UNIVERSITY OF AGDER

Performance Evaluation of Robust Header Compression Protocol for Low Data Rate Networks

Author: Faraz Iqbal

Supervisor: Frank Y. Li (UIA)

Co-Supervisors: Marrian Hauge & Margrete Allern Brose (FFI)

A thesis submitted to fulfill the requirements for the degree of Master of Science at the University of Agder, Faculty of Engineering and Science

Status: Final

Abstract

Mobile Ad hoc networks (MANETs) have limited capacity due to properties of the physical medium for tactical operations. Several traffic types are typical for tactical applications, i.e. transmit frequent short IP packets (e.g., VoIP and friendly force tracking message). The RTP, TCP/UDP and IP headers comprise a significant overhead for these traffic types. Therefore, robust header compression (ROHC) protocol can useful to save the bandwidth for such applications.

This thesis work is divided into two tasks. First one is the technological background of the robust header compression (ROHC) protocol. Furthermore, a brief introduction and comparison of the related research work which has been performed for header compression in MANETs.

Second task is a real-life testbed scenario where a hybrid wired channel is emulated with loss or delay which is caused by an imperfect wireless channel for a single hop case. Traffic source (compressor node) and bridge node emulation approaches are proposed for channel emulation. Additionally, Netem (network emulator) is used as emulation tool for these approaches.

ROHC protocol performance is evaluated over bridge node emulation testbed scenario. Vyatta Linux routers are already integrated with ROHC library and perform the compression and de-compression functions respectively. The performance of the ROHC protocol is assessed in terms of robustness against transmission errors.

Finally, we concluded that robust header compression (ROHC) protocol is robust up to 40% channel loss for independent packet loss pattern. On the other hand, burst/consecutive packet pattern interrupts the ROHC operation when channel loss is equal to or more than 20%. Additionally, it is observed that ROHC protocol fails to de-compress the header at de-compressor end when consecutive packet loss duration is equal to or more than 30 seconds.

Consequently, ROHC is very robust therefore it can useful for MANETS header compression where the delay ranges from milliseconds to seconds. Furthermore, it is also effective for satellite communication which has longer channel delay and RTT.

Keywords: Bridge, Compressor, De-compressor, Linux, MANETs, MGEN, Netem, Wireshark.

Preface

This report is the result of the Master`s thesis (IKT-590) to fulfil the requirements of MSc in ICT, at the Faculty of Engineering and Science University of Agder (UiA) in Grimstad, Norway.The course contains 30 credits in the degree of MSc ICT. The thesis work has been started from January 07, 2013 and ended on June 03, 2013.

The main objective of this report is to analyze the performance of robust header compression for low data rate networks in a single hop scenario. Additionally, a real scenario in the form of testbed is implemented.

This thesis topic is initiated by FFI. I would like to thank my supervisors Mariann Hauge & Margrete Allern Brose (co- supervisors, FFI) and Frank Y. Li (Internal supervisor, UIA) for their valuable assistance in giving ideas to select the thesis topic. In their supervision, I have learnt a lot about project content and technical report writing.

Faraz Iqbal
University Of Agder
Grimstad, Norway

Table of Contents

List of Figures

List of Tables

Abbreviations

ACK	Acknowledgement
APP	ROHC Application
AODV	Ad hoc on demand distance vector
BER	Bit error rate
CID	Context identifier
CO	Compressed header
CRC	Cyclic redundancy check
DSR	Dynamic source routing
ESP	Encapsulating security payload
FC	Full context state (de-compressor)
FO	First order state (compressor)
HTB	Hierarchical Token Bucket
IP	Internet Protocol
IPHC	IP header compression
IR	Initialization and refresh state (compressor)
LIB	Library
LSB	Least Significant Bits
MGEN	Multi generator
PCAP	Packet capture
NACK	Negative acknowledgement
NETEM	Network emulator
NC	No context state (de-compressor)
O-mode	Bidirectional optimistic mode
R-mode	Bidirectional reliable mode
ROHC	Robust Header Compression Protocol
RC	Repair context (de-compressor)
RTP	Real time protocol
RTT	Round-trip delay time
TC	Traffic Control
TCP	Transmission control protocol
TOS	Type of Service
TTL	Time to live
SN	Sequence number
SO	Second Order state (compressor)
UDP	User datagram protocol
U-mode	Unidirectional mode

Chapter 1 Introduction

1.1 Background and Motivation

Mobile Ad hoc networks are becoming popular due to certain characteristics e.g. rapid development, dynamic multihop network topology. These characteristics enable the deployment of MANETS for bandwidth limited communications, which have data rates ranging from several kbps to hundreds kbps. There are certain traffic patterns that are most demanded for military communication services such as short IP packets (voice over IP & friendly force tracking). These patterns contain traffic types of RTP, TCP/UDP, and IP headers.

A military IP network can be based on fixed wired infrastructure, satellites and networking equipment in operations and command centers. But it is not practically feasible to develop a static wired network on a battlefield. Therefore, mobile wireless network is the only realistic way for networking infrastructure deployment [1]. Wireless network is capable for real-time voice, data and streaming video communication. However, the capacity of wireless networks is generally limited.

The problem of capacity limitation can be aborted with header compression. IP header compression is the process of compressing protocol headers before transmission and uncompressing them to their original state on reception. This type of compression is achieved by considering the redundancy in header fields of the same packet as well as consecutive packets of the same packet stream. There are popular header compression techniques used in wired and wireless IP platforms such as Van Jacobson`s header compression, RTP Header compression and unified header compression. But these are not suitable for wireless networks [2]. Robust header compression has better performance for wireless network i.e. LTE, WiMAX and GSM radio communication. ROHC is already implemented for one hop scenario in 3GPP standard [2].

However, the performance of robust header compression (ROHC) protocol is not tested in a Linux based device [3] (see operation systems in reference). Therefore, it is important to evaluate the performance of header compression protocol in a realistic way (testbed scenario). This motivates us for the performance evaluation of ROHC protocol in a Linux (Debian distribution) based Vyatta router.

1.2 Problem Statement

We are interested in performance evaluation of the robust header compression (ROHC) protocol when it is being used with UDP/IPv4 traffic over MANETs. This protocol is being used in cellular networks and works well for 1-hop wireless connections; however no previous work exists on how this protocol performs in Ad hoc networks for one hop, especially based on a real-life testbed. The performance of the ROHC protocol is assessed in terms of robustness against transmission errors. The problem statements are the followings.

- How the ROHC protocol performs with the imperfect channel conditions?
- What are the conditions where ROHC device (de-compressor) fails to decompress the compressed header (de-compression failure)?

- Does the ROHC protocol recover from a failure state?

The other task is to build a real-life testbed to evaluate the performance of ROHC with transport protocols (UDP) and packet formats (IPv4) in a single hop scenario.

1.3 Approaches

Ad hoc networks are based on wireless channel communication link. A wireless channel is prone to packet loss & delay. We will emulate the hybrid wired channel with packet loss and delay conditions which are caused by imperfect wireless channel and signal propagation.

Network emulator (Netem) is proposed for the wired channel emulation. This emulation technique can modify the characteristics of the packet stream with respect to configured parameters. It is being a practical lab scenario. We have Linux based routers which are already implemented with ROHC protocol library. Furthermore, the routers are directly connected through Ethernet cable.

In our testbed scenarios, network emulation is performed with two different approaches for the hybrid wired channel which is followed here.

1. Source (Compressor) node emulation: The generated packets are first emulated with Netem queue for packet loss or delay. Then, this emulated stream is forwarded to ROHC library for compression. Afterwards, the emulated compressed traffic is transmitted to receiver (de-compressor) end. Hence, a packet emulation of the source (compressor) node reflects an imperfect real-life channel condition for the receiver (de-compressor) node. In this way, we evaluate the ROHC protocol performance at the de-compressor end.

2. Bridge node emulation: Vyatta Linux router is configured as bridge node which is used as real-life channel between source and receiver nodes. This bridge node is configured with Netem queue and adds the packet loss or delay in the compressed packet stream. The source (compressor) node sends compressed packet stream towards bridge node which emulates the traffic and then forwards it towards de-compressor node.

The protocol robustness is evaluated with respect to packet loss percentage over the channel. In such case, the packets are captured at compressor and de-compressor ends with a packet sniffer (Wireshark).

First, we count the transmitted and received packets at compressor & de-compressor ends respectively. It tells us the overall packet loss at receiver ends based on channel emulation.

Secondly, we examine the packet loss percentage for which the de-compression failure events happened. Furthermore, we extract the reasons of de-compressor failures. The reason might be consecutive packet loss or IR header packet loss. Moreover, the compressed packets are discarded which are received during de-compression failure duration at the de-compressor end and will not de-compress.

The packet losses are counted by packet sniff Wireshark log file at the de-compressor node Ethernet (eth0) interface. The trace analysis of the Wireshark captured log file displays the de-compression failure events during the real test time duration. Additionally, the Vyatta Linux router also provides a log file for packet loss over a real test case. The packet losses time period are identical in both log files (Wireshark & Vyatta router).

Finally, we count the discarded packets due to de-compression failures for test duration at the de-compressor end. Actually, these packets are lost because ROHC protocol cannot decompress it due to channel loss. Furthermore, the number of packet lost due to de-compression failures expresses the ROHC robustness level with respect to different channel loss percentage. It also describes the ROHC protocol robustness level with different channel emulation conditions such as forward channel emulation or forward & reverse channel emulation.

1.4 Report Outline

The rest of this report is organized as follows:

Chapter 2 describes an introduction of three header compression schemes and a research work of header compression in MANETs.

In Chapter 3 we will describe the robust header compression (ROHC) protocol terminology, framework, and profiles with reference of RFC 3095, RFC 5225 and RFC 5795.

Chapter 4 is a brief introduction of ROHC protocol integration in Vyatta Linux based router.

Chapter 5 defines the testbed scenarios and configurations for robust header compression (ROHC) protocol performance evaluation.

Chapter 6 describes the experimental results based on the testbed configuration. Furthermore, results are shown with trace analysis, graphs and tables.

Chapter 7 is the discussions of various issues which are faced during thesis work.

Finally, Chapter 8 concludes the thesis with Future work followed by references.

Chapter 2 Technology Background

The chapter is divided into two sections. First of all, Section 2.1 describes the header compression techniques for an IP infrastructure. Secondly, a brief description of related research work which is performed for the improvement and enhancement of header compression technique.

2.1 Header Compression Techniques

Header compression is a process which compresses the protocol header of data packet before data transmission and de-compresses them to their original state at the reception end. There is a brief introduction of the three header compression schemes which are used for IP based architecture [4]. The compression schemes are followed.

2.1.1 Van Jacobson Header Compression (VJHC) [5]

This scheme classifies the packets into separate flows with respect to certain parameters such as packets with the same set of IP addresses, transport protocol type & TCP port number. Each flow is identified with a context Identity (CID) at the compressor and de-compressor ends.

First, a compressor sends the uncompressed (complete field information) header to the receiver. Afterwards, the compressor neglects the unchanged fields and fields that do not change frequently in the packet header. A further saving of bandwidth is achieved by transmitting the difference (i.e. differential coding) in the value of the field rather than the entire fields.

Error recovery is achieved with TCP/IP mechanism because this technique does not support for separate feedback.

Advantages: Improved performance over slow speed serial links. It can compress the TCP/IPv4 header from 40 Bytes to 4 bytes.

Disadvantages: Retransmission of packet in case of packet loss or corrupt that is costly for low speed connections and links which are high prone to errors like wireless networks. Additionally, VJHC does not support UDP and IPv6.Therefore, it is not considered as good option for wireless channels.

2.1.2 IP Header Compression (IPHC) [6]

This scheme is similar to VJHC such as it ignores the unchanged fields in the packet header. The main difference between two algorithms is that IPHC only compresses the IP header. Furthermore, this technique can be used with any transport protocol or tunnel encapsulation and IPv6.

Advantage: IPHC can reduce the IP header up to 2 bytes for non TCP session and 4 bytes for TCP session. Moreover, it can support any transport protocol or tunnel encapsulation.

2.1.3 Robust Header Compression (ROHC) [7]

ROHC is comparably better than VJHC and IPHC due to high robustness and improved efficiency. The schemes is high suitable for high BER (Bit error rate) wireless environment due to its high robustness and reliability.

Advantage: ROHC scheme can compress such type of protocol header i.e. RTP/TCP/UDP/UDP-Lite. It is more robust than VJHC and IPHC.

In this thesis work we focused on ROHC and evaluate the performance in terms of robustness over imperfect channel conditions.

In the next section we will describe the related research work for the header compression in the MANETs.

2.2 Related Work of Header Compression in MANET

This section summarizes the research work that is already performed for header compression improvement in MANETs. This section is based on study relative to research papers.

2.2.1 SEEHOC (Scalable & Robust End-to-End Header Compression Techniques for Wireless Ad hoc Networks) [8]

This research paper highlights the challenges for end-to-end (multihop) header compression in wireless Ad hoc networks. Furthermore, it proposed an end-to-end header compression algorithm for multimedia flows over MANETs. This could be a cost effective technique to raise the utilization and efficiency for the end-to-end wireless Ad hoc networks. It proposed an RTP/UDP/IP compressed header format to measure the performance of SEEHOC Algorithm. The performance evaluations are based on OPNET simulations.

Approach for End-to-End Compression Challenges

There are certain challenges that are addressed in this research. A brief overview is followed here.

Packet Format Identity: It uses a different identity number which specifies the format of its header. It assigns a four bits number for header compression packets identification other than the reserved or assigned network protocols (0, 4, 5, 6, 7, 8, 9 & 15), such as 11.

Context State Management Mechanism: Header compression protocols can follow two techniques for context maintenance at the de-compressor side which are following.

1. **Explicit retransmission request**: Compressor updates the context states when it detects packet losses or in case of context recovery request from de-compressor.

2. **Soft-state**: Compressor updates the context with full/partial header packets for de-compressor end periodically. If context is not updated after a defined time period then the context is deleted at de-compressor end.

Both of these techniques have their own limitations and benefits. SEEHOC suggests a hybrid context state management mechanism which has benefits of both techniques.

Hybrid Context State Management Mechanism: The context is updated periodically with a large time interval between two consecutive updates. A context state timer is refreshed by context state update packets.

SEEHOC Algorithm

It defines the operating mechanism for the source, intermediate and destination nodes.

Source (Compressor) Node: A generation number is added in compressed headers and IR headers to overcome the mismatch (occur between a packet flow and its context state at a node). In the case of packets rejection, the node establishes the flow context state periodically. The node deletes the context state flows when the time out period is elapsed.

Intermediate Node: It routes the compressed packets to their destination. These nodes do not perform any compression or decompression. If the CID of the new flow is already used by other flow then it either used CID swapping or may be does not establish the context state.

If the CID is unique or not shared among flows then decrements the TTL field. Furthermore, if the TTL field is greater than zero then it forwards the packet towards next hop or destination; otherwise drop the packet.

Destination (De-compressor) Node: It establishes the context state for flow with new CID, use the CID swapping mechanism or reject the context state request for same CID numbers. It performs the MAC address verification, validity of generation number, updates the flows context state for the compressed packets. It deletes the context states after the time out period of the context state packets, compressed packets or update packets.

2.2.2 ARHC (A Robust Header Compression Method for Ad hoc Network) [9]

It proposed a header compression algorithm that can be deployed in hop-by-hop or end-to-end mode.

ARHC approach addressees the de-synchronization (link connection failure between compressor & de-compressor nodes) issues which lead to poor network performance. The context asynchrony (link failure) is much frequent due to the poor wireless channel quality. When it appears, the later successive packets will decompress corruptly according to old context. Consequently, periodically transmission of full header packet is required to update the context at de-compressor end.

The authors argue that there is no need to preserve the header context while compressing or decompressing with ARHC integration in MANETs. If a packet is lost or corrupted, it will not affect the subsequent packets due to independence of successive compressed headers.

ARHC Approach

The protocol can compress UDP/IP, TCP/IP and RAW IP header. It can compress headers without reference the relativity between packets if only the first packet has successfully delivered.

Header Field Classification: The protocol classified the header field for TCP, IP and UDP in to following classes i.e. constant, meaningless and critical.

General Compression Method: The compressor generates a unique context ID and records it. The context id comprises of four specific indexes such are source & destination addresses, source & destination ports, protocol and all constant fields. Compressor sends the full header packets with the context ID. At the arrival of this information, de-compressor stores these indexes in its memory (context table).

For the next outgoing packet transmission, the compressor removes the constant fields and four indexes from header and only sends context ID with critical field's value. De-compressor retrieves the four indexes and constant field's values from the context table which is index by context ID. Furthermore, it computes the meaning less header fields from the preset function and gains the critical fields from currently received compressed header. Consequently, the de-compressor can store the original field's value by this information.

2.2.3 Hybrid Header Compression for Ad hoc Networks [10]

The authors proposed a hybrid hop-by-hop and end-to-end header compression framework. The framework addresses the context initialization and control-message overhead issues which are associated with node mobility in MANETs.

