between logical and transport channels and provides some error correction.

### 2.2.3 Layer 1

Layer 1 is tasked with the actual sending of data, it converts bits from previous layers into analog waves for sending and the other way around when receiving. Since layer 1 transfers bits via a physical medium, this is also where bit errors are introduced. There is no single channel today that is error free and there will most likely never exist a channel that can guarantee correctness. This is important, since it is not error free, bit errors are introduced. The robustness of the protocols are designed to detect errors, but in the cases where a protocol is not robust enough, the entire infrastructure of the stack is threatened. Errors must be detected and corrected (either by resending packet or correcting corrupt bits), else unexpected behaviours are introduced. Although some errors can be detected, and corrected, in Layer 1, most of the high level error detection is made further up in the stack. The normal procedure is to detect/correct errors in L1, detect errors in L2 and have further detection and retransmission in L4.

## 2.3 Internet of Things

IoT is an umbrella term for small embedded devices that are connected to the internet. IoT devices are increasingly applied in industrial applications to increase efficiency and reduce waste. IoT devices are fairly simple, with low complexity in its data collection.

An IoT device is generally supposed to be able to sustain itself for a period of 5-10 years without maintenance. The lifetime limitations of an IoT device is mostly dependent on the battery and the operational time is solely based on the energy efficiency of the device. The most battery consuming part of the IoT device is the radio module. It has a relatively large energy consumption compared to the IoT devices other processes [8]. To minimize the energy consumption of the IoT device the time the radio module is turned on should be as small as possible. One way to optimize this would be to minimize the transmission payload the radio module needs to transmit and thereby minimize the energy consumption of the radio module.

### 2.3.1 Narrowband IoT - NB-IoT

NB-IoT is a Low Powered Wide Area Network (LPWAN) technology standard developed by 3GPP. These NB-IoT devices operate in the medium frequency band, 300 kHz to 3MHz. The NB-IoT standard is now considered "completely developed", meaning no additional functionality may be added, mainly since 3GPP froze the project in June 2016 [15]. This means that only minor corrections and essential backwards compatibility changes will be allowed. The project was frozen with the specifications of Release 13 (LTE Advanced Pro), which in turn allows NB-IoT to implement and use technologies up until 5G, but does not include 5G. Although NB-IoT may use a couple of technologies used in Pre-5G, the data is still sent on medium frequency bands. This ensures two major factors, communication can be long-range and uses less power compared to a high frequency band. NB-IoT is thus better suited for the task of self sustainability for a long period of time as well as frequent long-range data transfers.

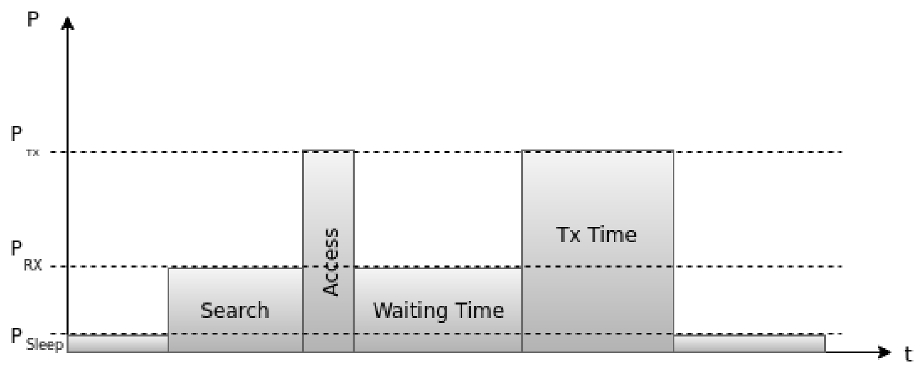


**Figure 2.8:** The NB-IoT energy cycle

### 2.3.2 Communication cycle

The LTE stack uses different methods for communication than regular internet does. Since devices are not connected via a physical static link, power has to be devoted to find antennas, set up a connection before data can be sent or received. Depending on what kind of device, set-up may take a couple of milliseconds up to minutes. This is under the assumption that an antenna is within reach, else set-up time will have an infinite limit.

A device can, simplified, be in three different states; sleep, Receive (RX) and Transmit (TX). In sleep, no communication is used. This does not imply the device is powered down or is not in use, it only states that there are no need for communication at that time. A device can remain in this state for years and will try to be in this state whenever transmission or receiving of data is not needed. In RX, the device is set to listen. Parts of the device are woken up and more power is devoted to receive packages. In TX the device transmits data to the antenna. This part uses the most amount of power. The device will always try to be in the lowest possible state. This does mean that it will only be in TX when it is sending. If it is not finished with the transmission but can't transmit instantaneously, it will revert to the RX state and save energy [8].

Figure 2.8 shows a typical communication cycle. The device wakes up from sleep mode, set up the connection, asks if it is able to transmit. When it receives the go ahead it will send its data. After it has sent the data, it may continue to stay in RX mode ready to receive data or return to deep sleep as seen in the Figure 2.8. After a timeout the device will go back to sleep mode. This does mean that there is only one part that can be affected by RoHC, the second TX part.

## 2.4 Header Compression

Header compression of protocol headers is made possible due to redundancy of information in the header fields, both within the same package and in consecutive packages belonging to the same data stream. Header compression assumes the redundant data fields are recovered at the decompressor or endpoint of the link and stored in a context. The compressor and the decompressor update their respective contexts upon certain, not necessarily synchronized, events. Disruptive events may lead to inconsistencies in the decompressor known as context damage, which in turn may cause incorrect decompression. A successful Robust Header Compression scheme needs mechanisms to minimize the possibility of context damage in combination with mechanisms for context repair if a failure were to arise [3].

### 2.4.1 History of Header compression

Before RoHC was created and implemented, more primitive solutions were in place to help with what was then modern technologies and their limitations. Compressed TCP has its background in the late 20th century. The internet was on the rise, personal computers became increasingly more powerful and moved into people's homes. This quick expansion of the internet proved to be a challenge, a complex problem of the time [6]. Problems that are hard to fathom today when our phones have more computational capacity than all computers used at the moon landing 1969 [4] and we have connection speeds that are rapidly increasing [7]. Compressed Transport Control Protocol (CTCP) is looking to help keep headers of TCP down and is a predecessor to RoHC, born from an earlier protocol, Thinwire-II [2]. Thinwire-II was a protocol that began addressing overhead issues, or rather the problem with sending a lot of unnecessary data when updating a tiny segment. Depending on the field of work, this can absolutely be necessary, but not in a computer interaction setting. Thinwire-II was a protocol developed 1984, when connections often sent characters at a time. That did not cut it when CTCP came around 1990, when bulk data was sent more often. Personal computers left the iterative 'terminal' and old protocols like Telnet and Thinwire-II became redundant when internet speeds increased and new modern protocols like File Transfer Protocol (FTP) began sending bulk data. This did however come with different, more complex problems. During this period, congestion was a real threat and with increasing traffic the problems got worse. For reference, during this period of time, network connections had a transmission rate of 300 - 19.200 bps compared to some areas today where we have 1 Gbps. Thus, every byte that did not have to be sent was a victory.

CTCP is a protocol that can only be used for TCP/IP, a protocol suite that the internet does rely heavily on. It is also a protocol used to provide a reliable transmission of data over an unreliable communication line. One of CTCP's problems are; "All error detection is based on redundancy yet compression has squeezed out almost all the redundancy in the TCP and IP headers. In other words, the decompresser will happily turn random line noise into a perfectly valid TCP encapsulated IP packet." [6].

When communication ramped up, more robust solutions were needed. Robust Header compression was thus introduced 2001 [3]. Much of its work is based on CTCP, with certain key differences; it provides a more robust error detection scheme and it can compress UDP, RTP and many other protocols. Its task is to further remedy the same problems that CTCP provided a solution for, with even greater efficiency. To help keep congestion in Local Area Network (LAN)'s down by removing overhead in packages.

# Chapter 3

# Theory

## 3.1 Overview of RoHC

RoHC is a protocol that builds on having two state machines that operates within the same local network. The two state machines are located at the network boundary respectively at the client. The client state machine initiates a communication context by sending an Initialization & Refresh (IR) or Initialization & Refresh + Dynamic chain (IR_DYN). These messages are carrying the static fields in the protocol headers, fields that will remain the same throughout the context. This is called the static_chain. The dynamic fields are supplied in case of a IR_DYN message and denoted the dynamic_chain. These fields may change at spuriously throughout the communications context. When the state machine at the network boundary has received both the static and dynamic parts of the package stream it can recreate the original header and pass along the payload over the network boundary.

The State machines has three operational modes:
Unidirectional Mode (U-Mode), Bidirectional Optimistic Mode (O-Mode), and Bidirectional Reliable Mode (R-Mode). Each mode has its own objective, the U-Mode are utilised where there is no feedback channel or where feedback are unwanted. O-Mode and U-Mode are very similar in terms of operation, the key difference is that the O-Mode receives feedback irregular to keep the state machines up to date with each other. R-Mode on the other hand operates much like TCP and expects continuous feedback and from the state machine at the network boundary which grantees delivery and successful decompression at the network boundary. In Section 3.2 one can read a more detailed description of the modes and their strength and weaknesses.

All modes share the same set of states: IR, First Order Compression (FO), and Second Order Compression (SO). Each state symbolizes a level of compression. The IR state indicates that the connection is either not initialized or that the compressor is not confident that the decompressor has the correct static information. FO indicates that the compressor believes that the decompressor has received the static context correctly. The compressor then sends packages consistent of the dynamic fields of the original protocol header(s) and payload to the decompressor. Before the dynamic and static data has been correctly parsed by the decompressor it cannot reconstruct the original protocol header and thereby not relay the data across the network boundary. If the compressor believes that both the static and dynamic fields has been received and stored correctly by the decompressor it ascends to SO. In SO the compressor sends a minimal header and data to the decompressor. The

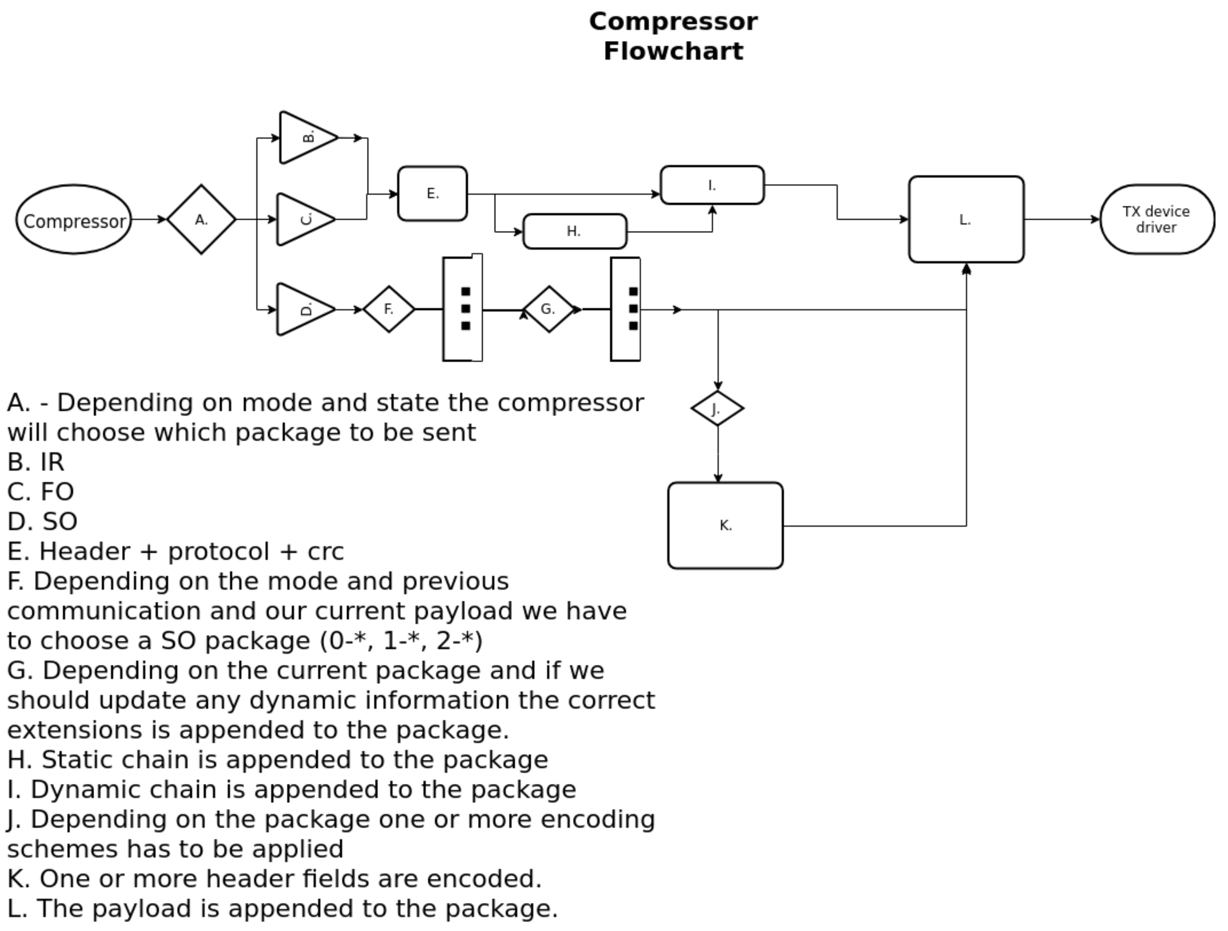
SO may update dynamic fields or just relay data to be passed across the network boundary.

The SO state are what separates U and O-Mode from R-Mode. In U and O-Mode the SO packages look very similar but in R-Mode more control data needs to be transmitted to the decompressor in order to assure delivery of the payload. A more detailed explanation can be found in Section 3.3.

As can be seen in Figure 3.1 the compressor starts by receiving an IP package from PDCP to relay to the decompressor at the network boundary. Based on the current mode and state it passes a set of different phases to compress the package before sending it to lower layers which are responsible for sending the data to the decompressor at the network boundary.