In this approach, context initialization is conducted through routing information field which minimizes the overheads of the frequent context initialization and routing messages during the period of high node mobility.

There are two approaches in hybrid header compression schemes. First approach permits the end to end compression at transport-level and application-level headers. Second approach allows the hop-by-hop compression at network level. Both of these are briefly introduced as follow.

A. End-to-End header compression

In this case, there is no need for periodic updates. Moreover, the source and destination nodes initialize an end-to-end context to compress and de-compress the end-to-end headers. The end-to-end headers do no need compression at intermediate nodes therefore; intermediate nodes just forward the packets on discovered routes.

B. Hop-by-Hop header compression

An intermediate node can only compress the network level headers. In this case, two adjacent nodes develop a synchronized hop-by-hop context to compress and de-compress the network headers. Network level headers must exist at each intermediate node in uncompressed format for packet forwarding information.

Hop-By-Hop Context Initialization

The proposed context initialization algorithm caches the address information at the intermediate nodes which is transmitted in **AODV** protocol 'route request' and 'route reply' messages. Therefore, address fields will not send during context initialization of IP header compression. Consequently, it minimizes the overheads by caching address information.

Each route request is labeled with a tuple (s, d, t) where s and d are the source and destination nodes and t is the time when the route request is acknowledged. Furthermore, the unique time stamp is used as **address label** ℓ. The size of the label is specified as 7 bit.

The routing requests do not require transmitting the entire path information between source and destination nodes because this function is possible through the intermediate nodes. In this case, intermediate node selects the route from its stored routing table for desired destination end.

2.2.4 A Joint Approach for IP Overhead Reduction in Wireless Ad Hoc Networks [11]

The authors proposed a joint approach where compression is added with the Ad hoc routing protocol for high compression gain and bandwidth efficiency.

Proposed Approach Characteristics

1. There is no compression/de-compression performs at intermediate node. The end- to- end nodes do the compression/de-compression whereas the intermediate nodes follows the packet forwarding.
2. Feedback is not used by the destination node for context maintenance. The context updates are applied periodically.
3. A label switching technique is used which tag each stream with a flow identifier (CID) and use cache information at the intermediate node for next hop route selection.
4. Dynamic source routing (DSR) is integrated with Label switching for route selection and packet forwarding.

Integration of Label Switching with Ad hoc Routing Protocol

Route Identification: Assign a flow identifier or label for each specified data stream and then integrates with the Ad hoc routing protocol such as Dynamic source routing (DSR). Every intermediate node maintains a flow table which holds the packet flow IDs (CID) and MAC addresses for upstream & downstream nodes. In this way an intermediate node selects the next hop for a packet.

Flow Identifier (CID) Uniqueness: Label switching experiences a main problem in Ad hoc networks that is how to assign a unique flow identity for a data stream. It is achieved with the MAC addresses of the upstream and downstream. If two data stream exist with same identifier at intermediate node then, it is differentiated by MAC addresses of source and destination.

In case, two different flows ID are assigned to the same destination node. In this situation a conflict occurs at the intermediate node. The paper proposed two solutions to resolve this problem. First, the CID denies message will be send by the intermediate node to the source requesting node therefore it will select another CID. If it is not possible by the source then the intermediate node reject the establishment of new context. Second way is CID swapping by the intermediate node.

2.2.5 Comparison

The section highlights the characteristics of header compression schemes in Table 1 which are already explained in the previous sections. We have compared the compression schemes working modes & approaches, compression formats, merits & demerits. Furthermore, it also extracts the similarities and discriminates among header compression schemes.

In Table 1, first row comprises the schemes name. Similarly, first column consists of certain features.

	SEEHOC	**ARHC**	**Hybrid HC**	**Label switching with DSR**
Working Mode	End-to-end header compression	Point-to-point/ end-to-end header compression	Point-to-point/ end-to-end header compression	End-to-end header compression
Header Compression Formats	RTP/UDP/IP	UDP/IP, TCP/IP, RAW IP	RTP/UDP/IP	RTP/UDP/IP
Algorithm	1. Compressor, de-compressor & intermediate nodes. 2. Assign a 4 bit identity number for compressed packet other than reserved protocol numbers. 3. CID Swapping for identical CID numbers.	1. Compressor and de-compressor nodes only. 2. Header field classification of constant and critical fields for header compression. 3. Compress headers without reference of the previous packets but the first packet should be successfully delivered.	1. Hop-by-hop compression is performed by network (IP) headers for packet forwarding by the intermediate nodes. 2. Ad hoc routing protocol (**AODV**), Route request is labeled with source, destination and time (route request ACK). 3. End-to-End compression is based on transport-level (UDP) & application-level (RTP) headers.	1. Compression / decompression applied at end nodes only. 2. Intermediate nodes store the hops information in cache to perform packet forwarding based on Label switching. 3. Joint optimization of the compression and Ad hoc routing (**DSR)** protocol. 4. Delta encoding & Cooperative Header Compression (COHC) schemes.
Merits	Improvement of 35% throughput for compressed packets.	Overcome the dependency of successive packets relativity, avoid asynchronous problems.	Packet loss reduces for compressed packet as compared with uncompressed one.	1. Transport overhead reduction up to 70%. 2. Average compression gain for delta coding

				and COHC is about 75% & 62% respectively.
Drawbacks	Size of compressed packet increases due to incremental encoding	Do not describe the routing at intermediate nodes.	a. Cache memory is required for label storage at intermediate nodes. b. Time delay for route selection due to RREQ (route request) & RREP (route reply) events.	a. DSR scalability problem. b. Unique assignment of flow identifiers. c. Node mobility affects the number of route breakage for delta encoding scheme which case packet loss.
Performance Evaluation	OPNET Simulations and SEEHOC rapid prototype.	Simulation based on glomosim.	NS-2	NS-2 (Network Simulator)

Table 1: Header Compression Schemes Comparison

Consequently, we have summarized the related work which addresses the improvements in header compression protocols. We found that all schemes support for end-to-end header compression working mode. Moreover, all schemes support RTP/UDP/IP compression format except 'ARHC' which supports only UDP/IP & TCP/IP.

Each scheme is identified with an individual approach. First, SEEHOC algorithm resolves the identical CID number issue with CID swapping at intermediate node. Therefore, an individual CID number maintains the unique identity of every single packet stream. Second one is Hybrid header compression scheme where context initialization is performed through Ad hoc routing protocol (**AODV**). It minimizes the overheads of the frequent context initialization. On the other hand, we have label switching header compression scheme with a joint optimization of Ad hoc routing protocol (**DSR**). Moreover, it reduces the transport overhead up to 70%.

All of the related research work leads to an advancement in header compression protocols which is already explained in the current section. But most of the schemes performance evaluation is based on simulations as depict in Table 1.

Hence, it is a realistic approach to evaluate the performance of header compression protocol in a practical way (testbed scenario). This motivates us for the performance evaluation of ROHC protocol in a real-life testbed case.

Chapter 3 Robust Header Compression Protocol

Robust Header compression (ROHC) is a standard method defined by the IETF (Internet Engineering Task Force). It is used for compression of certain headers in IP packets i.e. UDP/IP, RTPUDP/IP, ESP/IP, UDP-Lite/IP and TCP/IP. Figure 3.1 represents an RTP/UDP/IP compressed header packet where the header size is compressed up to 1 byte.

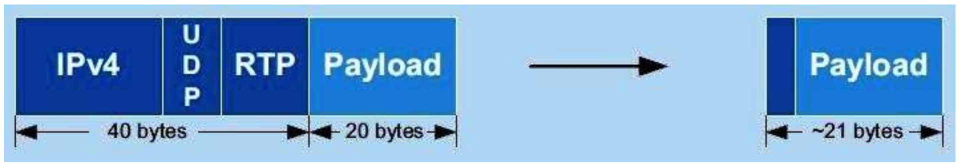


Figure 3.1: ROHC packet format [9]

Figure 3.2 describes the basic idea of compression/decompression between directly connected nodes. The ROHC scheme is used at layer 2 in a network stack where it permits that lower layers should use some level of error detection/protection mechanism.

In the beginning of header compression, Compressor sends few packets without compression and builds the context on both sides of the link. These packets are possibly repeated periodically until de-compressor contains full header. This context is actually information about static fields and dynamic field and relative vary pattern in the headers. The context is used for efficient compression and decompression (original state) of the packets.

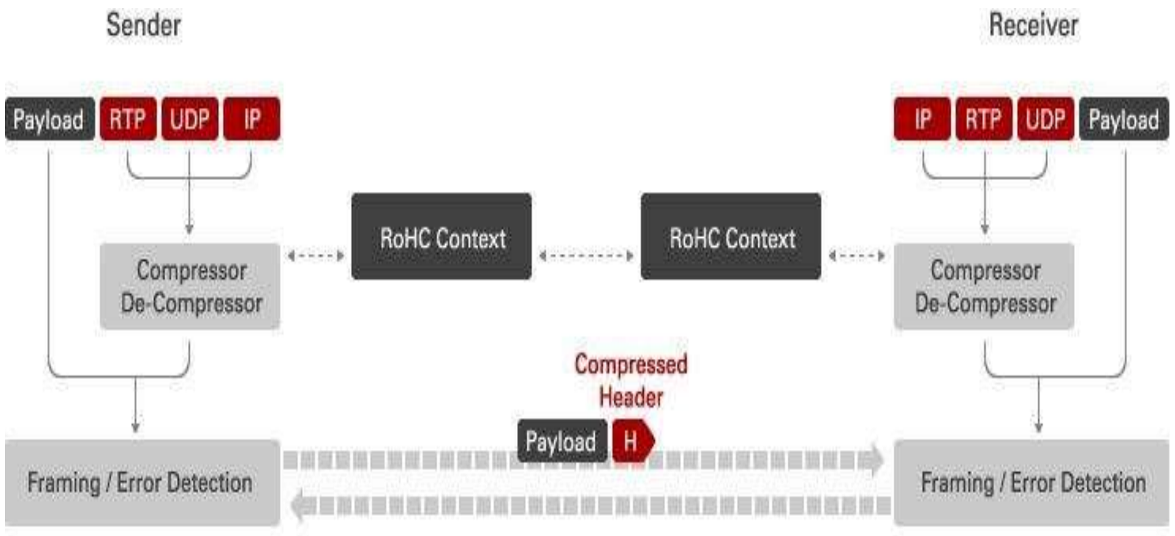


Figure 3.2: Application of ROHC in a protocol stack [10]

The compressor sends compressed header packets when it gains the de-compressor confidence. It means that the de-compressor has obtained full header frames and ready to receive the compressed frames.

A ROHC compressor has three states. Normally, it starts from Initialization and Refresh (IR) state and sends full header. In First-Order (FO) state, it stores the static fields (IP addresses & port numbers) on both sides of the connection. Furthermore, it also sends some dynamic fields. In Second-Order (SO) state, the compressor sends the dynamic fields by using a logical sequence number and CRC.

Similarly, de-compressor node has three states. In NC (No Context) state, the de-compressor has no context information. In FC (Full Context) state, de-compressor gains the all static and dynamic fields information and decompresses the full compressed headers. This is the highest state at de-compressor end. There is one middle state which is called RC (Repair Context). This state provides the static header fields to FC state in case of repeated failures.

Compressor and de-compressor states are further explained in Section 3.3.4.

The ROHC protocol can works in three specific modes .i.e., U-mode, O-mode and R-mode. In u-mode, the compressor sends periodic updates towards de-compressor. Furthermore, this mode has lower compression gain due to periodic updates. O-mode has higher compression gain than u-mode. In this mode, compressor does not send periodic update. This mode comprises a feedback mechanism for error recovery and IR header acknowledgement. R-mode is most reliable mode which uses ACK for each packet delivery at the de-compressor end. Furthermore, compressor sends periodic updates until ACK is not received from de-compressor end. ROHC modes of operations are further described in details in Section 3.3.4.

General Principles

Robust header compression is based on the following general principles [7].

1. Identification and grouping of the packets based on certain parameters such as IP addresses, port number and protocol type. Furthermore, additional application identifiers such as the synchronization source (SSRC) in RTP.
2. Compressor and de-compressor maintain a context for each packet flow. Every compressed header is identified with a context identifier (CID) for the labeling of packets.
3. ROHC should be familiar with the changing patterns in the header fields such as time stamp and sequence number in RTP header.
4. Use encoding techniques which encodes the most frequently changing pattern fields of a compressed header. It is used in packet formats and common to all profiles.

IETF ROHC WG (working group) was formed to develop new header compression protocols [12]. It has specified a common compression platform which sets the ROHC profiles and framework. In our research work we have focused on three RFCs. These are followed here.

- RFC 3095 – "ROHC Framework and four profiles: RTP, UDP, ESP and uncompressed" [7]
- RFC 5795 – "The Robust Header compression (ROHC) framework" [13]
- RFC 5225 – "Robust Header Compression Version 2 (ROHCv2: Profiles for RTP, UDP, IP, ESP & UDP-Lite)" [14]

RFC 5225 is the second version of basic ROHC RFC 3095. It seems to be best to use the second version of basic RFC and also improved support to handle pre-link reordering more efficiently. Specially, a clear distinction between framework and profiles is not prominent in RFC 3095 [14].RFC 5225 describes the improvements in basic profiles (RFC 3095) but it does not obsolete it. It is based on ROHC framework in accordance with RFC '5795'.

Note: ROHC software (ROHC-LIBRARY) implementation is based on the basic RFC 3095 due to software availability. The details are further included in chapter 5.

3.1 ROHC Terminology [7]

There are various terminologies which are being used in ROHC framework and profiles. It is better to know little bit about these terms which are followed here.

3.1.1 Context

Context is basically a state which is used to compress and decompress a header respectively. It contains the headers relevant information such as static fields, dynamic field, changing behavior of fields (IP-ID or timestamp, sequence number) or possible reference values for compression and decompression.

Context information is kept in the context table which is stored at both compression and decompression ends. The information is indexed with a label CID (context ID), sent along with compressed header and feedback acknowledgements.

3.1.2 Profile

ROHC profile defines the elements that build the compression protocol. It is expressed with an identifier which is a set of non-negative integers. The profile elements are followed here.

Packet Formats

It is related to the certain properties which are followed below [7].

Bits on the channel: It defines the bits for profile specific packet types. It is common for all profiles (e.g., IR and IR-DYN).

Field encoding: This indicates the outcome of an encoding method which is specific for a compressed field of a specific packet format.

Updating properties: It is used to update the de-compressor states. It may be different for specific profiles. It defines the way of de-compressor context update through fields or different packets types in a profile.

Verification: A mechanism which verify the update packets integrity at the de-compressor end.

Context Management

It is described with two mechanisms which are followings [7].

Robustness logic: Minimize the effect of packet lost or reordering events and prevent damage propagation.

Repair mechanism: Overcome the decompression failure and context damage at de-compressor side. It works with feedback logic as an optional.

3.1.3 ROHC Channel

It is defined as a logical unidirectional point to point channel carrying ROHC packets from one compressor to one de-compressor.

3.1.4 CRC Verification & Validation

CRCs are used to detect decompression failures. There are different types of CRCs used for validation and verification of headers at the de-compressor side such are 8-bit CRC, 3-bit CRC or 8-bit CRC [14].

3.2 ROHC Framework [7] [13]

It specifies the requirements and functionality of the ROHC channel. It contains the formats and structures that are common for all ROHC profiles. Furthermore, it describes the multiple compressed packet flow over the common channel.

It comprises of packet formats and encoding methods that are same for all profiles. The general ROHC packet format is define in framework standard RFC 5795. The ROHC profiles and certain header formats are improved in ROHCv2 (RFC 5225) which was already defined in RFC 3095 [13].

3.2.1 ROHC Packet General Format

It consists of four generic fields. Each of the field has variable length [13]. The fields are followed as.

1. Padding: It is assigned at the starting of the compressed header; this aligns headers to byte boundaries.

2. Header: It consists of compressed header fields. The header either start with a packet type indication or has a packet type indication followed by a CID octet. Furthermore, a body field is interpreted according to the packet type indication and CID information, as defined by individual profiles. The general header format is shown in Figure 3.3.

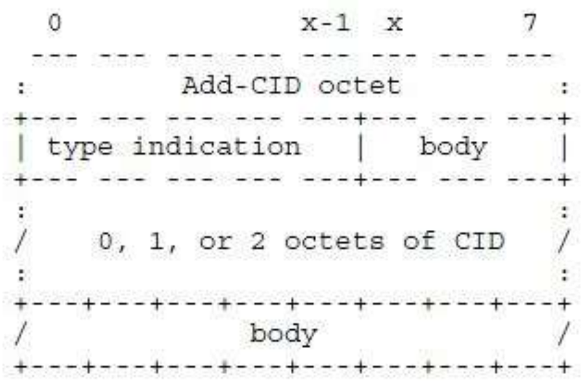


Figure 3.3 : Compressed header format [12]

3. Feedback: A variable length field used for context repair acknowledgement and identify the successful decompression attempts.

4. Payload: It varies with respect to different application data. It is the real data (voice or text).

3.2.2 Initialization and Refresh Packet (IR)

These packets contain the profile identifier which determines the header interpretation syntax. It associates a context identity (CID) with profile, which is stored at the both compression & decompression ends respectively.

This packet always updates the context for all context updating fields which are located in the header. It communicates the static and dynamic blocks of the context. An old context is cleared when compressor receives a request for a new context.

A ROHCv2 IR header has the following format i.e. CID, Profile field, CRC, static chain and dynamic chain.

CRC: This is 8-bit CRC over the entire IR header for error detection and validation.

Static Chain: This chain is only used for IR headers. It consists of one item for each header of the chain of the protocol headers (compressed), starting from the fields in the outermost header.

Dynamic Chain: This is same as static chain but it contains dynamic fields for sub headers. It is used for IR and repair header formats.

Note: Static and dynamic chains send together such as in IR header formats.

3.2.3 Compressed Header Formats [13]

Compressed header formats are divided in to two parts.

A. Common Compressed (CO) Header Format

This header format identifies the variation in dynamic header fields. This format comprises of a set of flags to indicate the field's presence and their changing value accordingly. This format can also update the control fields. It is protected by 7-bit CRC code.

Furthermore, communicate the irregularities in the packet header and carry the base header information (variable length).

B. Repair Compressed (CO-repair) Header Format

This format is used to identify the context damage state. It updates the context of all dynamic fields by conveying their uncompressed value.

The header format is based on fixed length with encoded CRC bits for validation and dynamic chain of variable length.

3.2.4 Least Significant Bit (LSB) Encoding [7]

ROHC protocol uses the LSB encoding for the header field (dynamic) which is usually subject to small changes. The 'k' least significant bits of the field value are updated at compressor end and further transmitted instead of original field value. The value of ‘k’ is a positive integer. After wards, the de-compressor receives the ‘k’ bits and develops the original field value with the help of previously received reference value (v-ref) [7].

The scheme is guaranteed to be correct if the compressor and the de-compressor each use interpretation intervals.

1. *In which the original value resides, and*
2. *In which the original value is the only value that has the exact same k least significant bits as those transmitted.*

The interpretation interval can be described as a function f (v_ref, k). Let

f (v_ref, k) = [v_ref – p, v_ref + (2^k - 1) - p]

Where p is an integer.

```
<----------------  interpretation interval (size is 2^k)  ------------->
|-------------+-----------------------------------------|
v_ref - p          v_ref                                v_ref + (2^k-1) - p
```

The function f has the following property: for any value k, the k least significant bits will uniquely identify a value in f (v_ref, k).

The parameter p is introduced so that the interpretation interval can be shifted with respect to v_ref, choosing a good value for p will yield a more efficient encoding.

The compressor (de-compressor) always uses v_ref_c (v_ref_d), the last value that has been compressed (decompressed), as v_ref.

Compressor: When compressing a value v, the compressor finds the minimum value of k such that v falls into the interval f (v_ref_c, k). Call this function k = g (v_ref_c, v).

De-compressor: When receiving m LSBs, the de-compressor uses the interpretation interval f (v_ref_d, m), called interval_d. It picks as the decompressed value the one in interval_d whose LSBs match the received m bits.
(RFC 3095, Section 4.5.1)

3.3 ROHC Profiles [7] [14]

The profile defines the compression algorithm and profile specific packet formats. RFCs can distinguish with respect to different profiles. Profiles can be organized with different applications. In general, it is composite of applications, transport & network protocols and header formats. In the same manner, ROHCv1 (version 1) & ROHCv2 profiles are arranged with certain layers protocol combinations such are RTP/UDP/IP, RTP/UDP-Lite/IP, UDP/ IP, UDP-Lite/IP, IP and ESP/IP.

Packet formats are already defined in framework section. This section describes the compressor and de-compressor devices.

3.3.1 Compressor Concepts

It is responsible to deliver the information which is desired for successful decompression attempt. This information is specifically based on confidence level between compressor and de-compressor. The confidence can achieve in such ways which are followed.

1. It maintains a reference header in context table when a connection is being established first time. This header saves the static and dynamic field information which is further use for the de-compression failure recovery.
2. Compressor updates the de-compressor context from the optimistic approach.
3. Feedback message received from the de-compressor for successful packet decompression acknowledgement. Moreover, context recovery request from de-compressor end.

Optimistic Approach (OA)

This approach ensures that at least one compressed header which has updated field information, must receive at the de-compressor end. This approach is always used for context updates at compressor side.

1. Compressor repeats the updates until it achieves the confidence level from de-compressor side (information received).
2. When the de-compressor received any context update through optimistic approach, then it transits to higher states. In this way it starts to process the fully compressed headers.

Compressor State Machines

Compressor comprises of three transition states as shown in Figure 3.4.

1. Initialization and refresh (IR)
2. First order (FO)
3. Second order (SO)

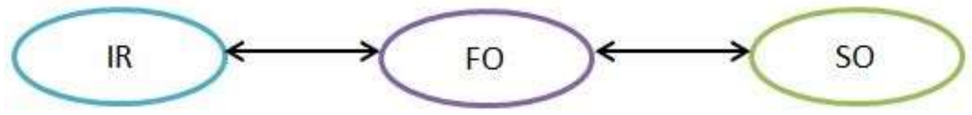


Figure 3.4: Compressor state diagram

The compressor starts from IR state, it sends complete header information (static & non static fields) in uncompressed format to get the confidence level at the de-compressor side. The confidence level is related to the correctness of the context.

The next stage is FO state, deals with the irregularities in the packet stream. In this state, compressor sends dynamic fields in a partially compressed format. It also updates a few static fields. The compressor also enters this state (FO) from SO state when the packet headers do not verify the previous pattern changing fields. When the de-compressor gains the updates of the new pattern then compressor move forward to higher state (SO).

The compressor enters SO state when it is confident that de-compressor has received all the changing field information such as SN (sequence number) or other patterns. However, compressor move back to FO state in case of context de-synchronization or header fields mismatch.

The compressor ensures the validity at the de-compressor by moving to lower states at the compressor end. Moreover, the states transition is reaction of the link and error conditions. The decisions for transitions among compression states are based on the following outcome.

- Variations in packet headers.
- Positive feedback from de-compressor (ACKs -- Acknowledgements).
- Negative feedback from de-compressor (NACKs – Negative ACKs).
- Periodic timeouts (in the absence of feedback).

3.3.2 De-compressor Concepts

De-compressor is responsible for the successful decompression of compressed header fields. The specific responsibilities are followed here.

1. First verified the decompression attempt with CRC which is present in every compressed header formats. At the successful verification (CO packets) or validation (IR packet), update the context table.
2. In case of transmission failures due to context information damage or residual errors, request for context repairs by using feedback.

De-compressor State Machines

De-compressor states hierarchy is identical to compressor. The difference is found in functionality which is described in this section. The three states are followed in the Figure 3.5 [14].

1. No Context (NC)
2. Repair Context (RC)
3. Full Context (FC)

In the beginning state (NC), there is no valid context. When de-compressor receives an IR packet from compressor end, it validates the integrity of the header with CRC-8 validation technique. If IR header is successfully validated, then de-compressor updates the stored context and uses this header as the reference header. In this state, only the static field information can be decompressed.

The validation of IR headers moves the de-compressor to higher state (FC). In this state, decompression can be attempted regardless of the received packet type. In case of damage context/de-compression failure condition, de-compressor moves one step down towards RC state or NC State for static context repair mechanism. If the de-compressor is not able to decompress the further incoming compressed packets or a local repair attempt fails then it sends a feedback messages for downward state transition. The feedback is sent to compressor node. Feedback packet format is described in Section 3.3.3.

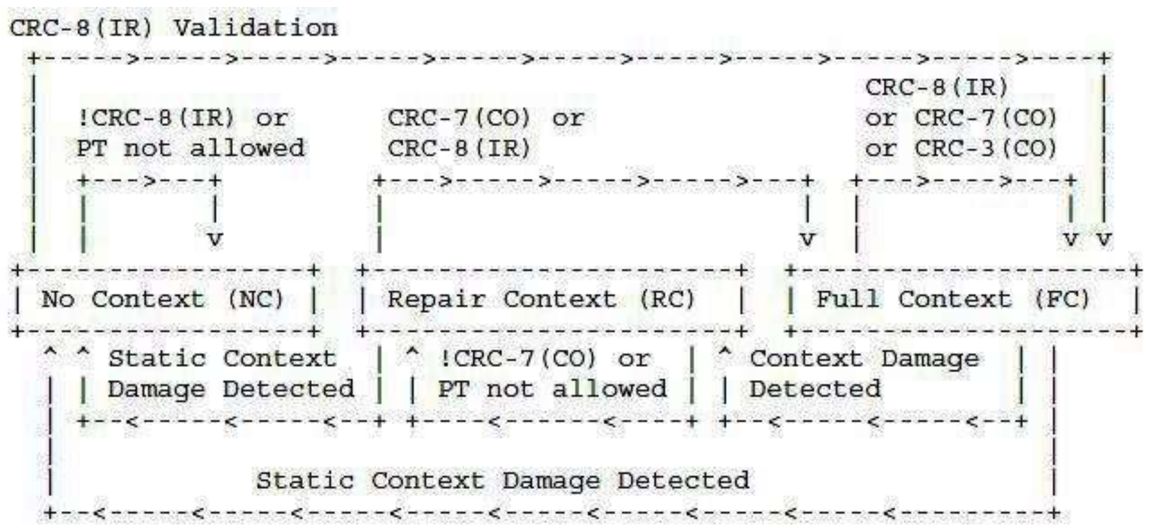


Figure 3.5: De-compressor state diagram [13]

Where:

CRC- 8 (IR): Successful CRC-8 validation for the IR header.
! CRC- 8 (IR): Unsuccessful CRC-8 validation for the IR header.
CRC- 7 (CO): Successful CRC verification for the decompression of a CO header.
! CRC- 7 (CO): Failure of CRC-7 (CO).
Static context damage Detected: Full header damage indication.
Context Damage detected: Dynamic fields damage in full header.
PT not allowed: Packet type (PT) for which the decompression's current context does not provide enough valid state information to decompress the packet.

Context Management [14]

The integrity of the updated context information is verified with CRC which is added in all header formats. A CRC verified packet updates the reference values of all header fields. Context validity does not represent a specific part therefore in case of non-validation of headers, the de-compressor can assume a few or entire context values are invalid [14].

The de-compressor supposes two assumptions of the failure.

1. State transitions which can be a reason of damage of the dynamic part (fields) of the context.
2. Repeated decompression failures and unsuccessful repairs, it is assumed that entire context is damaged. This includes the static part of the context. In this condition, entire context (static & dynamic fields) needs to be update.

Context Damage: In this condition, the de-compressor cannot decompress CO (compressed) header. It can only decompress IR headers protected by a CRC-7.

Static Context Damage: The de-compressor can only decompress the IR headers.

3.3.3 Feedback Logic [14]

The ROHC comprises of three types of feedback messages.

1. ACKs: Acknowledges successful decompression of a packet. Indicates the successful validity of context at de-compressor end.
2. NACKs (Negative ACK): Represent the invalid dynamic part of the context at the de-compressor side.
3. STATIC-NACKs: Indicate the entire static context is damaged at the decompression attempt.

The feedback messages convey the context management information for context repair or failure indications. The de-compressor sends NACK when it considers that context is damage. It sends a STATIC-NACK to indicate a static damage context state. Furthermore, the de-compressor is not expected to send ACK for each successful decompression attempt. Acknowledgement can be used for improving the compression efficiency at compressor end.

There are two formats which are used for feedback acknowledgement in ROHC. First one is Feedback-1 which is only an ACK message. Other one is Feedback-2, in order to send a NACK or STATIC-NACK. Additionally, Feedback-2 format contains the mode information for transition. Feedback-2 format is shown in Figure 3.6.

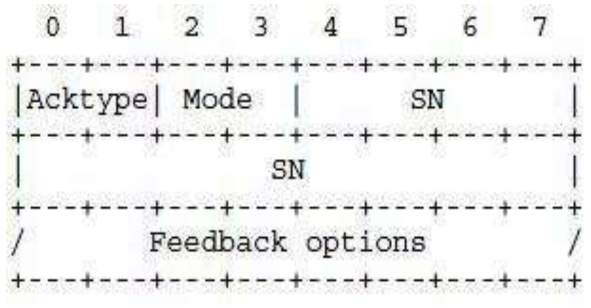


Figure 3.6: Feedback packet format [4]

Where:

SN = Sequence number
ACK type = Acknowledgement type

Mode: 0 = Reserved
1 = U-mode
2 = O-mode
3 = R-mode

3.3.4 Modes of Operation

ROHC can operate in three different modes. Each mode has its unique features that distinguish it from others. The mode selection is based on robustness, compression efficiency and reliability. It is possible to move from one mode to other but it is dependable on certain mode transition conditions. This sub section describes a brief introduction of each mode.

A. Unidirectional (U) Mode

There is only one forward path exists from compressor node to de-compressor node. Therefore, this mode is better choice where reverse path is unavailable or undesirable. The state transitions in compressor are performed at periodic time out interval and irregularities in the header field change patterns.

At the beginning of transmission, compressor sends full header information (static & dynamic) to de-compressor. This phenomenon is repeated by compressor until it is not confident that the de-compressor has received at least one full header. The amount of the full header is not fixed [7].

When the compressor gains the confidence that the de-compressor is in a position to receive compress header. Then, it moves forward to higher (FO & SO) states for compression as shown in Figure 3.7. The transitions to lower states are fixed according to state time out period.

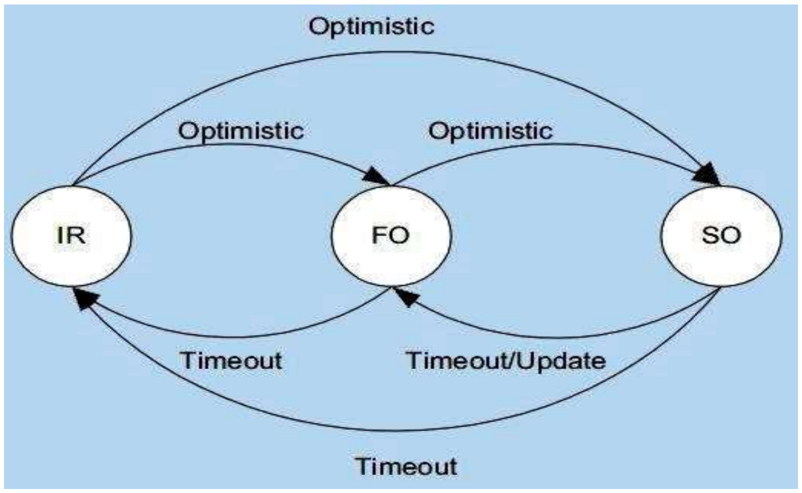


Figure 3.7: Unidirectional (U) mode [14]

Merits: Periodic updates are sent by compressor which makes the compressor and de-compressor synchronized.

Demerits: Feedback path is not supported therefore it is inefficient for error recovery and context synchronization. Furthermore, lower compression gain than other two modes due to periodic updates.

B. Bidirectional Optimistic (O) Mode

O-mode is similar to U-mode but it has two differences. One is the feedback channel for acknowledgement (ACK) or error recovery request (NACK) from de-compressor end. Second one is no periodic update from compressor end.

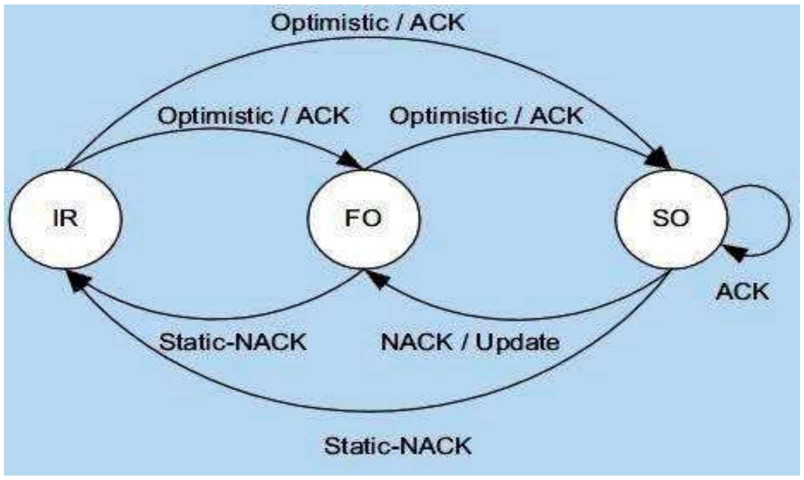


Figure 3.8: Bidirectional optimistic (O) mode [14]

The operation is such that a compressor sends full header packet to the de-compressor and waits for ACK. The de-compressor replied with an ACK message which indicates that it has built context table and ready for compress header. In case of packet loss/drop, the de-compressor sends NACK to compressor for retransmission of previous damage header. O-mode state transition is shown in Figure 3.8.