Until now we have mostly discussed the compressor, the decompressor is much similar to the compressor since it mostly follows the same set of operations to unpack and verify the correctness of the package. If you compare the Figures 3.1 and 3.2 you will notice the way they differ; the decompressor is clueless of what the compressor has decided to send, so in every given instance it has to be prepared to parse any given possible package. The relationship between the compressor and decompressor is much like a dance where the compressor leads and the decompressor follows, however the decompressor will not follow if the move is incorrect. This may result in the compressor dancing on its own and the decompressor is stationary until the compressor realizes its mistake and invites its counterpart to join again.

Both the compressor and decompressor states are revisited in Section 3.3.2 and Section 3.3.1 respectively.



A. - Depending on mode and state the compressor will choose which package to be sent
B. IR
C. FO
D. SO
E. Header + protocol + crc
F. Depending on the mode and previous communication and our current payload we have to choose a SO package (0-*, 1-*, 2-*)
G. Depending on the current package and if we should update any dynamic information the correct extensions is appended to the package.
H. Static chain is appended to the package
I. Dynamic chain is appended to the package
J. Depending on the package one or more encoding schemes has to be applied
K. One or more header fields are encoded.
L. The payload is appended to the package.

**Figure 3.1:** Compressor flowchart

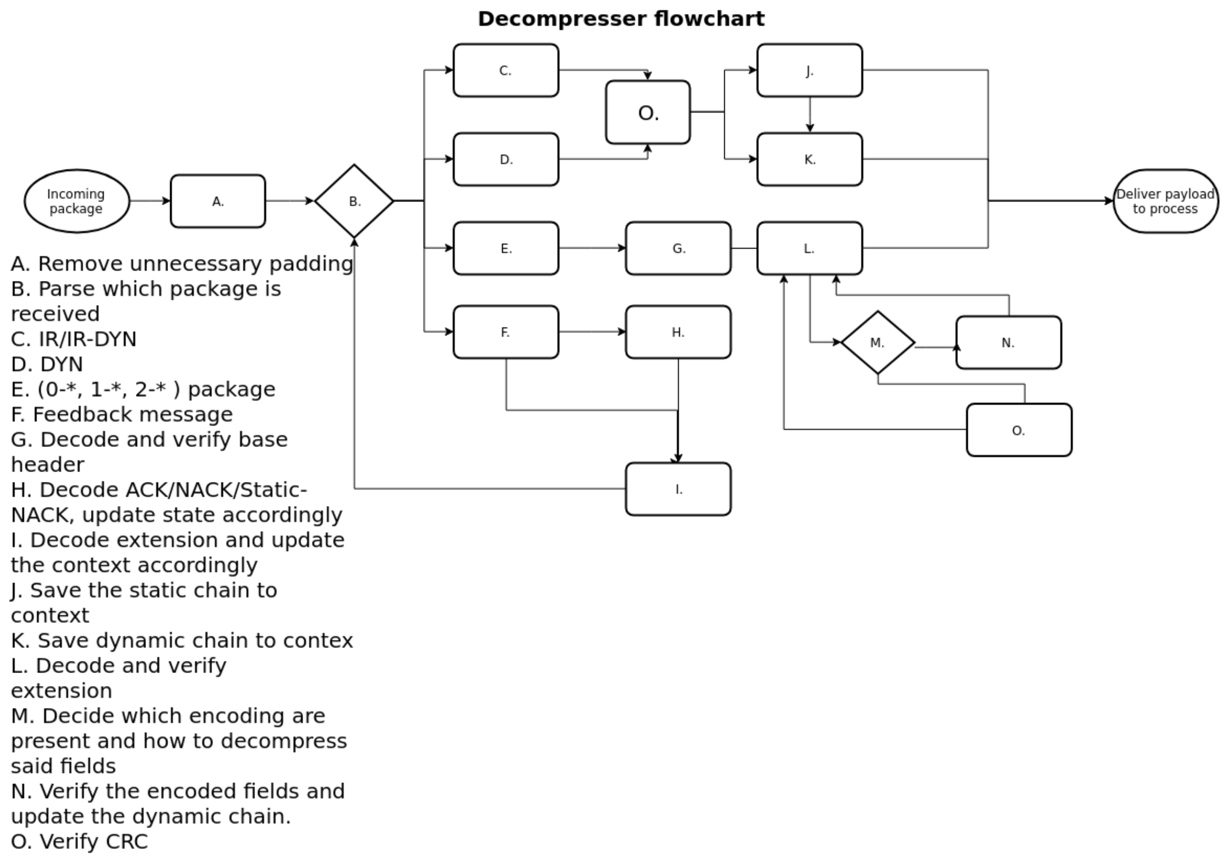

Decompresser flowchart
Incoming package
A.
B.
C.
D.
E.
F.
O.
G.
H.
I.
J.
K.
L.
M.
N.
O.
Deliver payload to process
A. Remove unnecessary padding
B. Parse which package is received
C. IR/IR-DYN
D. DYN
E. (0-*, 1-*, 2-* ) package
F. Feedback message
G. Decode and verify base header
H. Decode ACK/NACK/Static-NACK, update state accordingly
I. Decode extension and update the context accordingly
J. Save the static chain to context
K. Save dynamic chain to contex
L. Decode and verify extension
M. Decide which encoding are present and how to decompress said fields
N. Verify the encoded fields and update the dynamic chain.
O. Verify CRC

**Figure 3.2:** Decompressor flowchart

## 3.2 Modes of Operation

As can be viewed in Figure 3.3 RoHC implements three modes of operation.

- U-Mode
- O-Mode
- R-Mode

The modes of operation are designed with different link conditions in mind. When a

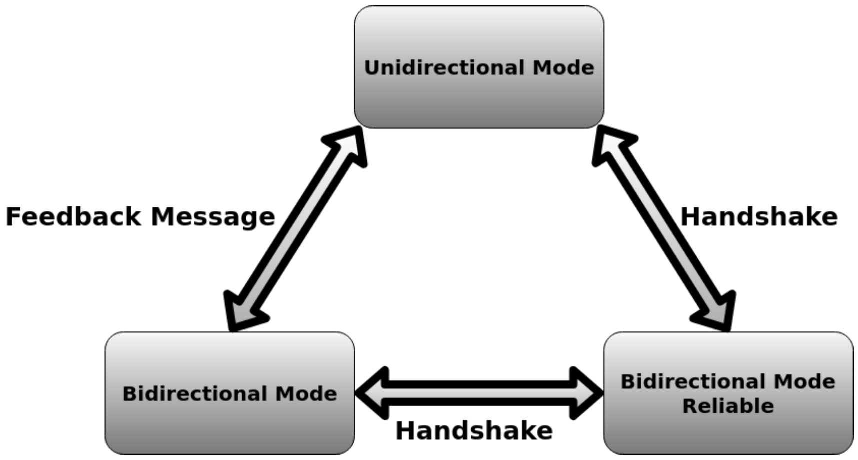


**Figure 3.3:** RoHC Mode of operation

connection is established between the compressor and decompressor both compressor and decompressor are set in *Unidirectional Mode* and *Initialization and Refresh* state. From there the decompressor may send feedback requesting a change in the operational mode and in the progress give feedback of the decompressors current state by sending an ACK, acknowledgment, NACK, negative acknowledgment, or static-NACK. The compressor and decompressor may operate in different states as long as the decompressor is in the equivalent or higher state than the compressor. This adds to the reliability of compression protocol. This allows the compressor and decompressor operate without any feedback as is done in *Unidirectional Mode*.

### 3.2.1 Unidirectional Mode

Unidirectional mode (U-Mode) is the most basic mode, it supports but does not require feedback from the decompressor. U-Mode is utilised where the link does not support feedback messages.

U-Mode utilises timeouts or package patterns to move between states since it does not require to get frequent feedback messages from the decompressor. The timeouts and movements are not specified by the standard and are up to the developers of the RoHC framework. This does not provide optimal decompression between the compressor and decompressor, the decompressor can stay in the fully compressed state through out the connection period but since it has no means of communicating this to the compressor the compressor will continue in its movement pattern and therefore refreshing contexts where it was not necessary.

To improve performance for the U-Mode over a link that have a feedback channel the decompressor may send an acknowledgement when a decompression succeeds. This allows the compressor to reduce or eliminate the initialization & refresh messages. The feedback in U-Mode may be sent by the decompressor at any rate but continuously.

To use the dance analogy, the compressor and decompressor will dance accordingly but if the decompressor fails to compress an IR, IR_DYN, or DYN message the compressor is dancing on its own and the data is not relayed over the network boundary. In the case of IP/UDP the data is lost before it has left the local network.

### 3.2.2 Bidirectional Optimistic Mode

O-Mode is initiated by a feedback message from the decompressor. O-Mode compared to U-Mode uses a feedback channel. The feedback channel are used for error recovery and, optionally, for acknowledgments of context updates from the decompressor to compressor. O-Mode does not utilises periodical refreshes and will always operate at the highest possible compression rate. O-Mode aims to maximize the compression rate and use as few feedback messages as possible. It reduces the number of damaged headers delivered to the upper layers due to residual errors or context damage.

Lets revert to the dance analogy again. In O-Mode the compressor and decompressor dance more in sync. The compressor dances nervously but is reassured by the infrequent nods or disapproving looks from the decompressor if it misses a step. The dance is therefore less likely to end in the compressor dancing on its own.

### 3.2.3 Bidirectional Reliable Mode

Just as O-Mode Bidirectional Reliable Mode, R-Mode, is initialized by a feedback message by the decompressor. However it differs significantly from both U and O-Mode since its intensive use of feedback messages and a stricter logic at both the compressor and decompressor that prevents context damaged unless hit with very high residual bit error rates. Feedback messages are sent to acknowledge all context updates, including the sequence number. It is said that not all messages in R-Mode updates a dynamic part in the RFCs, however this is not true for 3GPP since it does not support segmentation.

In the dance of R-Mode the decompressor and compressor are dancing without ever looking away from each other, each step is met with a friendly nod and each misstep is met with a smirk in the counterparts face. All steps are executed perfectly and all the data is guaranteed to be relayed across the networks boundary. It is important to note that it does not guarantee the delivery to the end node only that the data is correctly relayed across the network boundary.

## 3.3 States

The compressor and decompressor have three states that mirrors each other. In Image 3.4 one can observe the three different states and how the compression efficiency is increased in the higher states and the header size is increased in the lower states. Figure 3.4 also shows the relationship between the states, where each state is built on the lower ones. If a lower state fails, the higher states cannot convey data across the network boundary. The two state machines are working along side each other and only allows packages to be

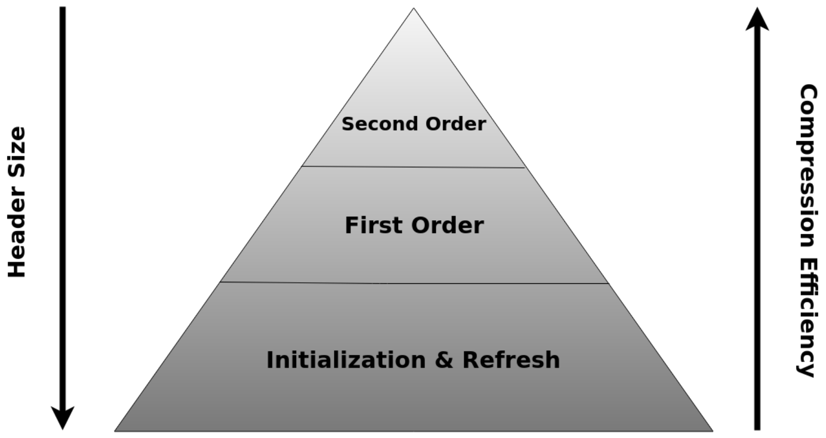


**Figure 3.4:** RoHC State Hierarchy

decompressed if the decompressors state machine is in an equal or higher state than the corresponding compressors state. The compressor and decompressor do not need to have a two way communications channel between them to operate, see 3.2.1.

The compressor and decompressor may update their state machines when they send or receive:

- An Initialization & refresh package - IR
- An Initialization & refresh package + Dynamic - IR_DYN
- A Dynamic Initialization & refresh package - DYN
- A Feedback message
- An Acknowledgment or Negative Acknowledgment

### 3.3.1 decompressor States

The decompressor states can be viewed in the Figure 3.5. The images describes each state and how each correctly received package affects the decompressors state, e.g. a IR_DYN message if parsed correctly moves the decompressors state to *Full Context*. As stated previously, the decompressor always operates in the highest possible state.

#### No context - NC

Before the decompressor is initialized it is said to be in the No context state. When the decompressor receives an IR or IR_DYN message it is then moved to either Static context state or full context state. The decompressor may be in the NC-state during two cases, if the Context Identifier (CID) is not in use or if the decompressor has repeatedly failed to decompress the incoming packages from the compressor. It then sends a Static-Nack if a feedback channel is available and waits for a IR/IR_DYN message to refresh the context, all subsequent packages that are not refreshing the context are discarded.

#### Static context- SC

The decompressor moves to SC state when the decompressor is confident the static variables are correct and the dynamic variables either hasn't been initialized or the decompressor has failed to decompress or send packages across the network boundary repeatedly. When the decompressor is in the static context state it requires a full DYN or IR_DYN message from the compressor to move up to a full context.

#### Full Context - FC

Full Context is when the decompressor is confident that the static and the dynamic fields are correct and it has successfully decompressed packages.

This is the highest decompressor state, the decompressor and compressor do not necessarily need to be in the corresponding state as long as the compressor is in a lower or equal state as the decompressor, the decompressor will be able to decompress the packages. If a mismatch between the compressor and decompressor or the decompressor has repeatedly failed to decompress messages the context is considered damaged. Context damage is going to be further described in 3.7.

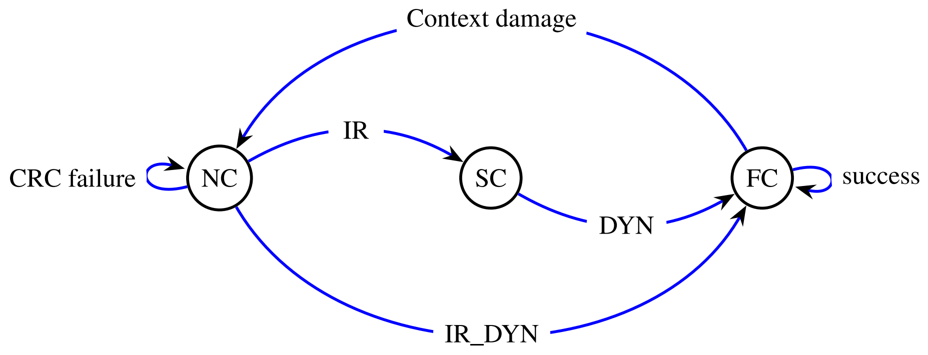


**Figure 3.5:** decompressors State transition mechanism

### 3.3.2 Compressor States

The compressor states are a bit different than the decompressors states. The compressor does not have to work at the highest compression rate and it is not defined how the compressor changes its state. The compressor may change its state according to negative acknowledgment or acknowledgment from the decompressor if both compressor and decompressor has negotiated to operate in O/R-mode. If it operates in U-Mode it may work according to a timeout routine or a predefined package movement pattern. It is up to the developers of the compressor to decide what best suites the compressors use-case. In the Figure 3.6 one can see how and why the compressor may change its state.

In U-Mode it is the compressor that dictates the pace and the compression rate while the decompressor decides if the data should relay across the network boundary. Whilst in O- or R-Mode the decompressor dictates both the compression rate and the correctness of the context.

### Initialization & Refresh

Initialization and refresh, IR, is a state which initiates a communication channel between the compressor and decompressor. The package sent to Initialize the channel may be a IR_DYN or a pure IR package. A pure IR package only contains the contexts static_chain, see 3.4.1, wheres the IR_DYN contains both the contexts static and dynamic_chain, see 3.4.2. The static_chain is the static variables of a package stream. The static variables are protocol specific and are specified according to each individual protocol. The dynamic_chain contains variables that may at random change during the package stream, these variables are also protocol specific.

The IR message starts or resets a package stream. It delivers redundant information to the decompressor which will later use this to produce a package that can be routed between network boundaries.

### First Order

The First order compression, FO, is a package that consists of only the type indication, dynamic_chain, and the payload. It will assume the decompressor have the static chain in its context. This will allow the decompressor to create correct packages that may be routed between network boundaries. We have chosen to refer this type of package as the DYN package, this to separate the IR and IR with the dynamic part (IR_DYN) packages to make it easier to distinguish the difference between states and its movement patterns.

The compressor moves to the FO state whenever the headers of the package stream changes its dynamic parts. It stays in FO until it is confident that the decompressor has received and updated the new dynamic variables.

### Second Order

The second order compression, SO, indicates that the compressor assumes that the decompressor has correct static and dynamic information. It will then send the minimal required information to the decompressor and thereby achieve the highest possible compression rate for the packages it is carrying.

When the compressor is in SO it has a set of packages it may send to the decompressor depending on the current state of the package stream as a whole. The packages sent are based on the current mode and the options in the static and dynamic_chain. The sent packages contains at least the CID and a Sequence Number, SN. A variety of packages may be sent in the SO state and the variables that may be sent are depending on the protocol in use.

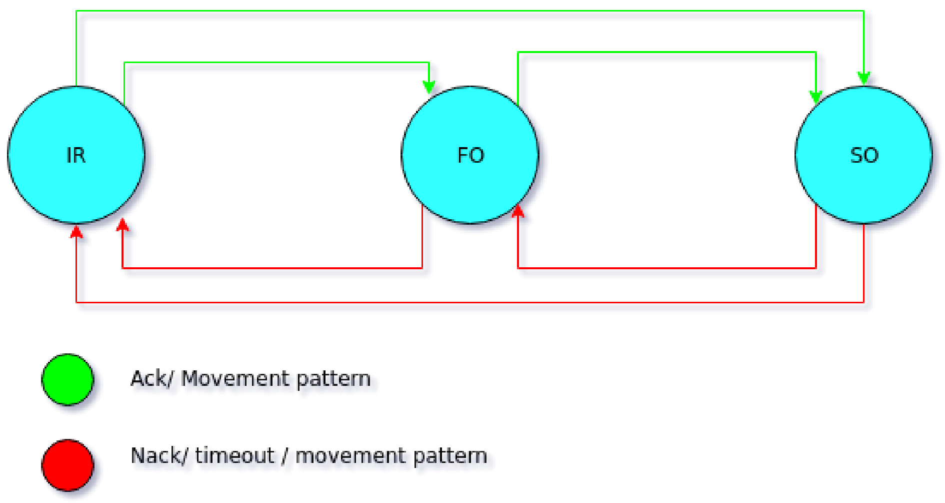


**Figure 3.6:** The possible movement patterns by the compressor

## 3.4 Package structures

A RoHC package may contain a variable amount of segments. A RoHC package has to contain at least a header segment or a feedback segment. Figure 3.7 tries to show the RoHC segment structure. As seen in Figure 3.7 a package may start with zero or more padding octets, it is followed by zero or more feedback segments. The feedback segment(s) and the package header are semi optional, each RoHC package needs to contain at least a feedback segment or a package header.

RoHC uses numerous packages to update its context, convey data to the counterpart, or initialize a context. The packages available can be placed in one of the following groups:

- Initialization & Refresh, see 3.4.1
- First Order, see 3.4.2
- Second Order, see 3.4.3
- Feedback, see 3.8

The vast majority of package headers are found in the Second Order compression. The headers found in Initialization & Refresh and First Order are packages that updates or initializes the whole static and, or dynamic chain. While the packages in the second order is more specialized to maintain the maximum compression rate. The Second Order package headers may update parts of the dynamic context by using specialized packages.

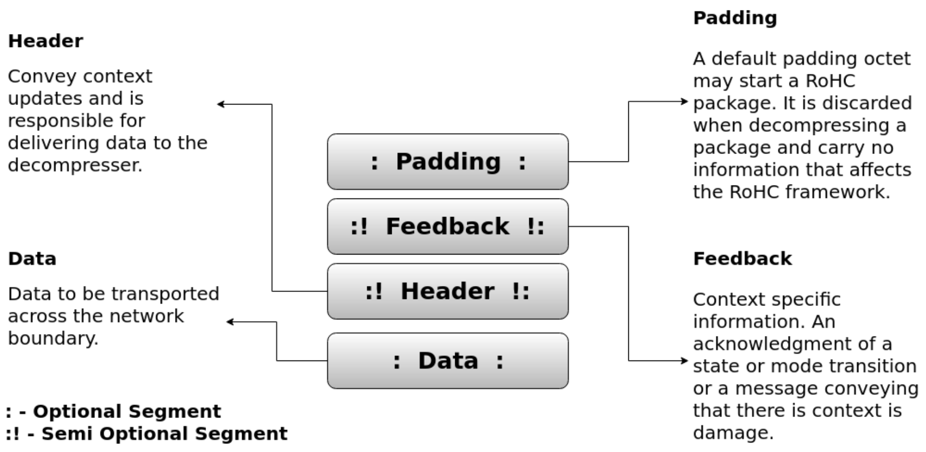

Header
Convey context updates and is responsible for delivering data to the decompresser.
Data
Data to be transported across the network boundary.
: - Optional Segment
:! - Semi Optional Segment
: Padding :
:! Feedback !:
:! Header !:
: Data :
Padding
A default padding octet may start a RoHC package. It is discarded when decompressing a package and carry no information that affects the RoHC framework.
Feedback
Context specific information. An acknowledgment of a state or mode transition or a message conveying that there is context is damage.

**Figure 3.7:** A visualization of the RoHC package segments

Feedback headers are a bit different from the other, they allow for extensions. The feedback extensions aims to convey the state of the decompressor to the compressor. Detailed information regarding feedback can be found in Section 3.8.

The most basic packages consists of CID, type indication, and body. A CID where $0 < CID < 16$ is called a small-CID. The header field that carries the CID information in a context that uses small-CID is called ADD-CID. The ADD-CID field can be viewed in the Figure 3.8, the first byte represents the ADD-CID octet. If $CID > 15$ then the CID is placed independently in one or two bytes after the type indication byte and the CID in the ADD-CID octet is set to 0. The ADD-CID octet with the CID set to zero is called the default padding.

### 3.4.1 Initialization & Refresh

Initialization and refresh, IR, is a package displayed in the Figure 3.9, which updates or initializes a context between the compressor and decompressor. The IR messages is responsible for updating or initializing the static fields in the current context e.g. the source and destination address in anIP package.

The IR-package can also update the dynamic parts of an context, this is indicated when the type indication field is set to IR and ends with a high (1) bit. This indicates that the IR messages contains the static fields followed by the dynamic fields. If there is more data left of the message the rest is considered payload. The static and dynamic fields are profile specific for the protocol(s) the RoHC is compressing.

#### Static chain

The static_chain is profile specific information that contains the static fields of a package stream. The static fields do not change through out the package stream. Figure 3.10 shows the static_chain for the UDP/IP profile (0x02).

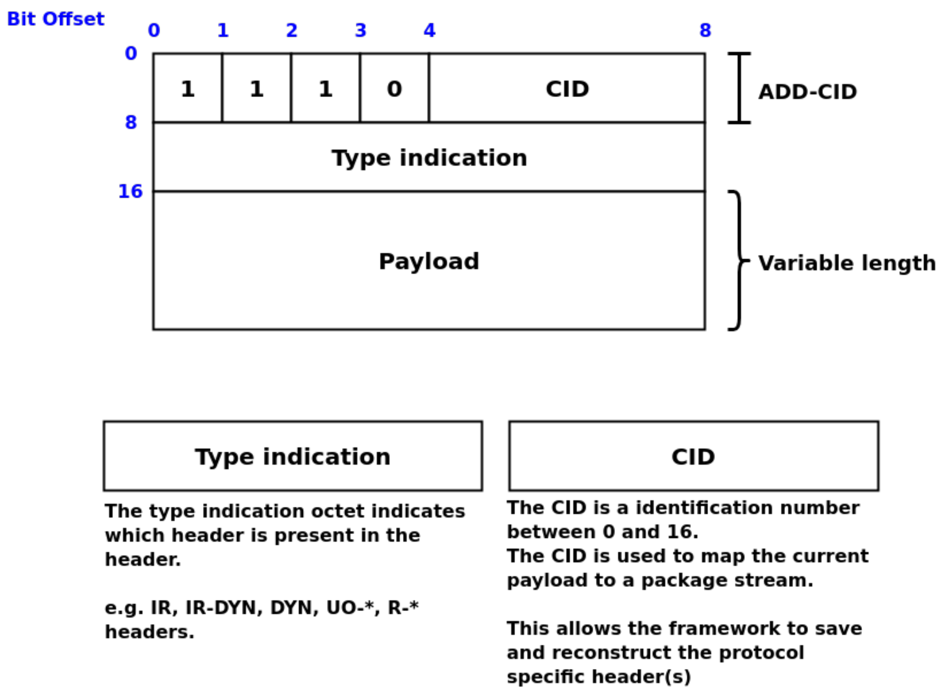

Bit Offset
0
1
2
3
4
8
0
8
16
1
1
1
0
CID
ADD-CID
Type indication
Payload
Variable length
Type indication
The type indication octet indicates which header is present in the header.
e.g. IR, IR-DYN, DYN, UO-*, R-* headers.
CID
The CID is a identification number between 0 and 16.
The CID is used to map the current payload to a package stream.
This allows the framework to save and reconstruct the protocol specific header(s)

**Figure 3.8:** A pseudo package with a small CID

## 3.4.2 First-Order compression

FO is the state where the dynamic parts of the context is initialized or updated. Figure 3.11 displays the DYN messages composition. The compressor assumes that the static parts already have been saved in the decompressors context which lets the compressor remove the static part from the message header. The name *First-Order compression* comes from the fact that it is the stage where information may be left out from the package header. If you compare the Figures 3.9 and 3.11 you will notice that the static_chain is no longer part of the message header. A quick disclaimer: the IR message does not include the dynamic chain but the IR message on its own is not able to convey data over a network boundary, a IR_DYN consists of both static and dynamic context and has the ability to transport data across the network boundary. If the decompressor successfully decompresses the DYN package the decompressor is said to be in FC (Full context).

### Dynamic chain

The dynamic_chain contains the dynamic fields of the profile specific headers. The dynamic fields are frequently changing in the original header and may change spuriously throughout the communication context. In the Figure 3.12 one can see the profile specific data for IP/UDP (0x02) that is stored in the dynamic_chain. The dynamic_chain may be updated at any time which has led the RoHC framework to allow second order packages to update dynamic variables to maximize the compression rate.

### 3.4.3 Second-Order compression

Second-order compression, SO, is the highest compression state and will assume that the decompressor context has all the necessary information to send the package payload to its indented destination. Package headers may be specific for the mode or can be used cross modes.