Note: Our ROHC library is based on O-mode which is further described in Chapter 4.

Merits: Higher compressions gain than U-mode because it does not update context periodically. Additionally, it reduces the packet loss with the availability of feedback channel.

C. Bidirectional Reliable (R) Mode

This mode is more robust than previous ones. The robustness is maximized with two things. First one, compressor sends periodic and repeated updates until acknowledgement is received from de-compressor. Other one is feedback channel for error recovery request indication. R-mode utilizes the secure reference principle instead of optimistic approach as used in O-mode.

Secure reference principle: The confidence level of the compressor depends on acknowledgements (ACK) from the de-compressor for every context updating packet. However, not every packet in R-mode updates the context [2] .

The compressor follows the secure reference principle and sends the context updating packets periodically and repeat until acknowledgement is received from the de-compressor end.

The periodic context updating packets reduce compression efficiency compared to O-mode. R-mode tries to improve the robustness in case of packet loss and bit errors. It has very low probability of the context invalidation but if it occurs, it can result in large number of damaged packets being delivered to upper layers [2]. Figure 3.9 depicts the state transitions in R-mode.

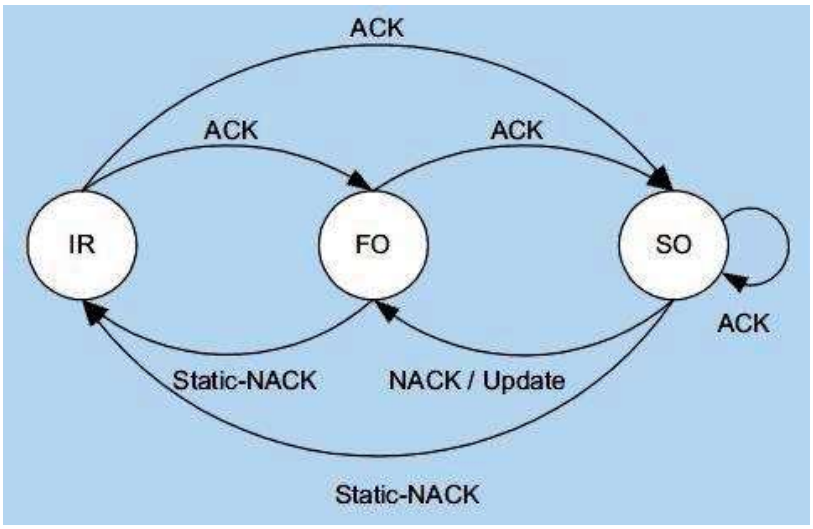


Figure 3.9: Bidirectional reliable (R) mode [14]

Merit: R-mode has lower context invalidation probability.

Demerit: It reduces the compression gain due to a lot of ACK and NACK messages.

Chapter 4 ROHC in Linux Based Router

This chapter gives a brief introduction of ROHC protocol integration in a Linux based router. It is jointly integrated in Vyatta Linux router by FFI (Norwegian Defence Research Establishment) & Thales Norway. The ROHC library implements the ROHC protocol as defined by the IETF RFC 3095. The library is implemented and published as an open source at https://launchpad.net/rohc. It provides a free ROHC implementation which conforms to ROHC standards [15].

The router comprises of an open source ROHC library and a Linux routing protocol which is built in the Vyatta framework. In our testbed scenario, an HP-Elite Book 8460p (Laptop) is working as vyatta Linux router and it has the following specifications .i.e. ROHC library version is 1.4.0 (released 17.05.2012), Kernel version is 3.0.32-1-586 Vyatta and Debian distribution 4.4.5-8.

4.1 Router User Space & Network Stack

A Vyatta Linux router is distributed in two functional blocks i.e. ROHC APP (Application) and Kernel space as shown in Figure 4.1. The ROHC APP block comprises of ROHC LIB (library) and PCAP (packet capture) LIB (library). The ROHC library performs the compression and de-compression tasks. The PCAP library captures the compressed packets at the incoming de-compressor interface which are assigned with a dedicated frame type (0x8888), as described in Section 4.2.1.

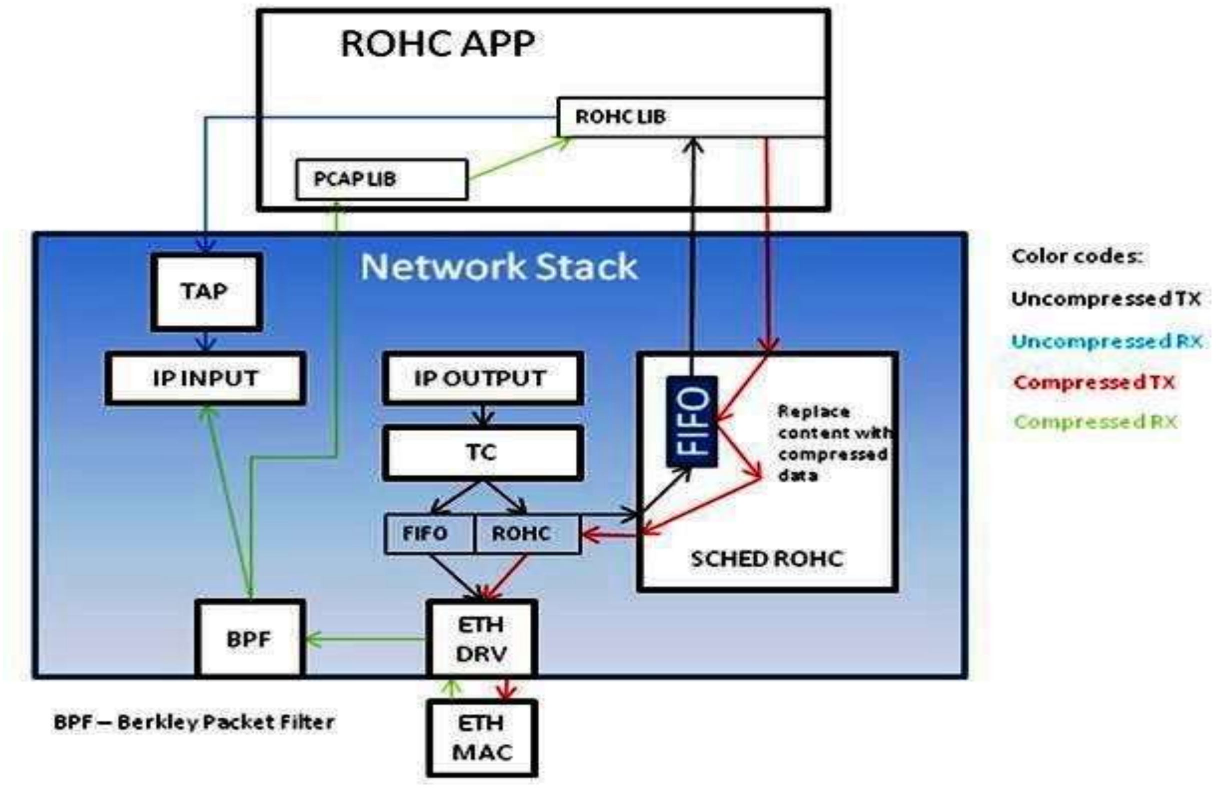


Figure 4.1: Vyatta Linux router functional blocks [1]

[1] Courtesy of Thales AS Norway.

Secondly, the kernel space has the following sub blocks i.e., SCHED ROHC (kernel module), Linux traffic control (*TC*) and TAP[2] module.

4.1.1Outgoing Traffic Flow

Traffic flow is described in the following order.

1. A normal IP packet reaches at the IP OUTPUT block from a traffic source or a local process.

2. Traffic control (*TC*) is part of the Linux iproute2 package which allows the users to access networking features. *TC* [3] could use to configure qdisc (queuing discipline) and packet classification in qdisc [16].

Queuing discipline: Packet queue with an algorithm which decides when to send which packet [16].

In the Linux traffic control block, packets are filtered with respect to TOS (type of service) field in the IP packet header.

4. ROHC queue sends the tag (TOS field) packets to SCHED ROHC where they are enqueued in the FIFO (first in first out) queue as shown in Figure 4.1. Packets wait in FIFO until they forward to the ROHC library. At the same instant, ROHC library is informed that packets are waiting in FIFO queue.

5. ROHC APP (application) module reads the packet (payload + header) and passes the packet header contents to the ROHC library for compression. The ROHC library compresses the packet header and sends back a compressed content towards the FIFO in SCHED ROHC module. At this stage packet payload is merged with a compressed header, called as **ROHC data**.

6. SCHED ROHC fetches original packet from FIFO and modifies the ROHC data contents with compressed packet contents (Ethernet header with type 0x8888). Finally the packet arrives at Ethernet driver which sends it on physical carrier towards de-compressor node.

4.1.2 Incoming Traffic Flow

1. Ethernet interface receives the incoming packet stream and forwards towards the BPF Berkley Packet Filter).

2. BPF provides a raw interface to data link layers. All packets on the network are accessible through this mechanism [17]. BPF sends original packet towards ROHC APP and IP INPUT block. IP INPUT neglects the compressed packet due to MAC header type 0x8888 (compressed header contents).

[2] TUN/TAP provides packet reception and transmission for user space programs. It can be viewed as a simple point-to-point or Ethernet device, which instead of receiving packets from a physical media, receives them from user space program and instead of sending packets via physical media writes them to the user space program [18].

[3] *TC:* Allows to perform packet shaping and adjust packet scheduling [19].

3. PCAP LIB (library) interface reads the compressed header packet and forwards it towards ROHC LIB. Finally, the decompress packet arrives at TAP interface that passes it towards the network stack input block. Hence, the decompress packet is treated as a normal IP (type = 0x800) packet for further actions.

4.2 Packet Formats

A new protocol number is defined for Ethernet frame which holds the ROHC compressed packet. This specific number distinguishes the ROHC data frame from normal IP Ethernet frame (uncompressed IP packet). In this way, Linux kernel at the de-compressor side would not interpret the ROHC-compressed payload of the frames as IP packets and would silently discard them [20].

The ROHC-compressed payloads frames are directly submit to the ROHC library by PCAP LIB (library) at the de-compressor end. There are two types of Ethernet frame for ROHC-compressed payload which are used for forward[4]and reverse[5]channels.

4.2.1 Ethernet/ROHC Data (0X8888) [20]

Compressor node transmits the Ethernet frame which holds the ROHC compressed data. The PCAP at the de-compressor node identifies the ROHC data with a specific protocol number .i.e. 0x8888, such as represented in Figure 4.2.

Ethernet Header SRC. MAC = 6 Bytes DEST. MAC = 6 Bytes Frame Type = 2 bytes	**ROHC Data** **46 - 1500 byte**

Figure 4.2: Ethernet frame for the 0x8888 protocol type [21]

Figure 4.2 depicts that 14 bytes are used for Ethernet header and rest for ROHC compressed data. ROHC compressed packet ranges from 46 byte to 1500 byte.

4.2.2 Ethernet/ ROHC Feedback (0x8889) [20]

De-compressor node sends ROHC compressed feedback towards compressor node for ACK & NACK. Feedback ROHC data size is normally shorter than 45 byte and it is labeled with protocol type 0x8889.

Ethernet Header SRC. MAC = 6 Bytes DEST. MAC = 6 Bytes Frame Type = 2 bytes	**ROHC Data** **0 - 45 byte**

Figure 4.3: Ethernet frame for the 0x8889 protocol type [21]

Furthermore, compressor node can also send packets with length shorter than 45 byte and protocol type 0x8889. Figure 4.3 shows the Ethernet frame for the protocol type 0x8889.

[4] Forward channels: Channel from compressor node to de-compressor node.
[5] Reverse channels: Channel from de-compressor node to compressor node.

4.2.3 De-compressed Packet

De-compressor node: ROHC library submits the decompressed header packet at TAP interface. The packet is treated as normal IP packet by the Linux kernel. The decompressed header packet format at TAP interface is illustrated in Figure 4.4.

Ethernet Header SRC. MAC = 6 byte DEST. MAC = 6 byte Frame Type = 2 byte	IP Header 20 byte	UDP/TCP Header 8 / 20 byte	Payload 0 -1500 byte

Figure 4.4: Normal IP packet at the de-compressor TAP interface

4.3 Linux Traffic Control (*TC*)

It is already described in Section 4.1.1 that input data packets are forwarded towards Linux traffic control. Traffic control enqueues the incoming packets according to selected queuing discipline. The classification can be configured with the traffic control tool Linux platform. Packets can enqueue in one of the ingress/egress queues method according to configured classification [16].

There are two types of queue, i.e. classless & class full [21]. In our case, HTB (Hierarchical Token Bucket) queue is used for traffic classification. Figure 4.5 illustrates the queue structure below.

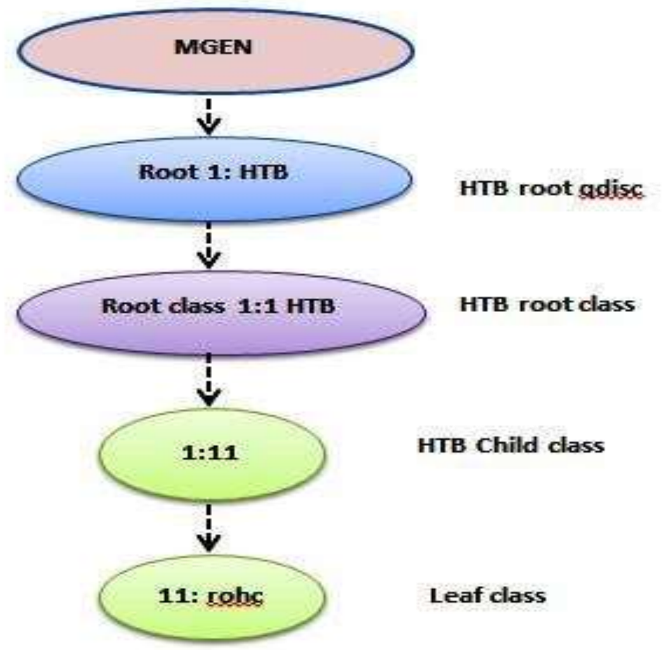


Figure 4.5: HTB class full queue discipline

In Section 4.3.1, it is briefly explained that how the 'HTB' queue classifies the packet stream.

4.3.1 HTB (Hierarchical Token Bucket)

HTB is a class-based egress qdisc [21]. It filters the incoming traffic and allows packet to be directed towards particular classes and sub-queues. Packet classification can be performed by analyzing the packet header fields (source & destination IP addresses, port numbers, TOS, etc.) and payload [21].

Figure 4.5 illustrates the HTB queue hierarchy. When traffic enters an HTB root 1: [6] qdisc, it is classified as follows.

1. HTB root queue 1: is the initial block of the queue discipline as described in command line (1). It divides the incoming packets into different classes. Packet classification is performed with the TOS (Type of service) field in the IP header. In our case, packets which need to be compressed are classified with TOS 0x28, 0x30 or 0x68 as given in Appendix A and in command line (2).

```
$TC[7]qdisc add dev $IF[8] root handle 1: htb default 16                                   (1)
$TC filter add dev $IF protocol ip parent 1:0 prio 1 u32 match iptos 0x28 0xff flowid 1:11      (2)
```

2. The root queue 1: classifies the traffic in root class 1:1 and further in sub-class 1:11, having link sharing and borrowing characteristics as described with command lines (3) & (4). Link bandwidth is assigned to the root classes which are further shared among the configured sub classes. If a sub-class (e.g. 1:11) meets it minimum capacity than it can borrow the bandwidth via root class 1:1 from the other sub-class e.g. 1:12, given that sub-class 1:12 has vacant bandwidth capacity.

```
$TC class add dev $IF parent 1: classid 1:1 htb rate[9]$RATE ceil[10]$RATE                  (3)
$TC class add dev $IF parent 1:1 classid 1:11 htb rate 1mbit ceil $RATE                     (4)
```

ROHC Queue

Leaf class 'rohc 11:' is sub-queue of sub-class 1:11 such as described in command line (5). It is a pfifo (packet first input first output) queue. The filtered packets wait in this queue. Therefore, they are ready to dequeue towards kernel module for further compression process.

```
$TC qdisc add dev $IF parent 1:11 handle 11: rohc limit[11] 5                                (5)
```

In the reverse direction, compressed data is dequeued in 'rohc 11:' queue by the kernel module. Finally it delivers to the Ethernet driver for data transmission.

[6] Root 1: Each qdisc is assigned a handle which is used for configuration statements to refer that qdisc. The handles of this qdisc consists of two parts, a major number and a minor number. The root qdisc '1:' equal to '1:0'. Class need to have the same major number as their parent [22].

[7] *TC*: The term specifies the 'queuing discipline' (qdisc) which is used to send outgoing packets at Ethernet interface [23].