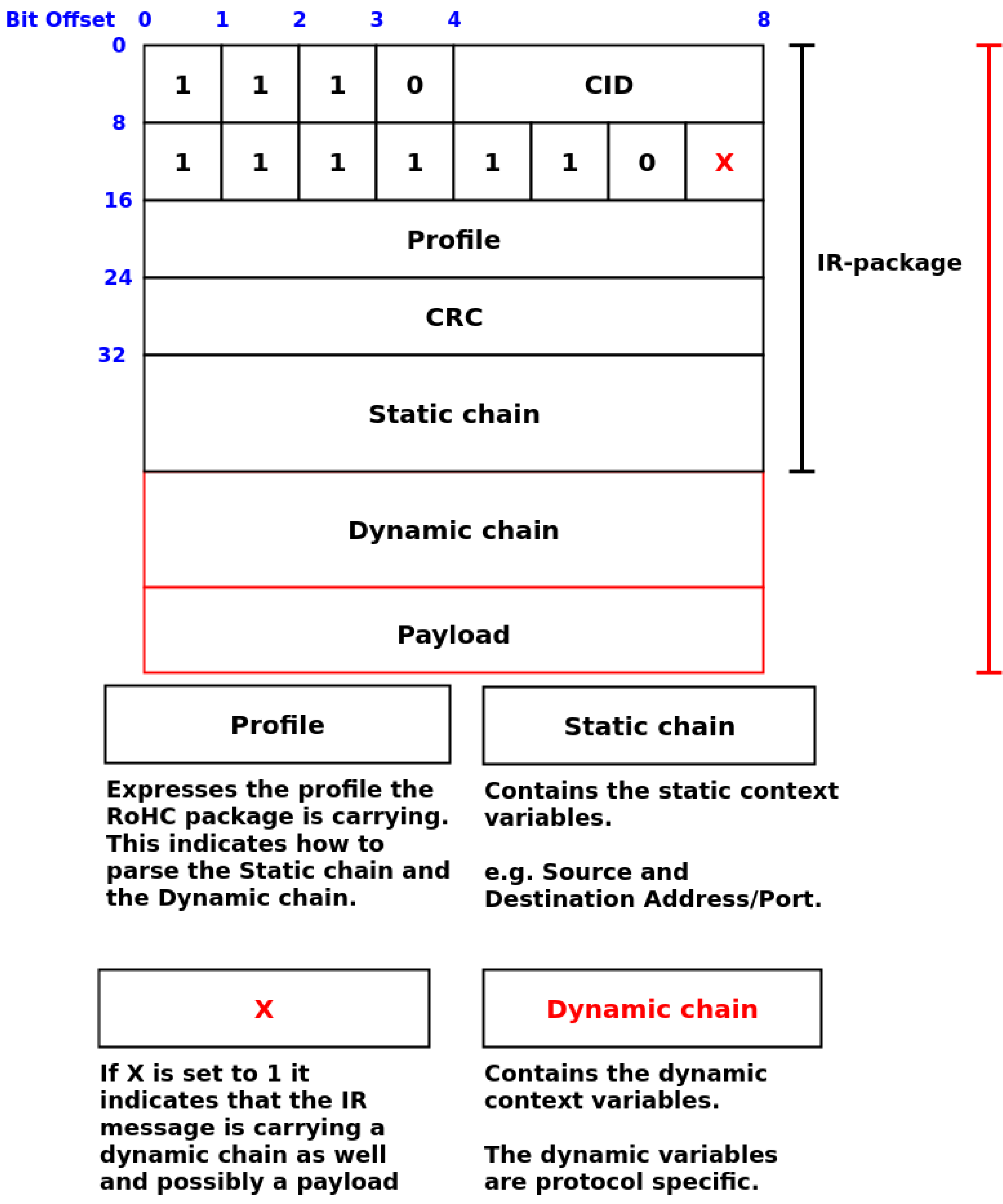
Some of the package headers may indicate a high(1) bit which signals the presence of an extension to the current header which updates some of the contexts dynamic information.

#### Type 0-package

The type 0-package can be viewed in Figure 3.13. The UO-0 package is used to achieve maximum compression. The package streams have converged to a pattern which makes, assuming the decompressor has the correct static and dynamic chains, all dynamic and static fields redundant.

#### Type 1&2-packages

The type 1&2-packages are used to update some of the contexts dynamic parts if the package stream diverges from its previous state. In the Figure 3.14 the UO-1 package structure is shown. These packages supply the decompressor with the dynamic fields which have diverged from the dynamic chain. If the dynamic fields have diverged too much from the current package stream the compressor will have to send a full DYN_MSG to update the decompressors dynamic context.



Expresses the profile the RoHC package is carrying. This indicates how to parse the Static chain and the Dynamic chain.

Contains the static context variables.

e.g. Source and Destination Address/Port.

If X is set to 1 it indicates that the IR message is carrying a dynamic chain as well and possibly a payload

Contains the dynamic context variables.

The dynamic variables are protocol specific.

**Figure 3.9:** An IR message

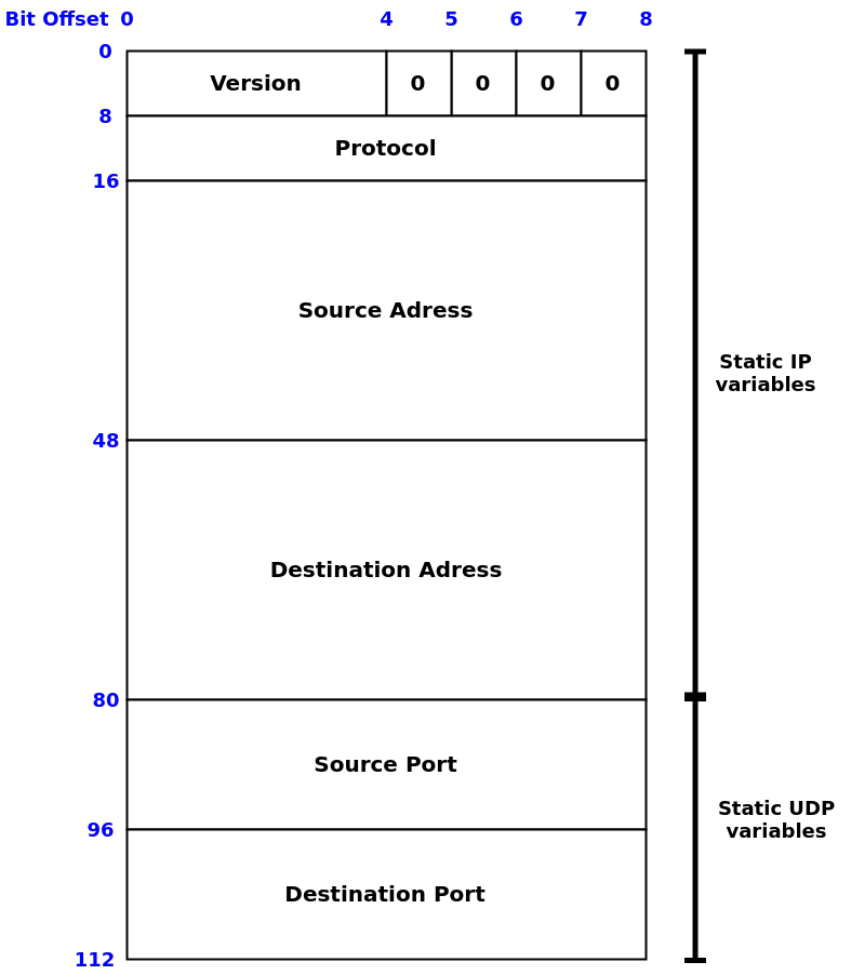


**Figure 3.10:** The static_chain for UDP/IP (profile 0x02)

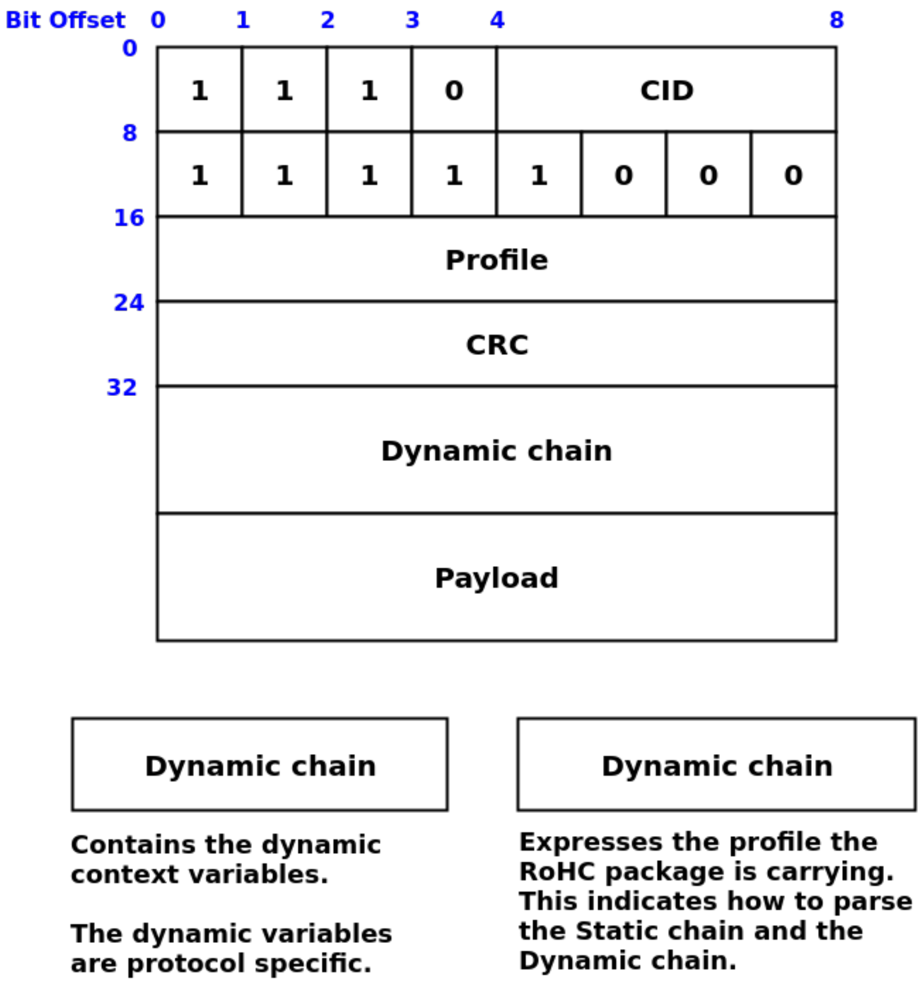

Bit Offset
0
1
2
3
4
8
0
8
16
24
32
1
1
1
0
CID
1
1
1
1
1
0
0
0
Profile
CRC
Dynamic chain
Payload
Dynamic chain
Contains the dynamic context variables.
The dynamic variables are protocol specific.
Dynamic chain
Expresses the profile the RoHC package is carrying. This indicates how to parse the Static chain and the Dynamic chain.

**Figure 3.11:** An DYN message

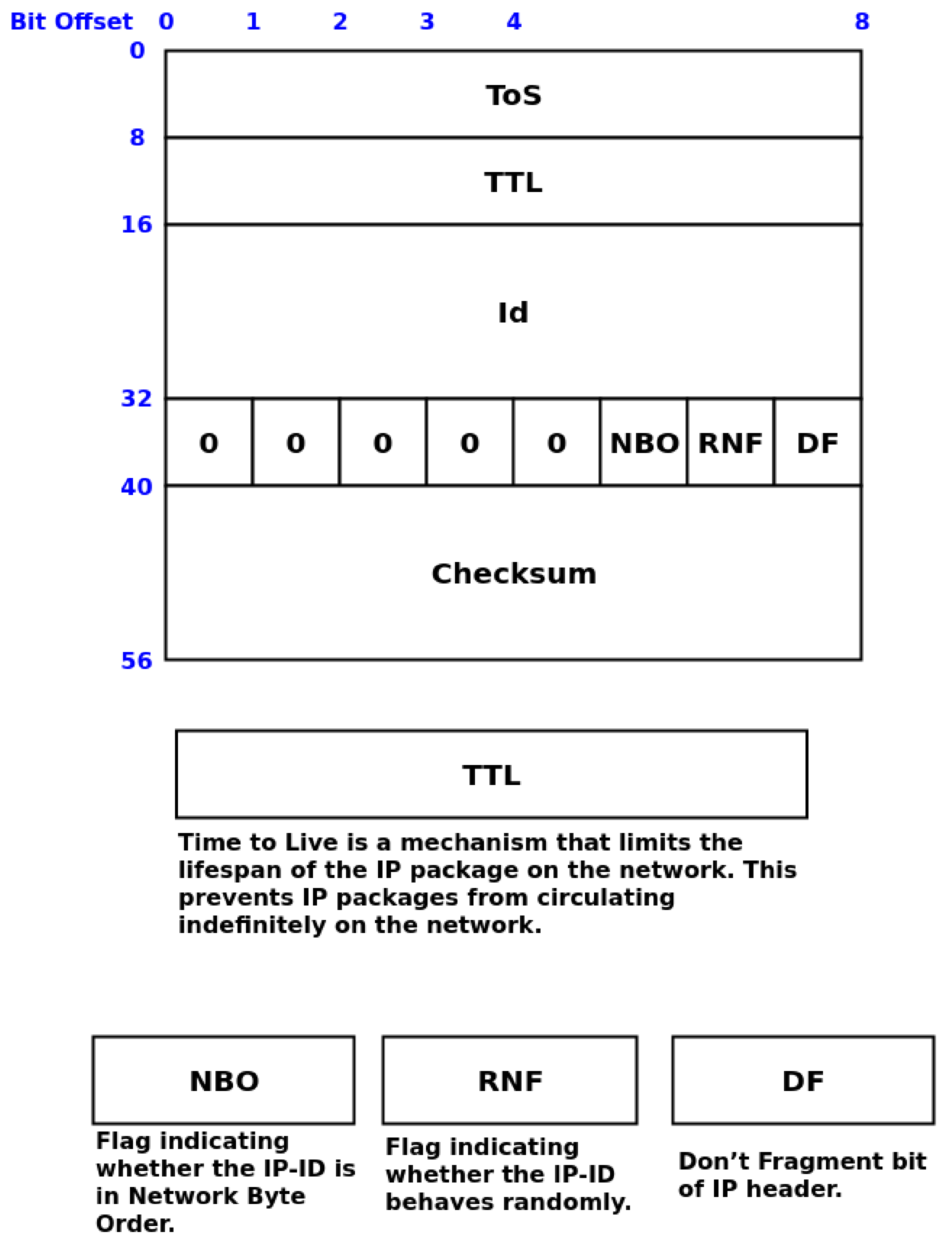

Bit Offset
0
1
2
3
4
8
0
8
16
32
40
56
ToS
TTL
Id
0
0
0
0
0
NBO
RNF
DF
Checksum
TTL
Time to Live is a mechanism that limits the lifespan of the IP package on the network. This prevents IP packages from circulating indefinitely on the network.
NBO
Flag indicating whether the IP-ID is in Network Byte Order.
RNF
Flag indicating whether the IP-ID behaves randomly.
DF
Don't Fragment bit of IP header.

**Figure 3.12:** The dynamic chain for IP/UDP (profile 0x02)

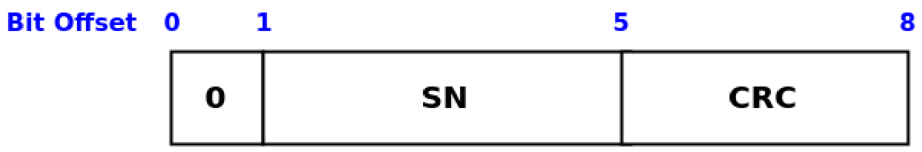


**Figure 3.13:** A UO-0 header

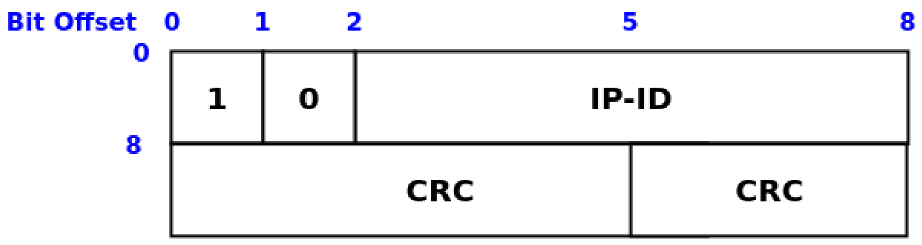


**Figure 3.14:** A UO-1 header

## 3.5 RoHC Operation Assumptions

RoHC operates under a set of assumptions:

- Context identifier - Each package stream has to be associated with an context identifier. The context identifier maps the static and dynamic fields between the compressor and decompressor. This allows for more than one package stream per unit.
- No package reordering - Packages may not be reordered since that would disrupt the state-machines in the compressor and decompressor which in turn would lead to high error rates and the compression factor to be lost.
- The packet type is provided in the compressed header.
- No package duplication - Package duplication would just as package reordering cause a failure in the state-machines.
- The package length is provided by the link layer - To achieve maximum decompression between compressor and decompressor RoHC expects the link layer to declare the package size.
- The compression negotiation is provided by the link layer - The configuration variables negotiated between the compressor and decompressor have to be negotiated by the link layer and provided to the RoHC framework.
- The link layer must also provide package framing that makes it possible to distinguish frame boundaries and individual packages.

Since it is only the sender and receiver on the communication channel, each packet stream has to be linked to a context identifier, CID, the channel between compressor and decompressor has to maintain the packet ordering. RoHC needs a link-layer protocol to negotiate a set of configuration variables. In the case of NB-IoT it is the PDCP protocol that provides the configuration variables. RoHC operates as a sub-protocol to PDCP since it is the PDCP protocol that is responsible for header compression in the LTE stack.

## 3.6 Configuration parameters

To configure RoHC the lower layers needs to provide some data to the RoHC framework. These variables can be seen in 3.15. The two variables of interest in NB-IoT are

- Supported profiles
- MAX_CID - Defaults to 15 but can be as high as $2^{16} - 1$.
- Maximum Reconstructed Reception Unit (MRRU)
- FEEDBACK_FOR

**Figure 3.15:** Configuration variables

the MAX_CID and the set of supported profiles, MRRU and FEEDBACK_FOR are not applicable in NB-IoT [1].

RoHC supports a set of profiles specified in [3], however basestations are not forced to support them all, e.g. a basesattion may only choose to support 0x01 which is theIP-UDP compression scheme. If the basestation does not support the devices supported profiles header compression between PDCP has to bypass the header compression state and send full headers to the basestation.

The MAX_CID defines the maximum active channels on the connection between the compressor and decompressor, it defaults to 15. A MAX_CID of 15(0xF) enables 15 concurrent communication channels between the compressor and decompressor since CID: 0 may not be used as a identifier [3].

## 3.7 Context Damage

Context damage arises when the compressor is in a higher state than the decompressor or if the decompressor has repeatedly failed to decompress messages from the compressor. The context may be repaired by an IR, IR_DYN or, DYN message depending on the severity of the damage.

When a context damage arises the decompressor may send a feedback message, if feedback is possible, and request a IR or DYN message to refresh the session. If the decompressor decides that the full session is damaged it may send a static_Nack which request a reinitialisation of the current context.

If feedback is not possible the decompressor will discard all packages until its context has been deemed repaired by the decompressor.

## 3.8 Feedback

A RoHC feedback element is the first segment in a RoHC message. The feedback element begins with a feedback type octet. The feedback octet is built by 4 static bits signaling that the following segment is a feedback element, followed by a 3 bit field that declares the size of the following feedback data in bytes. If the size segment is 0 the following byte describes the size of the feedback data. The feedback data is profile-specific and includes CID information.

The feedback data starts with the declaration of the CID; if small CID is used an Add-CID octet or if large CID is used a large CID field of 1-2 bytes starts the package. The following data is a feedback, it may be one of two feedback packages. If the size of the feedback data is $< 2$ the feedback data is of type 1. Type 1 feedback data includes only one byte with the sequence number. This is an ACK, in order to send a NACK or static-NACK the type 2 package must be used.

If the feedback data size is greater than 1 it is a type 2 package. The type 2 package starts with a 2 bit field which stores an Acktype. The Acktype is a value $0 \leq x \leq 2$, if the Acktype is 3 the package must be discarded. It is followed by a 2 bit mode field which indicates the decompressors current mode and the mode field is followed by a sequence number.

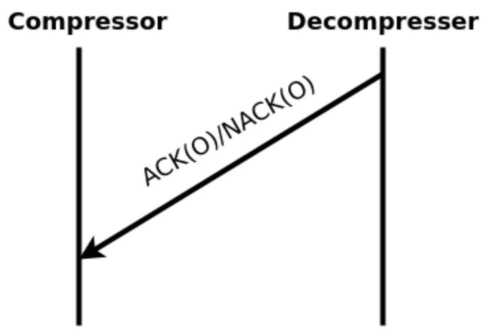


**Figure 3.16:** Transition from U to O-Mode

## 3.9 Mode transition

RoHC is always initialized in U-Mode. The decompressor may at any moment decide to initiate a mode transition between the state machines. A mode transition is initiated by a feedback message or segment from the decompressor carrying a feedback type-2 feedback header. The feedback type-2 segment has to include a CRC to be eligible to initiate a change in mode.

As can be seen in the Figure 3.3 all modes are reachable from any given mode. The transition mechanisms differers between modes which can be viewed in 3.16. The compressor only needs to decompress a single feedback segment to transition from U-Mode to O-Mode. The feedback segment has its mode set to O mode. If the feedback segments checksum is correct the decompressor will update the context to O-Mode. If the segments checksum is incorrect the package is discarded and the compressor will stay in U-Mode. The decompressor will continue to work in O-Mode since U- and O-Mode share the same package structure. The device will update its mode to O-Mode when it successfully decompresses a Feedback element from the decompressor.

In all other mode-transfers the compressor and decompressor makes a three way handshake. The three-way handshake can be viewed in Figure 3.17. It is initiated by a feedback segment from the decompressor. The compressor then continues to send IR/IR_DYN/DYN or UOR-2 packages until the decompressor has successfully decompressed a package. The decompressor then follows with an ACK with the successfully decompressed packages sequence number. When the compressor verifies the ACK sent from the decompressor it updates its mode to the new mode. When the decompressor yet again successfully parses a package both compressor and decompressor has updated its mode to the negotiated mode.

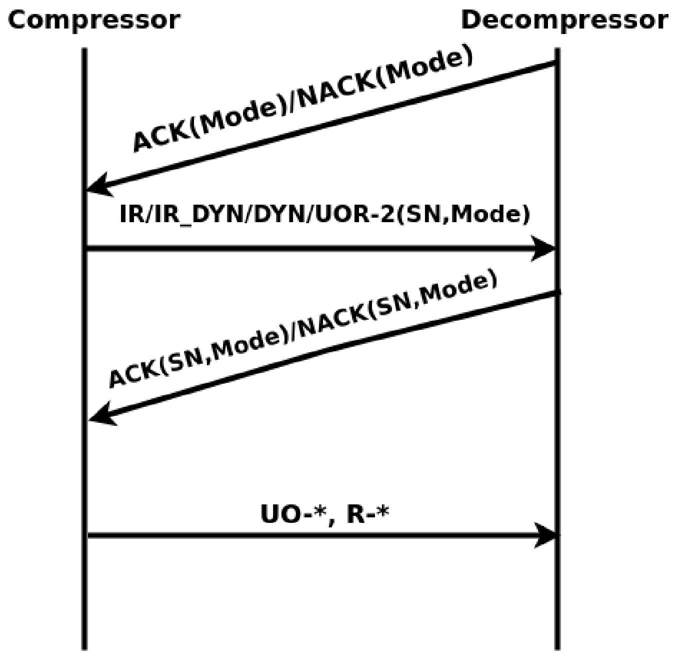


**Figure 3.17:** The generic three-way handshake

## 3.10 Robustness

### 3.10.1 Definition

Robustness is a relative term. Oxford dictionary defines robustness as "The ability to withstand or overcome adverse conditions or rigorous testing". In turn, this definition does translate to robustness within computer science as well but does not explain the term.

In computer science robustness is the ability for an executable or system to cope with errors during runtime. These errors include, but are not limited to, erroneous input, transmission error or external forces (e.g. power cuts). As Robustness is an umbrella term for robust systems within all subcategories, robustness within computer science merely adds another explanatory level. There exists robust networks, robust hardware, robust programming and of course robust header compression. The original definition does not change between different levels, but each level adds their own abstraction and explains it a bit differently. Robust programming much refers to user/programmer interaction, paranoia and dangerous implementations while robust networking refers to structures, unexpected situations such as a node disappearing or a huge load of incoming traffic.

Even though the further down you go the more "rules" are applied, each layer has the same fundamental idea; handle unexpected errors and events in such a way that it is easy to recover or simply continue without being affected.

### 3.10.2 Robustness in RoHC

In the RoHC scheme, robustness implies two things; provide a high compression rate with either improved or at least the same error detection as the uncompressed protocols does and provide a method to recover should errors occur.

Generally, RoHC does comply with the definition of Robustness. Is uses CRC as a method of error detection, regardless of mode. The main difference between modes lies in error recovery. Since not all modes utilises feedback, retransmission is used based on count/timeout rather than on request.

#### Error detection

As stated, RoHC uses CRC as the main error detection scheme. Even though checksum is more commonly used in higher layers, CRC seem to provide a better error detection rate in certain cases vs checksum. [12] In RoHC, CRC is used to detect two types of errors; transmission errors and context damage. Firstly, CRC is computed over the static part and the dynamic part separately. This increases CPU efficiency in RoHC by not recalculating CRC over bits that never change. Note, the static and dynamic parts are calculated in the order they appear in the header and the CRC is calculated over the complete uncompressed header. Given an IR package, the CRC can be used to detect transmission errors in the header. Usually there are more bits in other layers that also provides some error detection of transmissions, but since memory and buffer errors can occur, further error detection is highly recommended. When compressed packages start to arrive, the decompressor can recreate the original header, verify its integrity with the CRC and detect any context damage. Depending on mode and specific rule set, certain number of package must fail the context check before the decompressor reacts.

RoHC uses three different CRC, 3-bit, 7-bit and 8-bit. Each have their own use and all depend on what state, mode and package is being sent. Depending on what profile is being used (TCP, UDP or RTP) and in what state, RoHC can accept to also transmit that protocols checksum increasing error detection but decreasing the compression rate. As an example, in theIP/UDP scheme, an IR_DYN does include a CRC and checksum from theIP header.

#### Recovery methods

Recovery methods differ heavily between different modes. One can state that the robustness differ largely depending on mode. U-Mode does provide a recovery method but is also the mode that can cause immense package drops. O-Mode and R-mode utilise feedback and may limit how many packages are dropped.

U-Modes recovery method can hardly be called a recovery method. Mainly since it does not recognize that an error has occurred and only follows a pre-specified pattern. Such a pattern might be sending ten IR_DYN packages, followed by one FO package and one SO package on repeat. The compressor does not know if the decompressor have context damage, but since it is retransmitting the initialisation package so often, there is a high probability of recovery without loosing a large amount of packages. If this pattern would consists of only one IR_DYN followed by ten FO and 200 SO, the first package must arrive without damage else it is guaranteed that a high amount of packages will be lost before any attempt at recovery is made. Thus, balancing compression rate and recovery packages is highly important to not lose packages or waste compression. Note, 210 packages depending on context can do immense damage in some cases where 1 package is transmitted each day compared to VoIP where a huge amount of packages can be transmitted every second.

On the other hand, O-Mode and R-Mode both uses feedback which in turn means that the compressor can react to what the decompressor needs. R-Mode is a very strict mode and at the first hint of error, the decompressor will ask the compressor to resend context. This does increase robustness a lot. O-Mode is somewhere in between, meaning it does provide feedback but does not require the same robustness as R-Mode. This does mean it can maintain a higher compression rate while risking some package drops.

All modes does have some error recovery method, although very varied. Depending on the requirements, not all modes are optimal. While robustness is important, other aspects may indeed be more important. Thus, RoHC can live up to its name fully with R-Mode and really be a robust protocol while on the other hand RoHC can maintain some robustness and help decrease network congestion.

Chapter 4

# Method

To evaluate the RoHC frameworks potential for IoT devices, the experiments are conducted with an implementation of RoHC transporting IP/UDP (RoHC profile 0x02).

The Section will continue with a brief description of the frameworks implementation and continue to describe the tests that were performed and what data to be examined.

## 4.1 Design Choices

The RoHC implementation is based on two parts. A client-server architecture that encapsulates the RoHC framework and the RoHC framework itself. Both are written in C. The framework were implemented in C since it gives a clearer result of the experiments and its memory efficiency. C does not have a memory overhead and source files will be compiled into an executable file where the size is clearly defined.

The main design pattern in the framework is the strategy pattern since the implementation only supports a single profile at the same time. The implementation has 3 sets of compressors and decompressors that are responsible for decompressing/compressing data according to one of the three states. When a mode transition is completed a pointer to the compressor/decompressor functions is changed in the RoHC context struct. In a way the implementation is object oriented with an object at both the compressor and decompressor. The choice of the object oriented approach came of the implementation of the RoHC state machines and the option of using several parallel data streams, each stream has its own context and requirements. Note that C is not a object oriented language but a procedure oriented language and the wording of "object oriented approach" is figurative.