[8] $IF: Interface .i.e. Ethernet (eth0)

[9] Rate: Maximum rate this class and all its children are guaranteed [24].

[10] Ceil: Argument specifies the maximum bandwidth that a class can use. This limits how much bandwidth that class can borrow [22].

[11] Limit: The number of bytes that can be queued waiting for tokens to become available [25].

Chapter 5 Testbed Configurations

Testbed configuration is based on lab scenarios where ROHC operation is tested in the normal and network emulation conditions. Additionally, operations are explained with the traffic flow diagrams and experimental observations. The practical work is divided in two core sections. Section 5.1 describes the normal ROHC operation with perfect (no loss) channel condition. On the other hand, Section 5.2 depicts the ROHC operation with channel emulation which reflects the error-prone channel conditions. The communication channel between ROHC devices (compressor & de-compressor) is based on hybrid network where channel is a physical Ethernet cable. Additionally Section 5.3 describes the collective observations in the realistic emulation scenarios.

5.1 ROHC Operation

ROHC operation is performed in a real-life scenario where two nodes (HP-Elite Book 8460p) are implemented as ’Vyatta’ Linux routers. Additionally, ROHC queue execution scrip is given in Appendix A. The execution scrip builds the queue discipline for Linux traffic control (*TC*) in the Vyatta routers. Hp-8460PX1 performs the compressor function and HP-8460PX2 as de-compressor as shown in Figure 5.1.

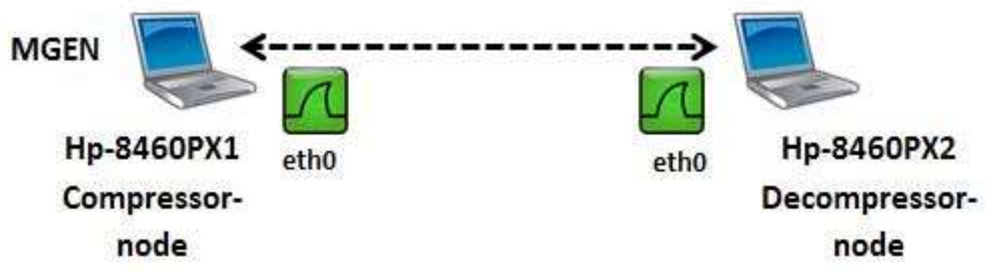


Figure 5.1: ROHC normal operation lab scenario

The traffic source (compressor) PX1 node generates the input traffic via MGEN[12] (multi-generator) version 5.0 with the following script parameters as provided in Appendix C MGEN script, where

Protocol type: UDP/IP, Traffic pattern: Periodic, Packet size=64 byte, Rate = 5 packets per second.

Figure 5.2 illustrates main functional blocks of the traffic nodes with the traffic flow chart. The operation is observed with the packet sniffer (Wireshark) at both nodes as represented in Figure 5.1. Since, we have two interfaces Ethernet (eth0) and TAP (tap0) at the de-compressor node. Therefore, compressed packets are captured on the Ethernet interface (eth0) and uncompressed at TAP (tap0) interface respectively. Furthermore, the mode of operation is ‘O-mode’ which is already described in Section 3.3.4.

[12] MGEN: Multi-Generator is open source software. It provides the ability to perform IP network performance tests and measurements using TCP/IP and UDP/IP traffic. The toolset generates real-time traffic patterns. Script files are used to drive the generated loading patterns over the course of time. The generated traffic can also be received and logged for analysis [26].

The brief description of the flow chart is followed here.

1. MGEN generates the IR header packets which is further passed towards the Linux traffic control (rohc 11: queue) and kernel module (SCHED ROHC). Afterwards, it is delivered to the ROHC LIB (library) for compression via FIFO queue. Since, compressor has three transition states where it starts from the initial state (IR) as represented by step 1 in Figure 5.2. In the IR state, compressor does not reduce the header original size and build a context table with a context ID e.g. CID =1.The CID (context id) contains the entire static and dynamic header fields information. Additionally, the CID contains the IR header which is known as reference header. The reference header has the v_ref value, as described in Section 3.2.4. At this stage, ROHC data packet size is equal to 91 bytes, where 64 byte is the packet size and remaining is the full UDP/IP header. Finally, kernel modifies the packet with Ethernet header (14 byte). At the end, IR-ROHC /IR header packet (= 105 byte, type=0x8888) is transported towards de-compressor node. It is normally sent three IR packets at one time which is implementation dependent.

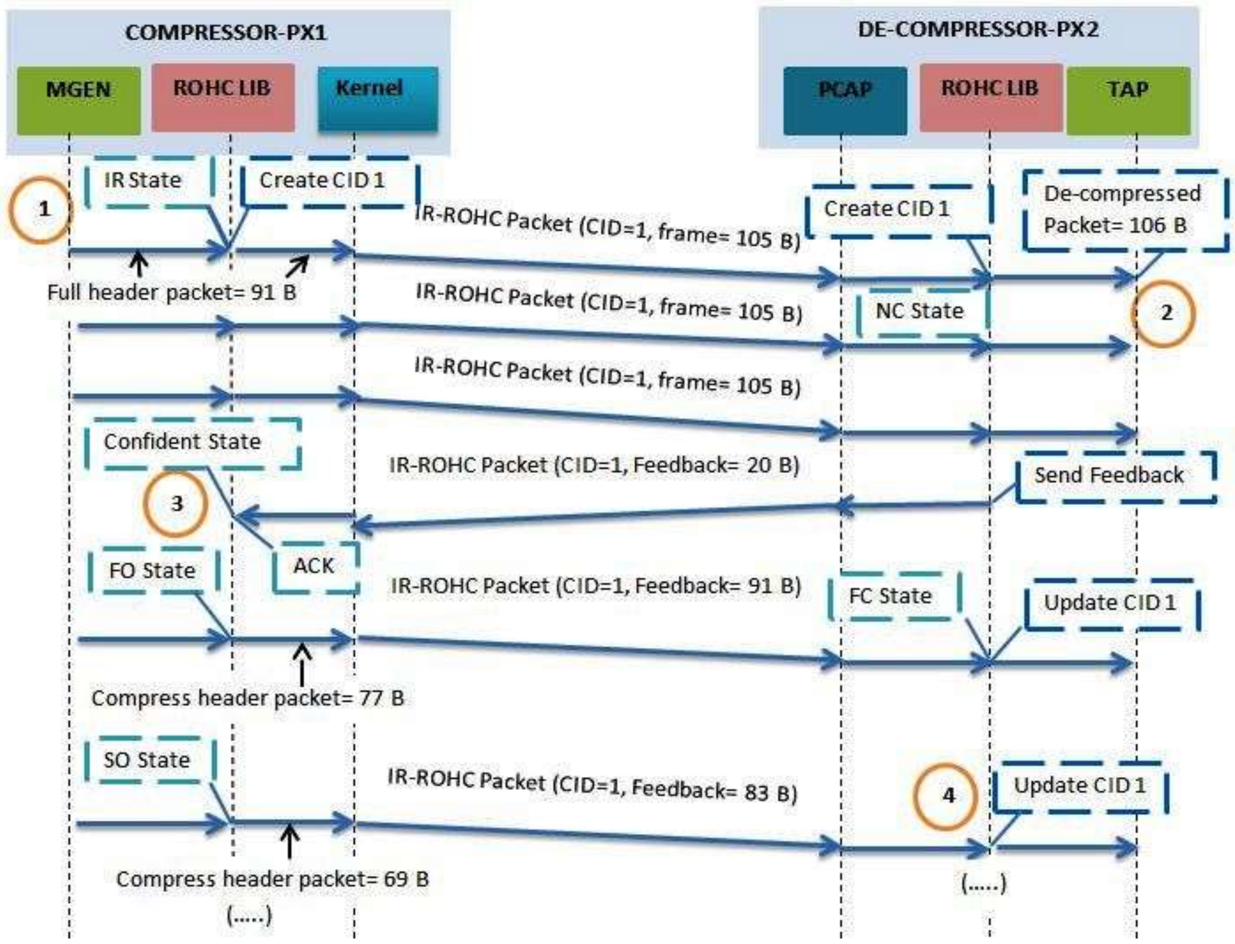

COMPRESSOR-PX1
MGEN
ROHC LIB
Kernel
DE-COMPRESSOR-PX2
PCAP
ROHC LIB
TAP
1
IR State
Create CID 1
IR-ROHC Packet (CID=1, frame= 105 B)
Create CID 1
De-compressed Packet= 106 B
Full header packet= 91 B
IR-ROHC Packet (CID=1, frame= 105 B)
NC State
2
IR-ROHC Packet (CID=1, frame= 105 B)
Confident State
IR-ROHC Packet (CID=1, Feedback= 20 B)
Send Feedback
3
FO State
ACK
IR-ROHC Packet (CID=1, Feedback= 91 B)
FC State
Update CID 1
Compress header packet= 77 B
SO State
IR-ROHC Packet (CID=1, Feedback= 83 B)
4
Update CID 1
Compress header packet= 69 B
(.....)
(.....)

Figure 5.2: ROHC operation traffic flows

Where, B =Byte
Full header packet = packet size + UDP header + IP header + Ethernet header (type= 0x8888)= 105 bytes
Compressor States: IR = Initialization and Refresh Packet, FO = First Order, SO =Second Order.
De-compressor State: NC = No Context, FC = Full Context.

2. De-compressor node (No Context-state): PCAP interface filters the ROHC IR packets with respect to unique protocol type (0x8888) and forwards towards ROHC library. The De-compressor stores IR header as reference header in the CID table. The CID number reflects the IR or reference header contents (v_ref) which should be match with compressor v_ref value. Furthermore, the packet is de-compressed by ROHC library and forward towards TAP interface. A feedback (size=20 byte, type=0x8889) is sent back to the compressor node which indicates that the de-compressor is able to transit in FC state for fully compressed header.

3. Compressor achieves the confidence level with the arrival of feedback (ACK) message. The state transition is upgraded to FO state at compressor end. UDP/IP header size is compressed up to 13 byte (approx. half). Hence, the compressed packet size is reduced to 77 bytes. Kernel modifies the packet with Ethernet frame (= 14 byte). Finally packet is transmitted on physical layer with size equals to 91 bytes.

4. De-compressor matches the current compressed header with the previous successfully decompress packet (IR header/ v_ref value) and update the context table. Then, state is updated to higher level .i.e. FC state. Finally the packet is de-compressed and delivers to the 'TAP' interface as normal IP packet (type= 0800).

Consequently, compressor reduces the header size from **28** bytes to **5** bytes in the SO state where the entire packet size equals to **83 byte** (type=0x8888). At this stage two ways transmission continues until de-compression failure or packet loss at de-compressor end. The experimental results are given in Section 6.1.

5.2 ROHC Operation over Imperfect Channels

Network emulation (Netem) is proposed for the performance evaluation of the ROHC protocol within the Vyatta Linux router. This emulation technique can modify the characteristics of the packet stream with respect to configured parameters. In this way, we will observe the ROHC protocol performance over channel packet loss and delay conditions respectively.

Netem is an enhancement of the Linux traffic control facilities that allows us to add delay, packet loss, and more characteristics to the packet stream at the selected network interfaces. In our testbed scenarios, network emulation is performed with two different lab scenarios for the ROHC protocol performance evaluations. The scenarios are traffic source (compressor) node and bridge node emulations respectively which are followed here.

5.2.1 Traffic Source Emulation Lab Scenario

In the traffic source emulation scenario, two nodes (HP-PX1 & HP-PX2) are directly connected through Ethernet cable as shown in Figure 5.2. Moreover, the emulation is performed at the source (compressor) node. Netem queue disc modifies the characteristics of packet stream generated by MGEN at traffic source node. In such case, an emulated data stream becomes the input of ROHC library.

Linux traffic control (*TC*) is modified with Netem queue discipline (disc) at the **compressor node PX1** for the emulation task. Netem queue discipline is given as script no.02 in Appendix B. In this scenario, Netem queue is the primary (root) queue of *TC* whereas 'HTB' queue works as secondary queue disc. Figure 5.3 depicts the queues hierarchy.

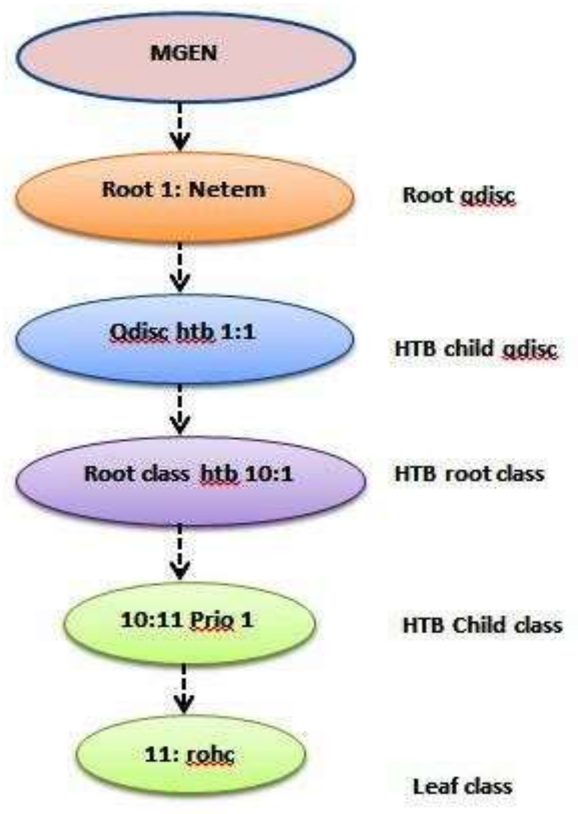


Figure 5.3: Traffic source emulation with Netem queue discipline

Netem queue emulates the MGEN normal IP traffic with loss or delay attribute. Then, the emulated packet stream is forwarded to the htb queue disc which performs its normal task as already explained in Section 4.3.1. The behavior of ROHC devices (compressor & de-compressor) is explained with the following traffic flow.

Traffic Flow Diagram

Traffic flow describes the ROHC operations over packet loss/delay emulation at the source (compressor) node. MGEN traffic (UDP/IP) is driven to the Netem queue. Header compression operation runs in a normal way where compressor node sends compressed header packet (= 83 bytes) without any loss in the SO state as shown in Figure 5.4. The emulation operation is explained with the following steps.

1. In packet loss condition: Netem queue emulates the generated packet stream with packet loss according to configured percentage at the compressor node. Moreover, the generated loss can be independent or burst pattern which depends on the Netem queue configured parameters. At this moment, ROHC LIB (library) has no input and compressor node does not send any packet towards de-compressor end. This reflects a real channel packet drop/loss for the de-compressor end.

2. The de-compressor node receives the first compressed header packet after packet loss duration. If the packet losses are independent then the de-compressor is able to de-compress the compressed header packet. Because the compressed header reference value has already reside in interpretation interval which is already stored in the context table. On the other side, if the generated packet loss is consecutive or burst pattern then de-compressor attempt additional decompression attempts or tries to decompress the next incoming packets at that instant.