### 4.1.1 Framework

The RoHC framework was implemented with a strict modularity in mind. The architecture consists of two main components.

- Generic RoHC mechanisms
- Profile specific code

There is a RoHC main program which exposes the compressor and decompressor to the encapsulating program. The RoHC module needs to be initiated by an encapsulating protocol or program. It requires a set of variables:

- A pointer to a function/device-driver that is responsible for the data after it has been compressed.
- A pointer to a function/device-driver that is responsible for the data after it has been decompressed.
- The largest CID that may be used. (MAX_CID)

The program only supports one profile at a time, this is done during compile time to minimize the executable file size. In our case this would be the UDP_IP(0x02) profile in RoHC.

### 4.1.2 Data Structures

The only generic data structure is the rohc_context, our context example can be viewed in the Figure 4.1. This is the object that represents the RoHC state machine. It holds the specific data needed by the RoHC module to operate. The static and dynamic chains are profile specific data structures that keeps track of the static and dynamic parts of the profile specific header(s) which is used to reconstruct the compressed header in the decompressor.

```
struct rohc_context {
uint8_t nbr_packages;
uint8_t mode_transition;
uint16_t CID;
uint16_t SN; // sequence number
uint32_t max_CID;
enum ROHC_STATE state;
enum ROHC_MODE mode;
enum ROHC_STATUS (*writeback)(uint8_t *data, uint16_t length);
enum ROHC_STATUS (*readback)(uint8_t *data, uint16_t length);
enum ROHC_STATUS (*compressor)(uint8_t *data, uint16_t length,
   struct rohc_context *ctrl);
enum ROHC_STATUS (*decompressor)(uint8_t *data, uint16_t length,
 struct rohc_context *ctrl);
struct static_chain s_chain;
struct dynamic_chain d_chain;
};
```

**Figure 4.1:** The RoHC_context structure

To minimize the branching factor of the RoHC module the specific compressor and decompressor methods are used depending on the contexts current mode. This also makes it easier to append supported profiles.

#### Memory Allocation Strategy

The RoHC framework operates on static arrays which makes it easy to calculate the memory requirements for the framework. The use of static memory compared to dynamic memory allocations also increase the efficiency since dynamic memory allocation would increase the time the CPU has to spend on searching for memory blocks large enough to fit the requested memory space. However the static allocation does not always provide an optimal allocation of space.

### 4.1.3 Test framework

To test the RoHC framework we developed a client-server solution where the client represents a compressor and the server represents a basestation decompressor. Each scenario starts with the compressor sending an IR/IR_DYN message to initiate the package stream. Where the decompressor may, depending on the scenario, send feedback messages or just sit passively and listen.

The server and client communicates over Unix domain socket, the Unix domain socket works as a regular TCP socket with the limitation that the Unix domain socket only operates within the same physical machine. The risk of residual bit errors in Unix domain socket are so small that during our experiments we are not taking them into account and may therefore not test the framework in a perfect scenario.

## 4.2 Tests

### 4.2.1 Compression Efficiency

To determine the compression efficiency five possible scenarios were designed. All scenarios are based on data streams that consists of 200 packages.

- U-Mode with a movement pattern of (IR,FO,SO)(5,5,5)
- U-Mode with a movement pattern state of (IR,FO,SO)(2,5,10)
- U-Mode with a movement pattern of (IR,FO,SO)(1,2,20)
- O-Mode (sends feedback once each 5 package received.)

To measure the difference between compressed and uncompressed we're going to count the header size and full package size at the compressor and the data received at the decompressor. Then compare the difference between the uncompressed and compressed data streams over time.

To be able to evaluate the compression ratio between the different states and modes we are going to compare how each mode/state affects the compression ratio. To get a comprehensive view of how RoHC changes the data-stream, we are going to measure:

- How does the compression rate change over time in a package stream.
- How much RoHC impact a package stream carrying 255 bytes of payload.
- The difference in total amount of transmitted data between a uncompressed data stream and a RoHC package stream.
- How the payload size affects RoHCs impact on the transmission window.
- How different modes and states affect the compression rate

### 4.2.2 Effect of RoHC on Energy Consumption

In order to be able to measure the effects of RoHC on an IoT devices energy consumption we are going to make our calculations based on the results of Amen Hussain's Master thesis: "Energy Consumption of Wireless IoT Nodes" [8]. In conjecture with the results from the Compression efficiency section we aim to see if RoHC may affect the overall energy consumption of an IoT device.

We are interested in examining:

- Does RoHC affect the energy consumption during RX?
- Does RoHC affect the energy consumption during TX?
- Is there a significant decrease in energy consumption by using RoHC compared to a uncompressed data stream?

### 4.2.3 Hardware acceleration

To get indications whether hardware acceleration may be a good suite for the RoHC framework we are going to review and search for code that is computationally intensive and see if it can be hardware accelerated. After making the assessment of computationally heavy code we are going to cross check our results with research on the subject. In conjecture with those results draw conclusions from the efficiency of hardware acceleration in RoHC.

We are interested in answering:

- Is RoHC suitable for Hardware acceleration?
- Is a hardware accelerated implementation of RoHC suitable for an IoT device?

## 4.3 Assumptions and Limitations

- Since the experiments are conducted using Unix domain sockets the risk of residual bit errors are negligible.
- Complete and correct package is sent to the RoHC module.
- Since the test environment uses Unix domain sockets to represent the lower layers the test framework cannot reliably replicate an error prone link.

Chapter 5

# Results

## 5.1 Compression Ratio

This Section presents findings regarding RoHC's compression rate with IP/UDP packages. There are two different definitions of compression rate depending if there is a payload present or not. Firstly, no payload is present which means compression rate is defined as compressed vs uncompressed header. Secondly, a payload is present and distinguishes how much difference header compression makes on an entire packet. The compression of the header will give the effectiveness of the header compression scheme but when looking at the compression rate of the package as a whole we will observe the impact of the compression on the transmission time.

Figure 5.1 shows the package sizes of each header we have examined. One thing to notice is the fact that even if a device only sends IR_DYN packages it will still outperform regularIP/UDP packages.

| Packet | Relative size |
|---|---|
| IR + DYN | ~103.5% |
| IP/UDP | 100% |
| IR_DYN | ~92.8 % |
| DYN | ~39.2% |
| UO-0 | ~7.1 % |

**Table 5.1:** Header compression rate of different packages in relation to IP/UDP

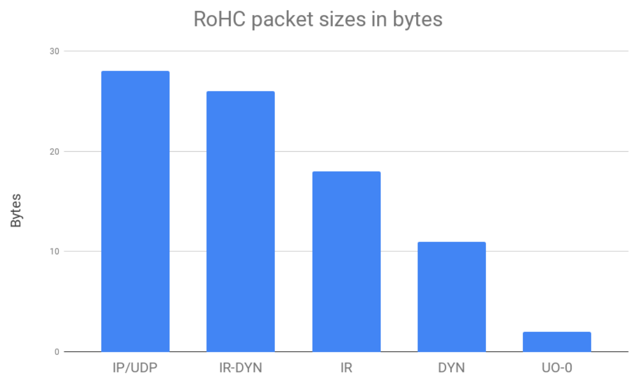


**Figure 5.1:** Header sizes

In the Figure 5.2, we can see how our tests differ in the amount of header data that has been transmitted. All RoHC data-streams originate with a full IR_DYN message where the reference model sends full a IP/UDP header. Then all the data streams diverge depending on the U-Modes movement pattern and the O-Mode depending on feedback from the decompressor.

In Figure 5.2 one can see how the movement pattern affects the amount of header data that is transmitted. The sum of header bytes in the movement pattern (IR,FO,SO)(5,5,5) is seen to increase noticeably faster compared to the other U-Mode movement patterns. This is due to the amount of IR and FO as well as the short package cycle. Because an IR message contains *thirteen* times more header-data than a SO package and a FO package contains *five* times more header-data than a SO package the sum increases rapidly. Each change in the derivative of each data stream signals a state change in the compressor. The DYN package contains 11 bytes of header-information and directly follows each IR_DYN package. When the compressor state changes to SO the header size is decreased to two bytes of payload, this is made possible by the redundancy in the static and dynamic header fields. From the graph in Figure 5.2 shows that the UO-0 packages contribute to optimal compression ratio.

From the data presented in Table 5.2 one can see that the header-data sent in O-Mode is noticeably smaller than in U-Mode, this is due to the decompressor being able to give feedback regarding the success or failure of decompression. This does however require a feedback channel to the compressor, which U-Mode does not. Figure 5.2 presents the tables metrics over time.

Note that the following sequence of movement pattern are not enforced by the RoHC standard but of our own implementation, the U-Modes movement patterns are undefined and at the developers choosing. The results however are not affected by how the compressor transitions its state but the amount of packages sent in each state when looking at the average header-payload of a state-cycle. The previous statement assumes that the compressor starts in IR and passes through all states in its state-cycle.

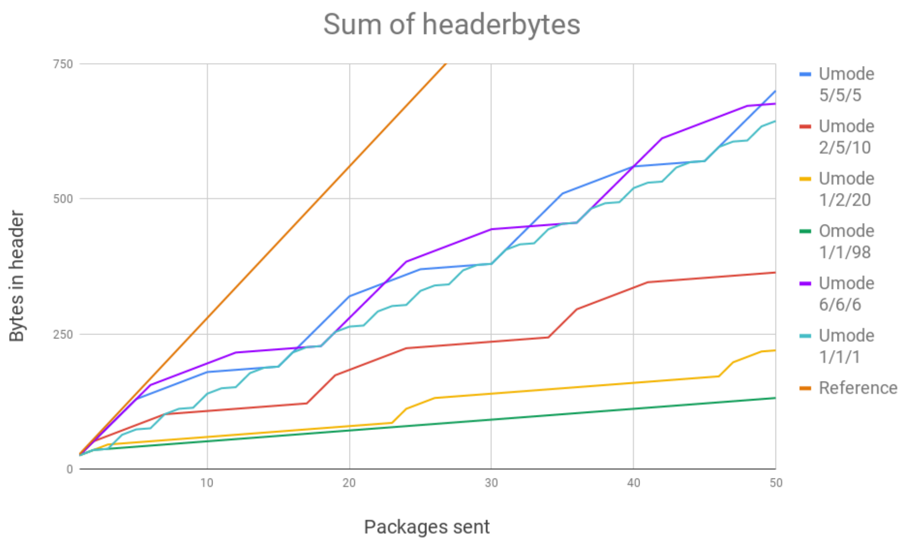


**Figure 5.2:** Sum of header bytes

| Variant | No. of bytes | Size compared to reference |
|---|---|---|
| Reference | 2800 | 100% |
| Umode 1/1/1 | 1280 | 45.7% |
| Umode 2/5/10 | 728 | 26.0% |
| Umode 1/2/20 | 400 | 14.3% |
| Omode 1/1/98 | 232 | 8.3% |

**Table 5.2:** Sum of header bytes in numbers

From the graph in Figure 5.3 one can read out the effect RoHC has on the transmission window based on the size of the payload. The effect of RoHC decreases exponentially when the payload size increases.

This graph is the result of n-bytes payload sent with a UO-0 header compared to a fullIP/UDP header with the same payload.

Figure 5.4 shows the compression rate of the header-payload over time. As seen previously in 5.2 O-Mode provides slightly better compression rate due to the information exchange with the decompressor. However it is interesting to see that the graphs compression rate has stabilized after 100 packages. This can be used to give an approximate compression rate of each mode and movement pattern provided in this experiment. In the graph we can easily distinguish when the compressor changes its state when the derivative changes drastically. Another thing to notice when looking at Figure 5.4, the shorter the compressors state-cycle is, the worse the compressor performs.

As seen in Figure 5.3 the impact of RoHC on the transmission time is still significant.

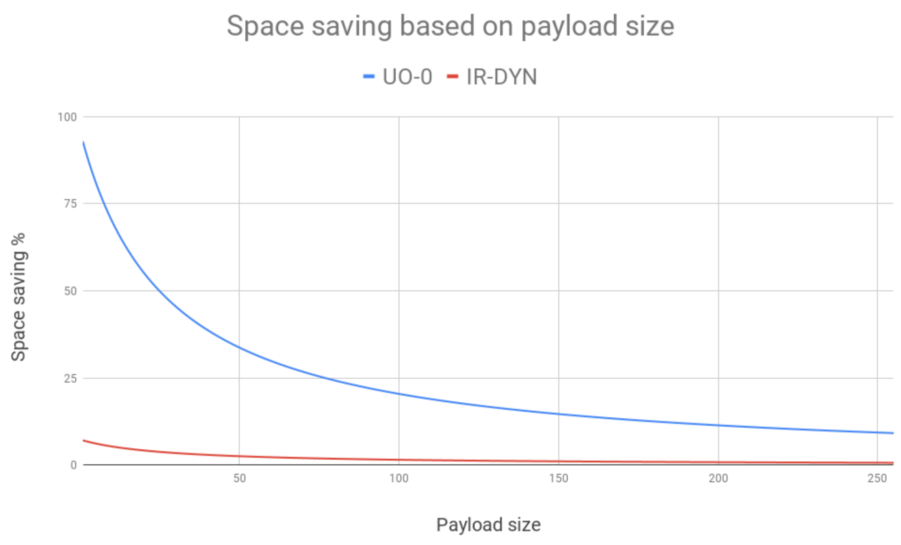


**Figure 5.3:** Space saving based on payload size

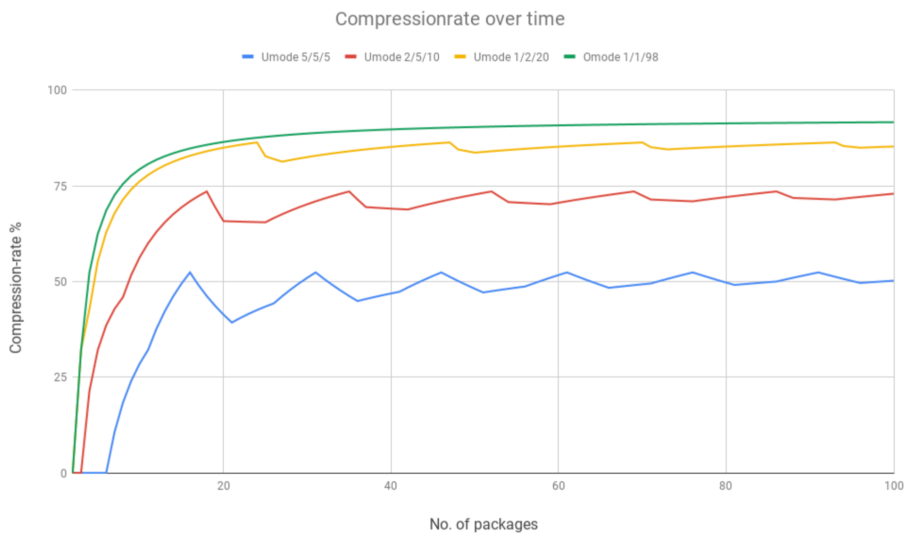


**Figure 5.4:** Compression rate of header data over time

The data provided in Figure 5.5 shows the effect of each Mode and movement pattern on the total amount of transmission data during the transmission of 100 packages carrying 20 bytes of data. The difference in transmission data between the data-streams is shown in the Table 5.3. The Table shows the impact of RoHC has on a data stream carrying 20-