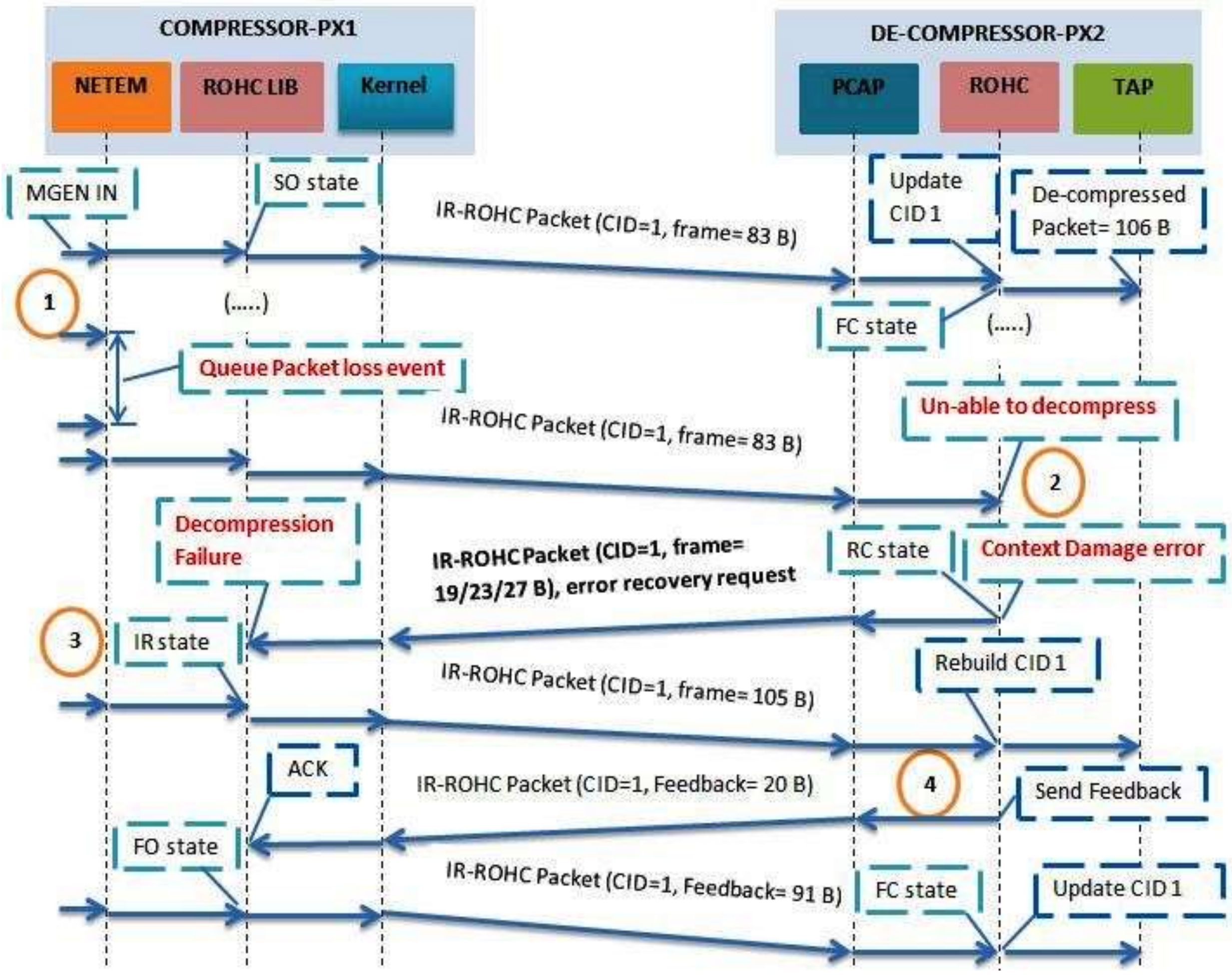

COMPRESSOR-PX1
NETEM
ROHC LIB
Kernel
DE-COMPRESSOR-PX2
PCAP
ROHC
TAP
MGEN IN
SO state
IR-ROHC Packet (CID=1, frame= 83 B)
Update CID 1
De-compressed Packet= 106 B
1
(.....)
FC state
(.....)
Queue Packet loss event
Un-able to decompress
IR-ROHC Packet (CID=1, frame= 83 B)
2
Decompression Failure
IR-ROHC Packet (CID=1, frame= 19/23/27 B), error recovery request
RC state
Context Damage error
3
IR state
Rebuild CID 1
IR-ROHC Packet (CID=1, frame= 105 B)
ACK
IR-ROHC Packet (CID=1, Feedback= 20 B)
4
Send Feedback
FO state
IR-ROHC Packet (CID=1, Feedback= 91 B)
FC state
Update CID 1

Figure 5.4: Traffic source emulation flow diagram

If additional attempts fail to de-compress the packet. Then, de-compressor transits to RC (repair-context) state. The reason is stated in RFC 3095 as follow.

"Many consecutive packets being lost between compressor and de-compressor (this may cause the LSBs of the SN in compressed packets to be interpreted wrongly, because the de-compressor has not moved the interpretation interval for the lack of input – in essence, a kind of context damage)". (RFC 3095, Section 5.3.2.2.3)

In RC (Repair-context) state, de-compressor sends an error recovery request (NACK) towards compressor node (PX1). Furthermore the de-compression failure is defined as.

De-compression failure state: The de-compressor fails to match the current header contents with the last successfully decompressed header, known as reference header. In this way, de-compressor is not able to update the LSB in the current compressed header according to reference header.

3. Compressor node is not aware from the de-compressor failure at the de-compressor end. It continues to send compressed header packets (= 83 bytes). When the compressor receives the NACK from de-compressor then it transits downward to IR state. It sends IR header (static plus dynamic fields) to the de-compressor node.

4. The de-compressor recovers from failure state with the arrival of IR header packet (= 105 byte) from the compressor end. The de-compressor stores the IR header as reference header and sends a feedback towards compressor end for acknowledgment. Finally, de-compressor state is updated to FC state and it is able to decompress the fully compressed header.

Limitation

We have observed a problem with the traffic source emulation scenario .i.e. the overall loss percentage at the compressor and de-compressor end does not remain equal. For example, if a Netem queue emulates loss percentage of 50% at source node then it should match with loss at the de-compressor end. But the received packet loss becomes 75% at de-compressor (PX2) node. The problem is further described in Chapter 7, Section 7.1.3.

Therefore, we needed a better testbed design to resolve this problem. Bridge node emulation testbed scenario resolved this issue efficiently.

5.2.2 Bridge Emulation Lab Scenario

This technique is used for single hop point-to-point nodes where bridge node behaves as physical communication channel for real-life case. Bridging connects multiple network segments (typically LAN segments) at the data link layer. It occurs at Layer 2 (data link layer) and IP addresses are not permitted on the interfaces being bridged. In our case, a bridge interface 'br10' is configured on a Vyatta Linux router (HP-8460PX3). The configuration of bridge is given in Appendix D.

The bridge interface bridges the compressor (HP-8460PX1) and de-compressor (HP-8460PX2) nodes with respect to Ethernet interfaces (eth0) respectively. The bridge (HP-8460PX3) interfaces (eth0 & eth1) are emulated by Netem queue for packet loss and delay respectively. Figure 5.5 demonstrates the single hop lab scenario for channel emulation.

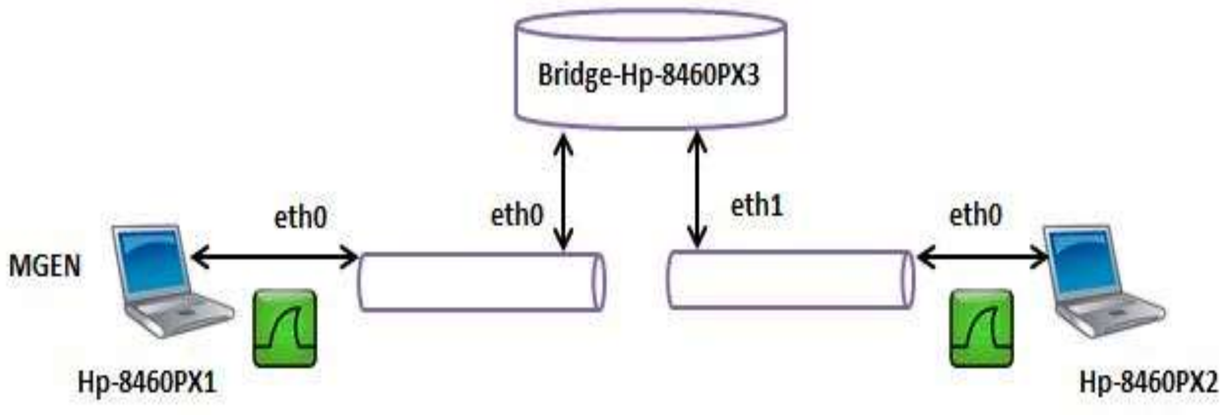


Figure 5.5: Netem bridge node scenario

In our bridge scenario, Ethernet interface eth1 of bridge (HP-8460PX3) is emulated for outgoing traffic towards de-compressor (HP-8460PX2). Furthermore, second Ethernet interface eth0 is emulated for feedback/error recovery request channel towards compressor (HP- 8460PX2). Packet sniffer 'Wireshark' captures the traffic at compressor 'eth0' interface for outgoing compressed packets flow. Similarly, packets are captured at the de-compressor Ethernet (eth0) interface for incoming compressed traffic and TAP (tap0) interface for de-compressed traffic respectively.

Hence, we will count the number of transmitted and received packets at compressor and de-compressor ends respectively. Consequently we will extract the packet loss percentage due to channel emulation.

Emulation Scenarios

This section compromises of two different bridge emulation scenarios .i.e. forward channel and reverse channel emulation respectively. It determines the impact of channel random packet loss (independent or burst) and delay attributes over compressor and de-compressor behaviors. This reflects a real network scenario. We have observed in our test results that independent packet losses do not cause de-compression failures unless it happens at the starting period of ROHC operation in Section 6.2.

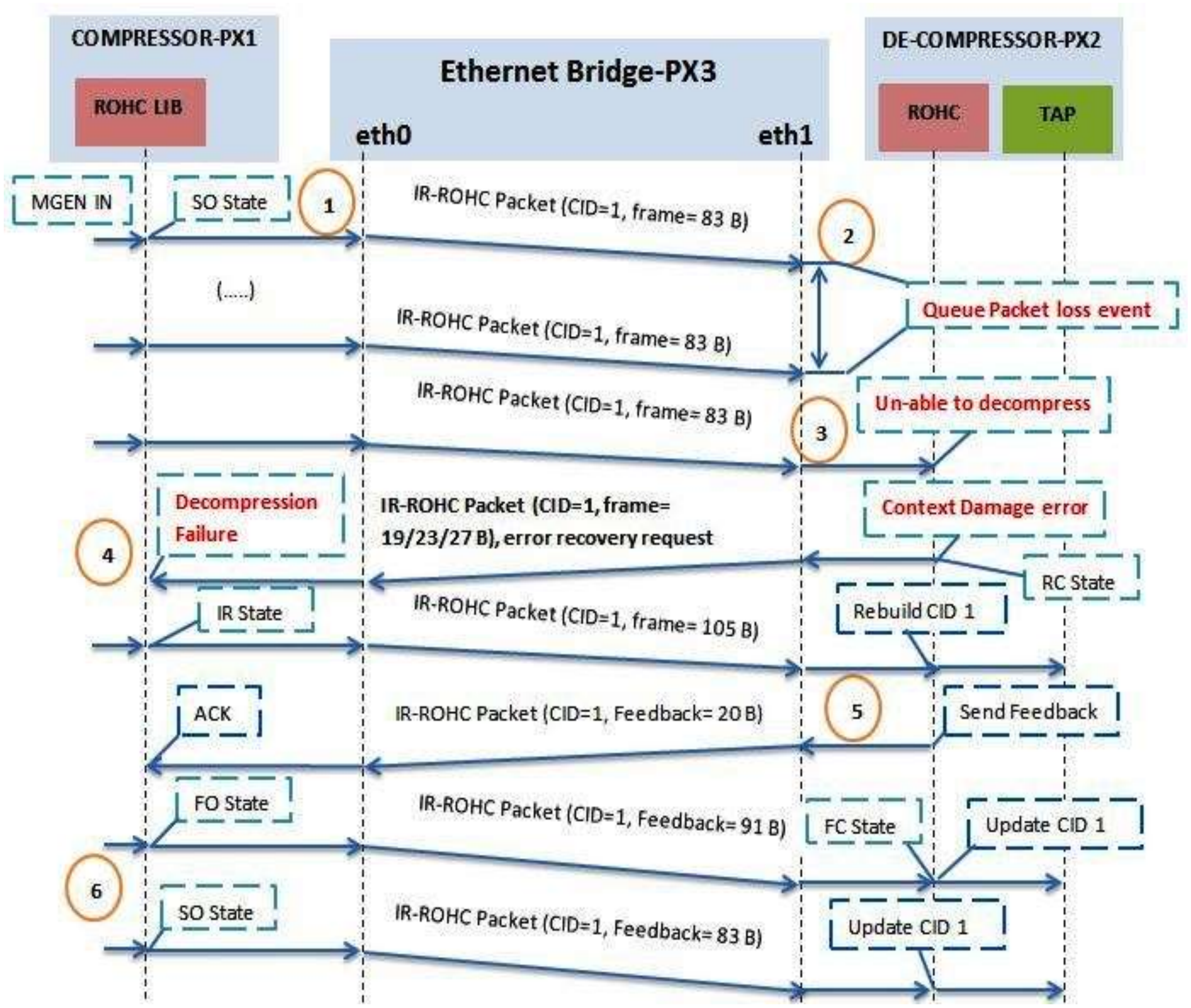
A correlation attribute is used to emulate the burst/consecutive packet losses but it is not acceptable due to weakness of Netem correlated loss generator [27] [28]. This problem is further explained in Section 7.2.1. **GI loss model**[13] is proposed to resolve the loss correlation problem in Netem queue.

[13] GI loss model: GI (General and Intuitive) loss model is used to fully characterize the 4-state Markov model. It is expressed with 5 parameters [Loss probability (P), Mean burst length E (B), loss density within the burst (ρ), independent loss probability (PISOL), and mean good burst length E (GB)] to emulate the independent and burst loss patterns [27].

1. Forward Channel Emulation

In a normal operation ROHC devices synchronize and work efficiently without any failure. Netem queue emulates the outgoing traffic at Bridge node (HP-8460PX3) Ethernet interface 'eth1' as shown in Figure 5.6. Traffic Flow diagram depicts the context damage and repair mechanism during de-compression failure instants performed by ROHC devices, as shown in Figure 5.6 which is described in the following steps.

1. The compressor (HP-8460PX1) is working in SO (second-order) state where it only sends the fully compressed header packet (size=83 byte, type=0x8888). On the other hand, de-compressor decompresses the packets in FC state.



COMPRESSOR-PX1
ROHC LIB
Ethernet Bridge-PX3
eth0
eth1
DE-COMPRESSOR-PX2
ROHC
TAP
MGEN IN
SO State
1
IR-ROHC Packet (CID=1, frame= 83 B)
2
(.....)
Queue Packet loss event
IR-ROHC Packet (CID=1, frame= 83 B)
IR-ROHC Packet (CID=1, frame= 83 B)
3
Un-able to decompress
Decompression Failure
4
IR-ROHC Packet (CID=1, frame= 19/23/27 B), error recovery request
Context Damage error
RC State
IR State
IR-ROHC Packet (CID=1, frame= 105 B)
Rebuild CID 1
ACK
IR-ROHC Packet (CID=1, Feedback= 20 B)
5
Send Feedback
FO State
IR-ROHC Packet (CID=1, Feedback= 91 B)
FC State
Update CID 1
6
SO State
IR-ROHC Packet (CID=1, Feedback= 83 B)
Update CID 1

Figure 5.6: Netem bridge forward channel emulation

2. Bridge (HP-8460 PX3) emulates the outgoing traffic at Ethernet (eth1) interface for packet loss via Netem queue. At this instant, de-compressor experiences packet drop/loss in data packet

stream. If the packet loss is independent pattern then the de-compressor does not lose the synchronization which is same as traffic source emulation scenario.

3. De-compressor receives the first incoming compressed header (CO) packet (=83 byte) after consecutive packet loss. Then it compares current packet header with the last successfully decompress header context. At this stage, de-compressor has not moved the reference value in the interpretation interval for the lack of input as already described in Section 5.2.1. In such condition, de-compressor is not able to verify the CRC in the full context (FC) transition state. Hence, it fails to de-compress the compressed header packet fields, known as de-compression failure state. On the other hand, compressor is unaware of the de-compression failure state therefore it sends continuously CO header packets (=83 byte). At this stage, all packets are discarded at the de-compressor Ethernet interface eth0 due to de-compression failure. The output of the TAP interface is zero at this moment and all packets are lost due to de-compression failure.

4. De-compressor stays continuously in the de-compression failure state then it sends NACK towards compressor node for context repair.

5. The compressor received the NACK and moves it state to downward direction .i.e. IR. It matches the last successfully decompressed header sequence number in the context table. Then, it sends the IR header packets (= 105 byte), as shown in Figure 5.6. The number of IR header is dependent on implementation. The de-compressor stores the IR header as new reference header. Moreover, compressor sends partially compressed headers packet (91 byte) which indicates the FO state.

6. Finally, de-compressor device comes back in the FC state where it decompresses the fully compressed header (83 byte) packets.

The results that show the de-compressor failure instant during lab test scenarios are given in Section 6.2 & 6.3.

2. Forward and Reverse Channels Emulation

Emulation is also performed for the forward & reverse channels conditions. The emulation scenario is divided in two conditions which are followings.

Condition 1: Forward & reverse channel loss attribute:

Bridge interfaces eth0 and eth1 are configured for packet loss in the outgoing direction as shown in Figure 5.7. In such condition over all packet loss increases during de-compression failure due to NACK lost at the bridge interface eth0 and does not receive at compressor end as marked by step 1 in Figure 5.7.

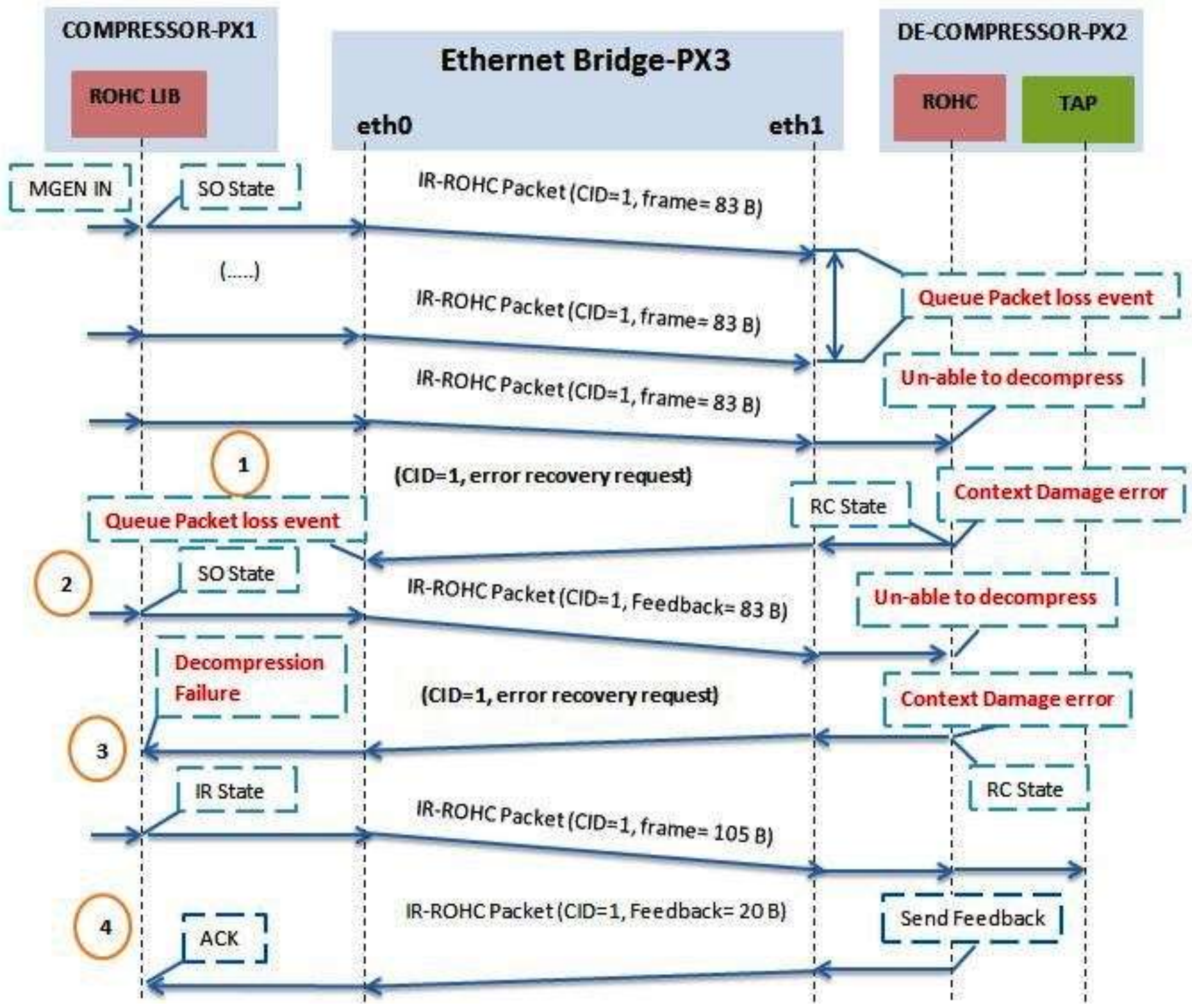


Figure 5.7 : Netem bridge forward & reverse channels emulation

1. The de-compressor sends NACK for error recovery but it is dropped at bridge PX3 interface eth0.
2. Compressor node is not aware of the failure at de-compressor and sends compressed packets continuously. All packets are discarding at de-compressor end. The output of the TAP interface is zero at this moment and all packets are lost due to de-compression failure.
   In step 3 & 4 compressor received the NACK and sends IR header respectively. De-compressor stores the IR header as new reference header.

Condition 2: Forward channel is emulated with Loss (%) & reverse channel with delay:

Bridge PX3 interface eth1 is emulated with packet loss percentage for forward channel. Similarly, bridge interface eth0 is emulated with packet delay for reverse channel.

It is concluded that the packet loss increases at TAP interface during de-compression failure instant at the de-compressor end. It is observed that NACK reaches at the compressor node with time delay during the de-compression failure state. At this moment, all received packets are lost at the de-compressor Ethernet (eth0). Hence, this also increases the overall packet loss at TAP interface.

The experimental results are given in Section 6.2.2 & 6.3.2.

5.3 Testbed Observations

We have observed identical observations with both testbed scenarios (Sections 5.2.1 & 5.2.2).These are followed here.

5.3.1 State Transitions

In the beginning of communication setup, compressor node sends three IR headers (105 byte) at the same time and upgrade it states to FO. Afterwards, compressor sends three partially compressed header packets (91 byte) in the FO state and transit to SO. It shows that compressor does not wait for feedback message for state transition from de-compressor as described in O-mode. The number of IR and partially compressed headers packets is implementation dependent.

It shows that ROHC library does not fully compliance with the RFC 3095 for feedback options [29] [30].

5.3.2 Independent Packet Loss

Independent packet losses are the errors happen individually and are independent of each other. It is already described in the emulation scenarios that independent packet losses do not affect the header compression operation. But there is a rare case where independent packet loss can create a de-compression failure state. The case is followed here.

The compressor sends IR header towards de-compressor at the communication start time period. But de-compressor does not receive IR header due to channel packet loss. Then, compressor sends partially compressed (91 byte) and fully compressed (83) header packets. The de-compressor node directly receives the compressed header in NC state. At this moment, de-compressor has not reference header in the context table therefore it discards all packets at Ethernet interface. The RFC 3095 describes the following condition such as.

"In the No Context state (de-compressor end): The de-compressor discards all packets until a static update (IR) which passes the CRC check is received". RFC 3095, Page 59.

In this way de-compression failure occurs at the de-compressor end. Finally, the de-compressor sends NACK to the compressor node for IR header.

5.3.3 Consecutive / Burst Packet Loss

It is noted that the de-compressor failure occurs when the consecutive packet loss equals or greater than 30 seconds. Because the INTERVAL = a (i) – a (i-1) (time elapsed between the arrival of the previous, correctly decompressed packet and the current packet) exceeds from the inter-arrival times. It is written in the RFC 3095 such as follow.

“Inter- packet time: If the INTERVAL is judged to be at least 2^k (k = least significant bits) packet inter-arrival time, the de-compressor adds 2^k to the reference SN and attempts to decompress the packet using the new reference SN (sequence number). (RFC 3095, Section 5.3.2.2.4)”

The inter-packet time is implementation dependent. It is experienced 30 second in our realistic scenario.

5.3.4 Error Recovery Request / NACK

Consecutive packet loss pattern: NACK is normally sent at a specific time period during de-compression failure instant. It is observed that this period ranges from 1 second to 2 seconds when first compressed packet is received after consecutive packet loss duration. The error recovery request contains the last successfully de-compressed header sequence number. In this way, the compressor identifies the next full header compress packet which is sent to de-compressor end.

Note: Feedback (NACK) packet is protected with CRC option and size is not fixed. It depends on de-compressor which field information it required. In our observations, the NACK packet size varies during tests i.e. 19 byte/23 byte/27 byte & 31 byte. NACK is only sent when decompression of several consecutive packets has failed or a consecutive packet loss at de-compressor end [7]. Moreover, the feedback is rate limited [7]. The rate-limiting algorithm may add latency to error recovery mechanism. Actually, it is a compromise between quick recovery on errors and over- reacting on sporadic problems (= too many transmitted feedback & transitions to lower compression states).

Chapter 6 Experimental Results

The experimental results are based on the bridge emulation lab scenario which is already described in Section 5.2.2. This chapter is divided into three sections. Moreover, each section is described with practical observations which are based on test results and RFC 3095.

Section 6.1 describes the ROHC protocol operation results over error free channel condition which is already described in Section 5.1.

Sections 6.2 and 6.3 described the ROHC operation results for testbed channel emulation scenarios. Furthermore, the channel emulation is achieved with Netem (network emulator) at the bridge HP-8460PX3 for packet loss & delay, as already described in Section 5.2.2. The testbed results are based on two types of packet loss patterns that are observed over an imperfect channel conditions. First one is independent packet loss where the errors happen individually and are independent of each other over the channel. Second one is the burst packet loss in terms of consecutive packet loss pattern which has longer periods of high loss density. These long periods of high loss density can have significant effect on e.g. Voice over IP [31]. Furthermore, all observations are based on Bridge emulation lab scenario (Section 5.2.2).

MGEN generates the packet flow for 1000 seconds with following parameters.

Protocol type: **UDP/IP**, Traffic pattern: **Periodic**, **Packet size=64** byte, **rate = 5 packets/second**, Source IP = **192.168.1.12**, Destination IP = **192.168.1.13**. MGEN scrip is given in **Appendix C**. The maximum data bit rate for single test duration is **2.56 Kbit/second .i.e., 5 x 64 x 8 = 2560 bit/sec**.

Each section constitutes the experimental measurements which are taken at the de-compressor node interfaces (eth0 & tap0) as shown in Figure 5.6.

Ethernet (eth0) interface: Received the compressed input packet stream from compressor at de-compressor end.

TAP (tap0) interface: Received the de-compressed input packet stream from ROHC library at de-compressor end.

Note: All observations are based on Wireshark trace analysis which captures the data packets at the de-compressor Ethernet (eth0) and TAP (tap0) interfaces respectively.

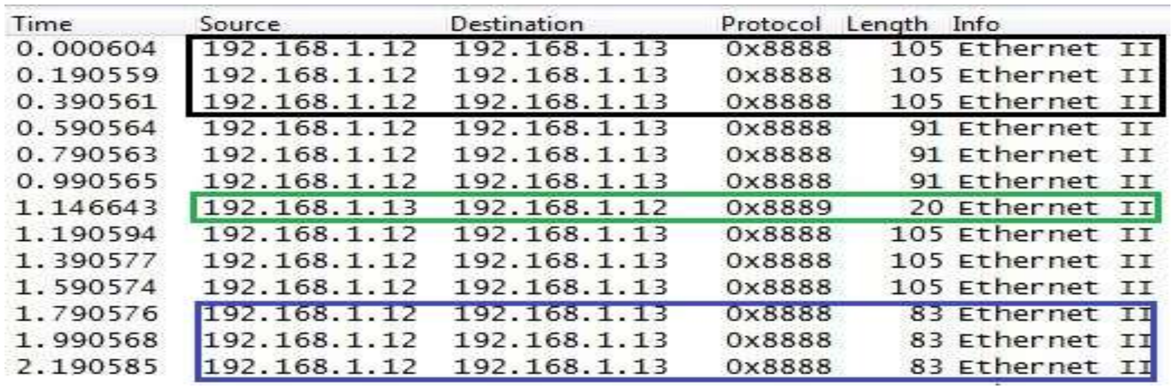
6.1 ROHC Operation

The observations for the test case in Section 5.1 are followed here.

6.1.1 De-compressor Ethernet Interface (eth0)

ROHC normal operation trace analysis is shown in Figure 6.1.The compressor (IP = 192.168.1.12) node sends IR header packets (size = 105 byte) towards de-compressor (IP = 192.168.1.13) node as marked with black line in Figure 6.1. In the beginning, compressor sends three IR headers and three partially compressed (size = 91 byte) headers. Additionally, MAC protocol packet type is 0x8888 that is unique for compressed header packet. The de-compressor receives the IR header and stores it in the context table as a reference header. The de-compressor replied the compressor with a feedback message (size = 20 byte) of MAC protocol packet type 0x8889 as marked with green line in Figure 6.1. Then, the compressor again sends full header packets (= 105 byte). Afterwards, compressor directly moves towards higher state SO (packet size = 83 byte). Since, the ROHC devices follow the O-mode of operation. Therefore, compressor should wait for the feedback after IR header but it moves to next higher state FO where it sends partially compressed header packet (= 91 bytes).

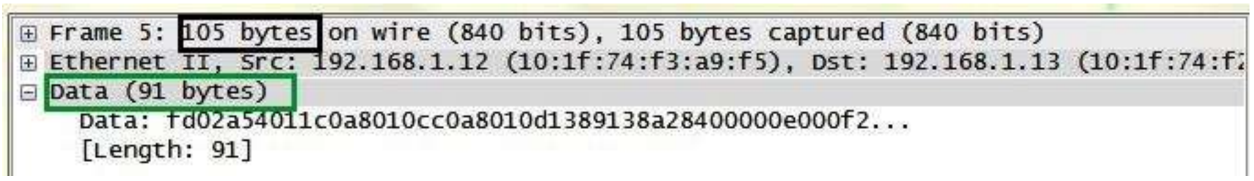
It is already described in Section 5.3.1 that compressor does not follow the state transition mechanism according to RFC 3095.



```
Time       Source        Destination   Protocol  Length Info
0.000604   192.168.1.12  192.168.1.13  0x8888     105 Ethernet II
0.190559   192.168.1.12  192.168.1.13  0x8888     105 Ethernet II
0.390561   192.168.1.12  192.168.1.13  0x8888     105 Ethernet II
0.590564   192.168.1.12  192.168.1.13  0x8888      91 Ethernet II
0.790563   192.168.1.12  192.168.1.13  0x8888      91 Ethernet II
0.990565   192.168.1.12  192.168.1.13  0x8888      91 Ethernet II
1.146643   192.168.1.13  192.168.1.12  0x8889      20 Ethernet II
1.190594   192.168.1.12  192.168.1.13  0x8888     105 Ethernet II
1.390577   192.168.1.12  192.168.1.13  0x8888     105 Ethernet II
1.590574   192.168.1.12  192.168.1.13  0x8888     105 Ethernet II
1.790576   192.168.1.12  192.168.1.13  0x8888      83 Ethernet II
1.990568   192.168.1.12  192.168.1.13  0x8888      83 Ethernet II
2.190585   192.168.1.12  192.168.1.13  0x8888      83 Ethernet II
```

Figure 6.1: Compressed traffic at de-compressor Ethernet interface eth0

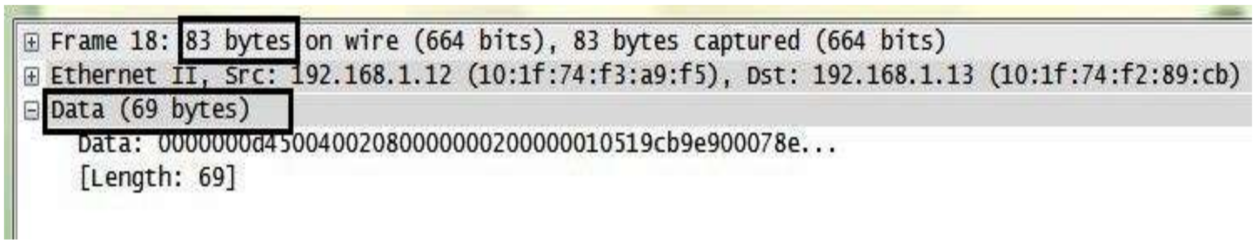
IR header packet format is shown in Figure 6.2. The data field (91 bytes) comprises of data (= 64 byte) and UDP/IP header (= 28 byte).



```
⊞ Frame 5: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
⊞ Ethernet II, Src: 192.168.1.12 (10:1f:74:f3:a9:f5), Dst: 192.168.1.13 (10:1f:74:f
⊟ Data (91 bytes)
    Data: fd02a54011c0a8010cc0a8010d1389138a28400000e000f2...
    [Length: 91]
```

Figure 6.2: IR header packet

The compressed header packet format is shown in Figure 6.3. The data field (69 bytes) in the packet format represents the ROHC data packet as already described in Section 4.2.1. In such case, packet consists of data plus compressed header. The data size is 64 byte whereas compressed header size is 5 byte. Hence, the header is compressed from 28 byte to 5 byte.



```
⊞ Frame 18: 83 bytes on wire (664 bits), 83 bytes captured (664 bits)
⊞ Ethernet II, Src: 192.168.1.12 (10:1f:74:f3:a9:f5), Dst: 192.168.1.13 (10:1f:74:f2:89:cb)
⊟ Data (69 bytes)
    Data: 000000d4500400208000000020000010519cb9e900078e...
    [Length: 69]
```

Figure 6.3: Compressed header packet

Test Result Graph

A graph representation is extracted from Wireshark capture log file for a single test time duration. X-axis starts from left side from 0 seconds and ends at 1000 seconds on right side. Similarly, y-axis starts from right side at point 0 and continue to upward direction. It shows the received compressed data traffic for the test duration of 1000 seconds at Ethernet interface eth0.