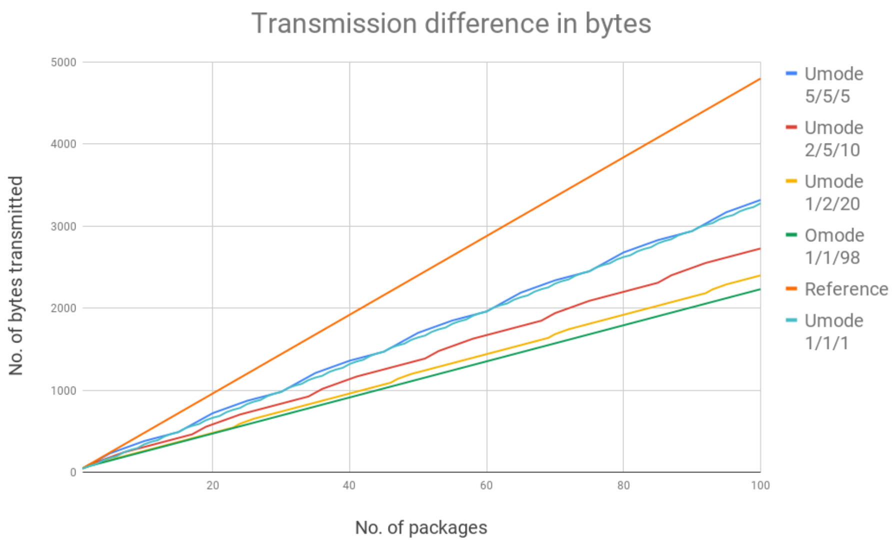


**Figure 5.5:** Difference in transmitted data with 20-bytes payload

bytes of payload/package. The O-Mode cuts the transmission payload to more than half of the reference which is a significant decrease of the transmission time needed to send the data.

| Variant | No. of bytes | Size compared to reference |
|---|---|---|
| Reference | 4800 | 100% |
| U-Mode 5/5/5 | 3320 | 69.2% |
| U-Mode 2/5/10 | 2728 | 56.8% |
| U-Mode 1/2/20 | 2400 | 50.0% |
| O-Mode 1/1/98 | 2232 | 46.5% |
| No header | 2000 | 41.7% |

**Table 5.3:** Sum of bytes in numbers, header and 20 bytes of data

## 5.2 Energy Efficiency

The results provided in Energy Consumption of Wireless IoT Nodes by Amen Hussain shows the power consumption of a LoRa-WAN IoT device [8]. The results show that the power consumption of the IoT device are closely related to the package size.

> "During these experiments, we observed that the energy consumption of /.../ increases linearly with respect to packet size" [8].

The results also show that the difference between pre- and post-radio transmission phases are negligible when the payload is increased.

The author continues to measure the difference in energy consumption regarding the size of the RX payload. The results were that the energy consumption during the RX phase were not depending on the amount of bytes received. To quote the author:

> "Tab. 4.6 proves that the frame size does not influence the energy consumption of mDot in the process of receiving. The difference in the average duration is around 1 ms which leads to a conclusion that after the receive event the mDot processes the data and it might take more time to process 240 bytes as compared to 120 bytes" [8].

mDot in the quote above refers to the IoT device on which the energy measurements were taken from.

The master thesis continues to provide data on how different scenarios affect the energy consumption.

In chapter 6 through 8 the author presents data on the distribution of the energy consumption of an IoT device with different sensors applied and different scenarios. In these chapters the energy consumption from the TX phase is between 0% and ~28%.

In "NB-IoT System for M2M Communication" by Rapeepat Ratasuk et.al the authors tried to calculate the battery life of an NB-IoT device given four scenarios.

| Message size / message frequency | Stand-alone (years) | In-band (years) |
|---|---|---|
| 50 Bytes / every other hour | 2.6 | 2.4 |
| 200 Bytes / every other hour | 1.2 | 1.2 |
| 50 Bytes / daily | 18.0 | 16.8 |
| 200 Bytes / dayly | 11.0 | 10.5 |

**Table 5.4:** Battery life of an NB-IoT device
[14]

These calculations have been made under the circumstances described in Table 5.4. The calucaltions are based of the asumptions presented in Table 5.5

| | Power[mW] |
|---|---|
| Battery power during Tx (Assuming 44% PA efficiency) | 543 |
| Battery power for Rx | 90 |
| Battery power when idle but not in power save state | 2.4 |
| Battery power in power save state | 0.015 |

**Table 5.5:** Power consumption assumptions used for the calculations in Table 5.4
[14]

Based on the results in Section 5.1 the energy consumption could be reduced by applying RoHC since the TX may have a large effect on the energy consumption. The energy consumption could be reduced by somewhere between 0% and 26% depending on the payload size and the energy consumption pattern of the device itself.

## 5.3 Hardware Acceleration

The main components of the RoHC framework is the compressor and decompressor. The compressor initializes the context between the compressor and decompressor and compresses the packages delivered by layer 3. The flowchart of the compression process can be viewed in Figure 3.1. In the flowchart one can see the high branching factor from the compressor, all packages have their own fields and each package sent in SO can include an extension and there is a large number extensions that are profile specific.

The decompressor has a similar structure as the compressor and it can be viewed in Figure 3.2. The main differences is that the decompressor decodes the encoded information for verification or to extract dynamic data from a message header or extension.

One thing we observed during the review of the source code were the lack of computational intensive functions. There are some encoding schemes such as the w-LSB and CRC that may be counted as computational intensive. However CRC and w-LSB encoding is not utilized on each package, both CRC and w-LSB encoding mechanisms are used on some header extensions and in the base headers.

Both the compressor and decompressor has to utilize the corresponding CRC and w-LSB functions to verify and update their dynamic context. These functions have negligible differences in computational complexity to the encoding mechanisms.

In the results of Hardware Acceleration of the Robust Header Compression (RoHC) Algorithm [11] one can find data on the hardware requirements of a pure software, a hardware-software, and a full hardware implementation of the RoHC framework. This data is shown in Table 5.6. What is interesting with their result is the impressive throughput and efficiency of the Full-Hardware implementation. The full hardware implementation is delivering a ~5100% increase in throughput for a ~230% increase in ALUT and a ~51% increase in Flip-Flops. That is an incredible trade-off. The full hardware implementation is not surprisingly performing better in both CPU frequency and memory usage as well as external memory usage.

| Design Metrics | SW | HW-SW | Full-HW |
|---|---|---|---|
| Throughput [Packages/second] | 4716 | 6250 | **244 000** |
| Power Consumption [Watt] | - | 1.5 | 0.9 |
| #ALUT | 2215 | 2578 | **7477** |
| #Flip-Flops | 1680 | 1680 | **2533** |
| Frequency [MHz] | 150 | 150 | 90 |
| Memory [kB] | 75 | 75 | 2 |
| External memory bandwidth [kbps] | 288 | 288 | 212.8 |

**Table 5.6:** Hardware metrics for RoHC implementation
[11]

Chapter 6

# Discussion

During the introduction we defined our research questions as follows:

- What is the compression rate of RoHC?
- Can RoHC increase the battery life of an IoT device?
- Can RoHC efficiently be implemented using hardware acceleration?

## 6.1 Compression Efficiency

Our initial results show that the compression rate of RoHC is largely based on how the states are traversed, which mode is applied and, payload size.

Given the data shown in Figure 5.1 and Table 5.1 we can conclude that the RoHC state greatly affect the compression rate, since IR-*, DYN and, (0-*,1-*,2-*) packages are state specific. This is due to RoHC being a lossless compression protocol. The state-transition is decided based on mode and the mode specific implementation. This implies that the RoHC will have a variable compression rate between ~7 and ~93%.

As can be seen in Figure 5.2 even U-Mode with a short transition cycle has a significant impact on the total amount of header bytes for a closer inspection of the results see the Table 5.2. We think it is interesting that the short transition cycle of the (n,n,n) transitions are equal or better than the counterparts. This is interesting to us since all of the (n,n,n) should move to the same asymptotical value but the short state cycle (1,1,1) seems to outperform the state cycles with a larger transition cycle. We believe this to be the result of the short state cycle diverging less from the asymptote.

As seen in the Figure 5.3, there is no negative effect of applying RoHC that we have observed. One can argue against this statement: It is possible to send the static and dynamic chains separately in an IR and a DYN package repeatedly. Thereby get an increase in transmitted bytes in regards to the original header. This is of course an edge case and requires malicious intent, because of this we have chosen to ignore the fact.

However the effect of RoHC may vary depending on the payload size as can be seen in 5.3. This shows that the space savings of RoHC will be most noticeable on payloads below 75 Bytes. However this does not remove the effect of RoHC on larger payloads. Even large payload sizes can benefit from RoHC, especially since the space saving is not linear. This does imply that even with maximum payload size, RoHC does provide a small portion of space saving. Even though space saving is small, it does make a significant impact over the course of a lifetime where every byte matters. Of course, RoHC will become more effective when used with small payloads. Based on this result we think that RoHC is best fit to operate in scenarios where a small payload is sent.

In order to have the best transmission rate, O-Mode is the way to go. As seen in Figure 5.2, a perfect O-Mode provides a larger compression ratio than U-Mode. However, it increases network traffic by requiring feedback data. Since a feedback message is by a variable length but at least 4 bytes, a perfect O-Mode will only transmit 2 bytes header but will receive 4 bytes feedback usually, making it just as good from a network perspective as U-Mode. Congestion may certainly have been a problem when RoHC was introduced, but is hardly a limiting factor today. The limits are rather in the IoT device, whether it can handle feedback or not based on preferences. As also seen in 5.2, U-Mode comes in many variants and shapes. As aforementioned, the movement pattern is defined by the implementer and is not set in the standard. This differs from O-Mode, where movements are made based on feedback and are thus much less predictable. Nevertheless, movement patterns have a considerable impact on the compression ratio. It is possible, by just considering transmitted data, that U-Mode will beat O-Mode, although the reason probably being a bad connection. This does mean that O-Mode will have more transmitted data, but a better success rate making it still favoured over U-Mode when possible.

Further, there is a large gap between different U-Mode movement patterns. This is to be expected but comes with a trade-off; Robustness vs. compression. O-Mode can maintain a good robustness while maintaining a good compression rate and instead place more load on the network. U-Mode must choose, full compression, full robustness or somewhere in between. What state to follow is as stated chosen by the implementer, but is suitably chosen based on the reliability of the connection. As seen in Figure 5.3 a U-Mode ($\infty$,0,0) is still a viable route to go. Another viable route is (1,1,$\infty$). As stated in Section 3.10, the latter will have a great compression at the risk of loosing packages. Note; consideration will have to be taking in regard to what is the risk of context damage and how many packages can we afford to lose. If the risk of damage is low and there is a high tolerance for errors, an aggressive movement pattern can be favourable. Preferably, U-Mode starts of with a few IR_DYN packages to initialise the stream. After that, IR-state does not have to be visited as often or as long, making the compression rate higher than a regular (n,n,n) cycle.

## 6.2 Energy Efficiency

We initially thought that RoHC at best would have no to a small effect on the energy efficiency of an IoT device. After reading [8] we started to suspect that RoHC might actually have a significant impact on the battery length of an IoT device.

Due to the difference between Lora-WAN and NB-IoT these results are not equivalent but they provide a ballpark Figure. The difference lies in the connection mechanism between the device and a base station. Since LoraWAN and NB-IoT have different connection mechanisms the setup time may vary which in turn would affect the energy consumption. Another difference between LoraWAN and NB-IoT is the difference in the carrier wave. NB-IoT is using a different base band than lora and operates with another standard which uses a less energy consumption during TX.

Since the energy consumption during the RX phase is not linear in the number of bytes that is received, the energy consumption of the RX window is largely based on the energy consumption of the IoT device itself. The energy consumption when sending data is on the other hand depending on the amount of data that is transmitted, the difference in energy consumption is due to the antenna. As we can see in the Figure 5.3 the data saving of RoHC is correlated to the payload size, this is due to RoHC only compresses the header data. This would make RoHC's effect on the energy consumption vary depending on the payload size where small payloads, around 0-75 Bytes, would prove a significant impact on the energy consumption. Since the RX phase are largely dominated by the time it takes the IoT-device to negotiate the connection to the base-station the energy consummation of the RX window can be seen as a constant while the TX phase can be decreases by utilizing RoHC.

As can be seen in Table 5.5 the power consumption of TX is roughly 6 times larger than that of RX. Since we have not made any real experiments of the power consumption pattern of a cellular IoT device we do not know how large portion of the energy consumption the different phases are. We can only say that RoHC will affect the TX phase.

It is hard to estimate the effect of RoHC on cellular IoT-devices since the LTE-network is using time slots to transfer data between the device and the base station. The transmitter needs to fill the transmission window it was given by the base station. If a message is smaller than the transmission window the device may choose to increase the coding of the message which will increase the odds of a successful package delivery and therefore decrease the risk of retransmission. This may also add to the decrease the battery consumption when looking at the entire package stream while when looking at a single package it may stay the same.

Based on the data provided in "NB-IoT System for M2M Communication" RoHC would produce a decrease in energy consumption between ~0 and 26% depending on the payload size, residual bit errors, transmission window etc.

If our results are verified it would be incredible news for all IoT hardware developers. It would increase the devices lifespan and in doing so lessen the environmental footprint of IoT in general.

## 6.3 Hardware Acceleration

As stated in the results the RoHC framework has very little computationally intensive work. The only functions that can be viewed as computationally intensive are the CRC calculations and the W-LSB encoding. However they are of complexity $O(n)$ where $n$ denotes the number of bits in the package. As seen in Table 5.6 the increase in throughput by a partial or full hardware acceleration of the RoHC framework is between 32.5% in the HW-SW solution and 5074% in the full-HW solution. Despite the significant increase in throughput it might not be worth the increase in chip-area or the implementation time when looking at it from the device perspective. This is because the chip-area may be a selling point for a hardware manufacturer. Another reason may be the use case of the device, if the device only sends a small amount of payload and infrequent the use of a hardware acceleration may be negligible. A cellular-IoT device which sends bursts of packages may have a decrease in power consumption due to the increased throughput and operational time. However we think the difference in energy consumption due to hardware acceleration is marginal at best.

It is hard for us to argue one way or the other regarding the application of hardware accelerated RoHC in cellular IoT devices since our research and experience in this area is limited. Because of our limited experience we choose to not express general recommendations due to our belief that miss information is worse than the lack of information.

## 6.4 Time to market

RoHC is a fairly complex protocol. It is easy to understand the basic function, to remove overhead. But, in order to make the protocol robust while supporting many underlying protocols, it requires different solution with many edge cases to guarantee full compression of several protocols. In turn, this make it very complex to understand and difficult to implement. During the implementation phase we asked our-self whether or not this would be worth the while to implement into a cellular-IoT device. Whether the results are giving a large enough edge to motivate a large amount of man hours to implement RoHC or not. As mentioned earlier in both compression rate and energy efficiency we believe RoHC to almost be a holy grail of cellular-IoT. It is possible to improve lifetime enough to make a significant impact on the market. This makes an implementation worth the time.

Of course, RoHC does add another layer of complexity to a product, providing another point of possible failure. The protocol is robust and set in stone, but it still requires a large amount of code to work. This does imply that the product has a greater risk of bugs. To quote the book Code complete there are "about 15 - 50 errors per 1000 lines of delivered code." [5]. It is a huge number that varies depending on project, programmer and many other factors. Our implementation of only IP/UDP consists of around 1000 lines of code. This would imply that our code contains at least 15 errors. This only emphasizes that there is a risk when implementing RoHC, that very well can affect the stability of the product. Further, by implementing RoHC a company is most likely delaying the release of their product. This may lead to loss of income, a loss greater than the gain of RoHC. In most cases it is impossible to say which route is the best. Strategic calculations such as a risk/reward calculation can be used to help make a well balanced decision. There are many aspects in this report that can help make that decision.

In the aspect of time to market we would argue that a software implementation would probably be the best solution since it has a shorter time to market, it allows for bug-fixes and, it is flexible for updates in general.

From our standpoint and from the results we have gathered, everything points towards that RoHC is worth implementing.

## 6.5 Sources of errors

In every experiment there are sources of error, this thesis is not different. It is worth to point out that these does not skew the result or make it inadmissible, but must be taken into consideration before a possible implementation.

Firstly, our test environment is not affected by external error sources such as residual bit errors. This implies that we have not been able to test the robustness of our implementation. There have been no residual bit errors, unstable connections or lack of signal. It means that many of our results are very optimistic and in some cases equal to a theoretical best.

Secondly, we have used Linux domain sockets in our implementation, which implies that there exists no wrongful transmissions. It also states that there is no package loss and thus does not put a strain on the system. Since we have not used hardware to test on, we have not been able to measure certain aspects, e.g. power usage. If we would have done this, we would have skewed the result towards specific hardware.

Thirdly, the measured implementation only containsIP/UDP support and has limited support for extensions. Since our baseline did not contain them and our interpretation is such that we do not believe these will be used, support has not been implemented. There are certain cases where these can and should be used although that would demand other aspects of the system and would not fit the scope of a NB-IoT device.

There are certainly aspects that may alter the result in a negative way. We can not find any sources of error that would better the result. Therefore we can conclude that our results most likely represents a best-case scenario of RoHC. We believe the data presented can be used as a baseline for further research.

## 6.6 Conclusions

From our results and discussion above, we have concluded that RoHC would make a great addition to cellular IoT devices. This is due to the increase in battery-life.

The main drawback of RoHC is its high branching factor. The high branching factor increases the required chip-area of an hardware implementation. This makes us believe that a hardware accelerated version of RoHC is not the best option for a cellular-IoT device. However a pure software implementation would increase the power usage of the calculations while have a smaller chip-area and being more easily modified.

We believe that RoHC will make cellular-IoT devices better and prolonging their battery-life noticeably due to RoHCs impact on the transmission window and the lower risk of residual bit errors compared to a non compressed data package.

Chapter 7

# Future Work

We believe there is still much research to be done. Not only concerning RoHC, but IoT in general. Although we've done some initial work regarding the effects of RohC, it lacks certain aspects as mentioned in Sources of Error. We do think that our thesis, stand alone, can get a good understanding and motivate the use of RoHC. Although, there is certain aspects that are critical to a product before implementation.

Our research questions have been focused on providing a overview of the RoHC protocol and investigate the usefulness of RoHC in a cellular IoT device. In future research we would suggest to investigate the following:

- How is the compression rate affected by burst of bit-errors?
- Does the error mechanisms in RoHC increase the amount of correct data to its destination.
- How does RoHC compare to a regularIP/UDP stream in an error prone network? Does this affect the energy consumption noticably?

Regarding our findings of RoHC's affect on the energy consumption of an IoT device. Since we have not conducted experiments of our own we would suggest future research on:

- Exact measurements regarding how RoHC will affect the energy consumption of an IoT device.
- If RoHC will provide a significant decrease in energy consumption even on error prone links.
- How is the energy consumption affected if RoHC is hardware implemented?

Chapter 8

# Acknowledgments

The authors of this thesis would like that thank our families, for continuous support. We would also like to thank our supervisors, Stefan Höst and Steffen Malkowsky for their immense patience and great answers to all kinds of questions. Further we would like to thank EIT and LTH for the preparatory work during five years. Special thanks to Fredrik Knutsson and Thomas Lundgren for your insight in cellular IoT.

We'd like to thank the Open Source community for their tireless work and dedication. Their work lays a cornerstone for the technological advances we see today.

# Glossary

**Bidirectional Optimistic Mode** is one of three modes of operation in Robust Header Compression. It utilises irregular feedback from the decompressor to acquire a high compression rate and robustness.. 17

**Bidirectional Reliable Mode** is one of three modes of operation in Robust Header Compression. It utilises large amounts of feedback from the decompressor to acquire a high compression rate and robustness.. 17

**Compressed Transport Control Protocol** is the first header compression protocol.. 15

**Context Identifier** A number between the 1 and $2^{16} - 1$ that is used to map a context to a data stream in RoHC.. 24

**File Transfer Protocol** is a network protocol used for the transfer of files between a client and server.. 15

**First Order Compression** is one of three compressor states in RoHC. It signalizes the that the compressor which to either update or initialize the dynamic context variables. The second order compression is the compressor state that sends DYN messages exclusively.. 17

**Initialization & Refresh** is one of three compressor states in RoHC. It signalizes a full update of the static and possibly the dynamic fields in the RoHC Context. IR may also refere to the IR package which is the package sent from the compressor with the contexts static information.. 17

**Initialization & Refresh + Dynamic chain** is an IR package that contains the dynamic chain as well as the static chain.. 17

**Initialization & Refresh of the Dynamic chain** is a RoHC package much similar to the IR package, the difference is that the IR package is carrying the static chain and the DYN message is carrying the Dynamic chain. The DYN message may also convey data to be sent across the network boundary.. 22

**Internet of Things** is a umbrella term for small self-sufficient embedded devices that is supposed to be able to run without maintenance for 10+ years.. ix

**Internet Protocol** is the primary protocol used to route datagrams across network boundaries on the Internet.. ix

**Linux** is a generic term referring to the family of Unix-like computer operating systems that use the Linux kernel. 57

**Local Area Network** is a computer network that is confined to a small and clearly defined geographical area. A LAN may contrary to a WAN operate without routers.. 15

**Long-Term Evolution** is a standard for high-speed wireless communication for mobile devices.. 2

**Low Powered Wide Area Network** is a WAN designed to allow long range communications at a low bit rate. It was primarily developed for IoT devices.. 13

**Medium Access Control** is responsible for mapping between logical channels and transport channels, multiplexing of SDUs from logical channels onto transport blocks to be written to the physical layer.. 12

**NB-IoT** is a Low Power Wide Area Network (LPWAN) communication standard for IoT devices that operates on cellular networks.. iii, 13, 14, 35, 36, 54, 57

**Non Access Stratum** is a functional layer between the core network and user equipment in the LTE-protocol stack.This layer is used to manage the initializing of communication sessions and maintaining continuous communications with the user equipment as it changes radio provider.. 10

**Open Systems Interconnection model** is a conceptual model that describes communication functions. The model partitions communications systems into abstraction layers where the goal is that each layers protocol can be replaced without affecting the underlying layers. Each layer are responsible for verifying incoming data and deliver the payload to the abstraction layer above. The abstraction layers can be viewed in 2.2. The OSI-Layers are labeled 1 to 7. Layer 1 is the lowest layer(physical) in the model and layer 7 is the highest layer(application) in the model. . 5

**Packet Data Convergence Protocol** is a LTE radio stack protocol, the PDCP layer is located above the RLC layer and below the IP layer (in the user plane) or the RRC layer (in the control plane). It is mainly tasked with integrity protection and chippering.. 12

**Protocol Data Unit** is a single unit of information transmitted over communications network.. 8

**Radio Link Control Protocol** The Radio Link Control protocol connects the MAC layer with the RRC layer on the control plane and the PDCP layer on the data plane. It offers transparent, unacknowledged and acknowledged data transfer and supports segmentation and reassembly.. 12

**Radio Resource Control** is responsible for connection establishment and release functions, broadcast of system information, radio bearer establishment, reconfiguration and release, etc. in the LTE radio stack.. 10

**Real-Time Transport Protocol** RTP is designed for end-to-end, real-time, transfer of streaming media such as video or audio.. 6

**Robust Header Compression** is a Internet protocol Header compression scheme.. i

**Second Order Compression** is one of three compressor states in RoHC. It is the highest compression state and sends the minimum amount of information to the decompressor to achieve the highest compression rate.. 17

**Service Data Unit** is a unit of data that has been passed down from an higher layer or sublayer to a lower layer in the OSI-Model.. 8

**Transmission Control Protocol** is one of the fundamental protocols of the Internet protocol suite. It provides reliable, ordered, and error-checked delivery of application-to-application data streams.. 5

**Unidirectional Mode** is one of three modes of operation in Robust Header Compression. It may operate without a feedback channel, and changes states according to a set package movement pattern, timeout sequences, and/or feedback from the decompressor.. 17

**User Datagram Protocol** is one of the primary protocol to provide end-to-end communication services for applications.. 5

**Voice over IP** A high-level transmission protocol to send voice data over the Internet. 2

**Wide Area Network** is a computer network that extends over a large geographical distances and operates through the uses of routers. The Internet may be considered a WAN.. 5

# Bibliography

[1] *3GPP Webpage*. visited on 2018-10-30. URL: http://www.3gpp.org/about-3gpp/about-3gpp.

[2] *A Thinwire Protocol for connecting personal computers to the INTERNET, RFC 914 1984*. URL: https://tools.ietf.org/html/rfc914.

[3] Bormann et al. *Robust Header Compression (RoHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed*. July 2001. URL: https://tools.ietf.org/html/rfc3095.

[4] *Cellphones have more capacatity than computers had*. URL: https://www.zmescience.com/research/technology/smartphone-power-compared-to-apollo-432/.

[5] *Code Complete, A practical handbook of Software Construction, Second editon*. ISBN-13 : 978-0735619678 ISBN-10: 0735619670. Microsoft, 2004.

[6] *Compressed TCP RFC 1144*. URL: https://tools.ietf.org/html/rfc1144.

[7] *Connectionspeed in US 2007-2017*. URL: https://www.statista.com/statistics/616210/average-internet-connection-speed-in-the-us/.

[8] Amen Hussain. "Energy Consumption of Wireless IoT Nodes". MA thesis. Norwegian University of Science and Technology, June 2017.

[9] *List over IoT studies*. URL: https://iot.ieee.org/articles-publications.html.

[10] Andreas Maeder and Arne Felber. "Performance Evaluation of ROHC Reliable and Optimistic Mode for Voice over LTE". In: *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*.

[11] Mohammed Al-Obaidi et al. "Hardware Acceleration of the Robust Header Compression (RoHC) Algorithm". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. July 2013.

[12] Craig Partridge, Jim Hughes, and Jonathan Stone. *Performance of Checksums and CRCs over Real Data*. Tech. rep. Bolt Beranek, Newman Inc, Network Systems Corporation, and Standfor University, 1995.

[13] *Protocol Data Unit*. visited on 2018-11-05. URL: `https://en.wikipedia.org/wiki/Protocol_data_unit`.

[14] Rapeepat Ratasuk et al. "NB-IoT system for M2M communication". In: *2016 IEEE Wireless Communications and Networking Conference*.

[15] *Stadnardization of NB-IOT completed*. visited on 2018-10-31. URL: `http://www.3gpp.org/news-events/3gpp-news/1785-nb_iot_complete`.

[16] William Stallings. *Data and Computer Communications*. ISBN-10:1-29-201438-5. Pearson, 2014.