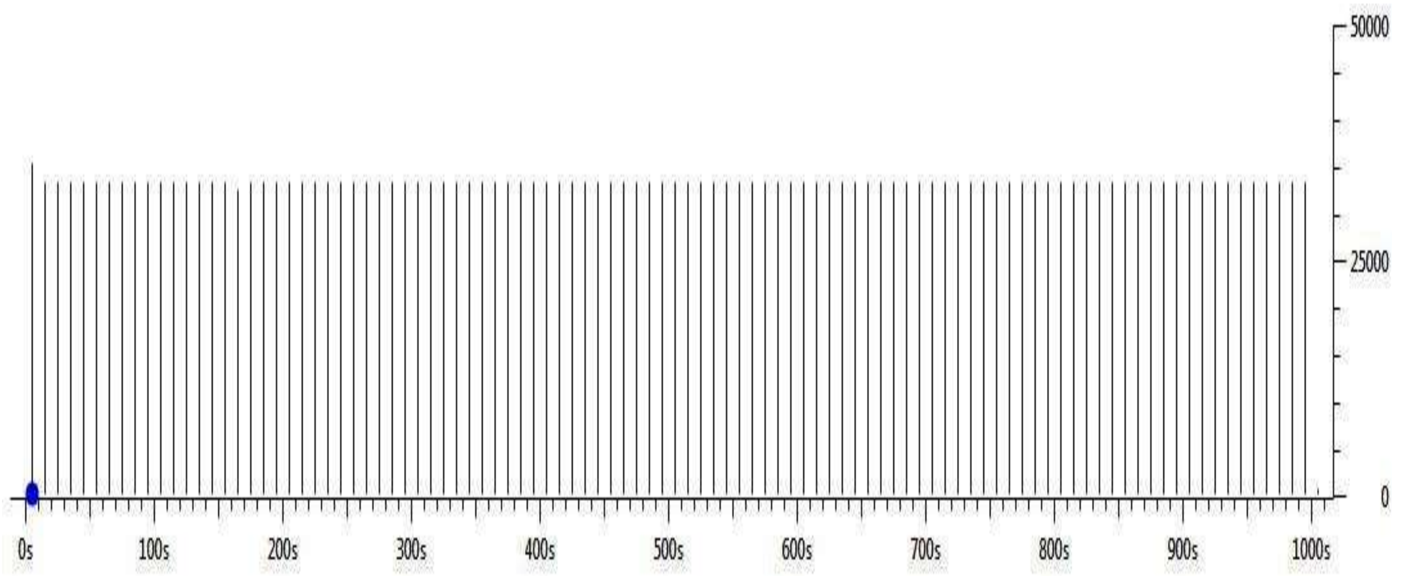


Figure 6.4: ROHC operation graph

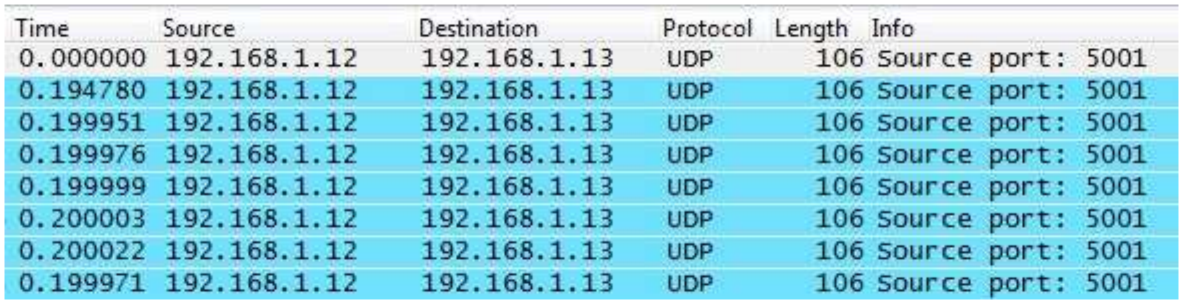
Where,

X-axis is divided into ticks where one tick = 10 seconds. Similarly Y-axis represents bit /tick.

The blue dot represents the feedback message (20 byte) on the graph in Figure 6.4. The constant data rate is equal to 2.56 Kbit/s (25600/10) on the Y-axis. Furthermore, there is no packet loss through the test time duration. Afterwards, PCAP block forwards the compressed ROHC data packets towards ROHC library for de-compression process as described in Section 5.1, Figure 5.2.

6.1.2 De-compressor TAP Interface (tap0)

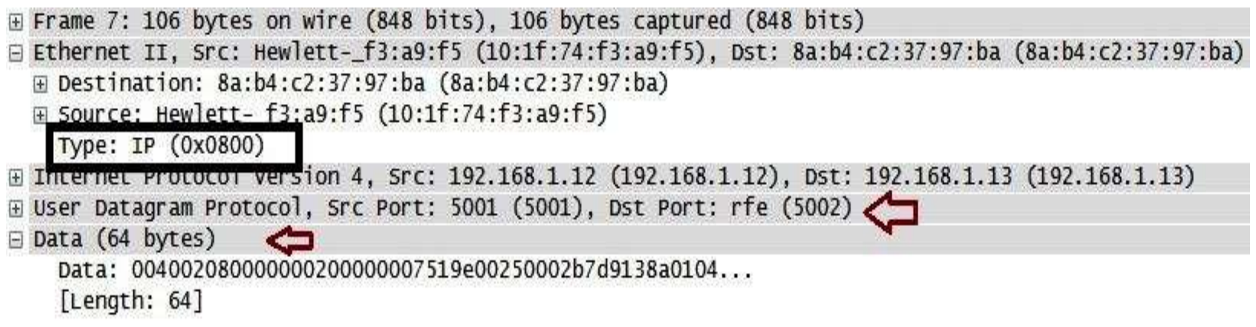
The compressed packets are de-compressed by the ROHC LIB (library) and deliver to the TAP device as shown in Figure 5.2. TAP interface forwards the de-compressed packets to the Linux kernel as normal IP packets. Figure 6.5 represents the normal UDP/IP packet after de-compression at TAP interface.



```
Time      Source        Destination   Protocol Length Info
0.000000  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.194780  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.199951  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.199976  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.199999  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.200003  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.200022  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
0.199971  192.168.1.12  192.168.1.13  UDP         106 Source port: 5001
```

Figure 6.5: De-compressed packet stream at de-compressor TAP (tap0) interface

Figure 6.6 shows the de-compressed packet format. The packet Ethernet type 0x8888 is replaced to 0x800 which shows a normal IP packet. Additionally, UDP/IP header is recovered in original size which is equal to 28 bytes.



```
⊞ Frame 7: 106 bytes on wire (848 bits), 106 bytes captured (848 bits)
⊟ Ethernet II, Src: Hewlett-_f3:a9:f5 (10:1f:74:f3:a9:f5), Dst: 8a:b4:c2:37:97:ba (8a:b4:c2:37:97:ba)
  ⊞ Destination: 8a:b4:c2:37:97:ba (8a:b4:c2:37:97:ba)
  ⊞ Source: Hewlett-_f3:a9:f5 (10:1f:74:f3:a9:f5)
    Type: IP (0x0800)
⊞ Internet Protocol Version 4, Src: 192.168.1.12 (192.168.1.12), Dst: 192.168.1.13 (192.168.1.13)
⊞ User Datagram Protocol, Src Port: 5001 (5001), Dst Port: rfe (5002)
⊟ Data (64 bytes)
    Data: 0040020800000002000000007519e00250002b7d9138a0104...
    [Length: 64]
```

Figure 6.6: De-compressed packet format at TAP interface

Test Results

Table 2 describes the captured packets detail at compressor and de-compressor nodes respectively. It shows that all transmitted compressed packets are received at de-compress Ethernet (eth0) interface. Furthermore, compressed packets are successfully de-compressed at TAP interface.

TX-Packets (eth0)-Compressor	RX-Packets (eth0) de-compressor	RX-Packets (tap0)- de-compressor	Drop Packets (eth0)-de-compressor	Drop Packets (tap0)- de-compressor	Loss (%) eth0	Loss (%) Tap0
5000	5000	5000	0	0	0	0

Table 2: ROHC operation

6.2 Independent Packet Loss Pattern

We have observed that the independent packet loss pattern can affect the ROHC operation if it occurs at the beginning of communication setup i.e. when a de-compressor does not receive the IR header due to channel packet loss.

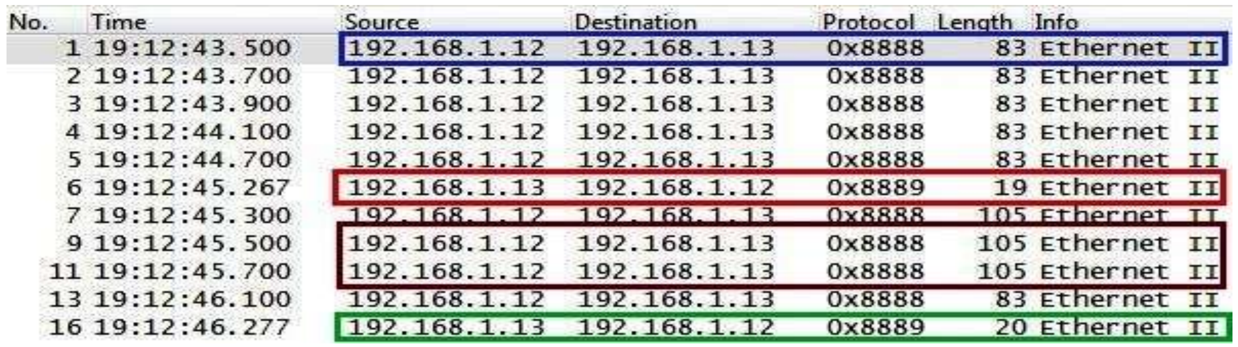
We have performed 10 tests for each channel emulation scenario. The time duration of each test is 1000 seconds. MGEN generates the packet at a rate = 5 packets/second at compressor node as given in script Appendix C. Therefore, 2.56 Kbit are transmitted in 1 second. Moreover, compressor sends 5000 packet during one test duration. We have evaluated the performance in two conditions which are followed.

6.2.1 Forward Channel Emulation

In such condition, forward channel is only emulated for independent loss pattern. The Netem emulation script is given in Appendix B, script no.06 where Bridge PX3 Ethernet interface (eth1) is emulated for this purpose.

The trace analysis is described in Figure 6.7 where compressed packets are captured at the de-compressor (eth0) Ethernet interface for 50% channel loss. It describes the general description which is identical for configured loss values .i.e., 50% & 60%. Additionally, the de-compression failure behavior is observed only at the beginning of the communication setup for 50% & 60% loss respectively. There is no de-compression failure for 40% channel loss emulation.

Figure 6.7 represents that the IR header packets (= 105 byte) are lost due to forward channel emulation at the Bridge interface eth1. The de-compressor receives the first packet (= 83 bytes) which is compressed as marked with blue line in Figure 6.7. At this stage, de-compressor is operating in NC (No Context) state. It does not hold the reference header in the context table therefore, CRC checksum fails. Because, the de-compressor is not able to directly de-compress the compressed header in the first de-compression attempt as described in Section 5.3.2.



No.	Time	Source	Destination	Protocol	Length	Info
1	19:12:43.500	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
2	19:12:43.700	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3	19:12:43.900	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
4	19:12:44.100	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
5	19:12:44.700	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
6	19:12:45.267	192.168.1.13	192.168.1.12	0x8889	19	Ethernet II
7	19:12:45.300	192.168.1.12	192.168.1.13	0x8888	105	Ethernet II
9	19:12:45.500	192.168.1.12	192.168.1.13	0x8888	105	Ethernet II
11	19:12:45.700	192.168.1.12	192.168.1.13	0x8888	105	Ethernet II
13	19:12:46.100	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
16	19:12:46.277	192.168.1.13	192.168.1.12	0x8889	20	Ethernet II

Figure 6.7: Trace analysis of forward channel independent packet loss pattern

At the de-compression failure moment, the ROHC library continues to generate a log message in Vyatta router log file .i.e.

```
19:12:43 vyatta rohc: ETH0 read_eth (rohc.c:779): Not able to decompress, drop packet. rohc_len -1
19:12:45 vyatta rohc: last message repeated 4 times
```

First message is generated at time 19:12:43, frame # 01 when the first compressed header packet is received. The ROHC library generates the log message for approximately 2 seconds. Then, de-compressor sends NACK (19 byte) towards compressor node (IP address = 192.168.1.12) as marked with red line in Figure 6.7.The compressor node receives this NACK and sends the IR header packets (105 bytes). De-compressor receives the IR header and replies with an ACK (20 bytes) as mark with green line in Figure 6.7. In such condition, all received compressed packets at Ethernet interface (eth0) are discarded during the de-compression failure duration at de-compressor node. Hence, 5 packets are lost at TAP interface which are discarded at Ethernet (th0) interface. These packets are considered as packet drop due to header compression failure at TAP interface.

Test Result Graph

Figure 6.8 shows the IO graph which represents the traffic flow for 50 % packet loss test (duration = 1000 seconds). The purpose of the graph is to represent the de-compression failure instant for test result. It demonstrates the NACK (19 bytes) with red dot and ACK (20 bytes) with blue dot. But the single time scale is equals to 10 second therefore NACK and ACK are merged with each other in Figure 6.8. The de-compression failure state is separately described in Figure 6.9 with better where time scale is equals to 1 second.

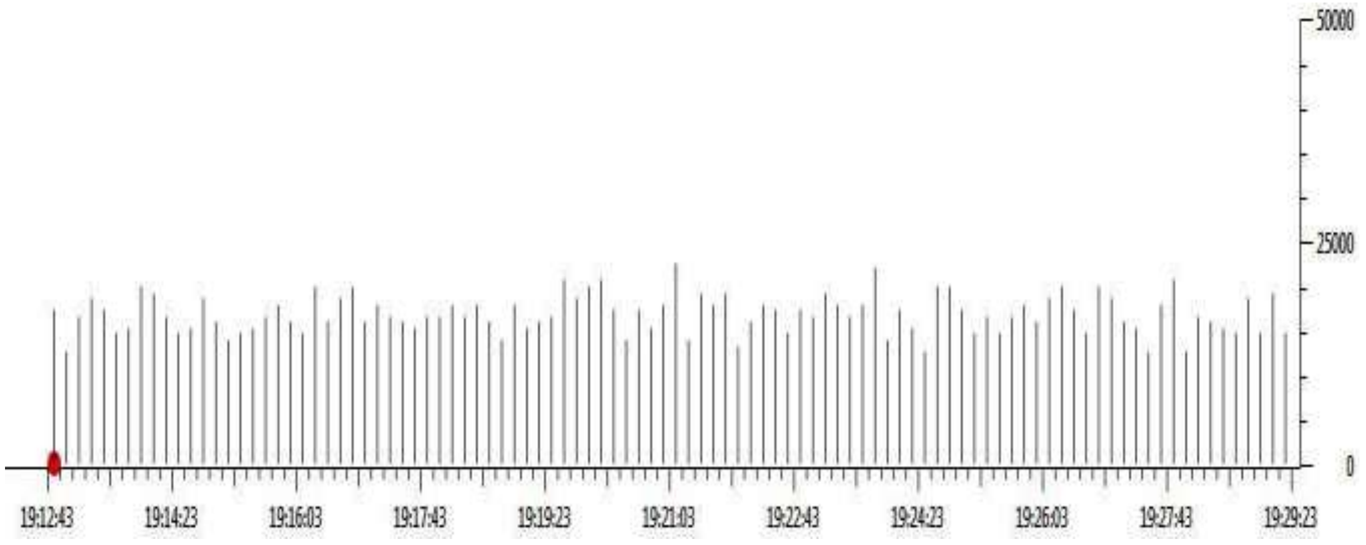


Figure 6.8: Graph independent packet loss pattern (forward channel)

Where,

X-axis is divided into ticks where one tick = 10 seconds. Similarly Y-axis represents bit /tick.

Figure 6.9 shows that ROHC operation works fine unless it is not interrupted by the independent packet loss at the starting period. Additionally, this observation is observed for all test results.

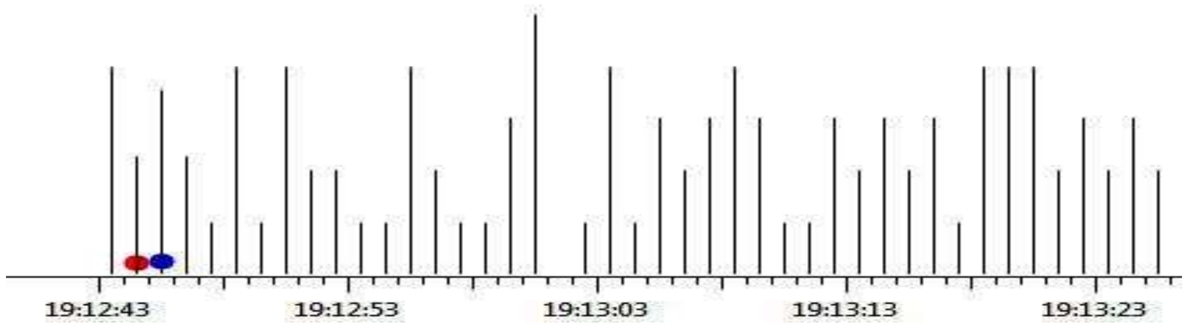


Figure 6.9: Decompression failure state over independent packet loss (forward channel emulation)

Where,
Red dot = NACK (19 bytes), Blue dot = ACK (20 bytes).
X-axis is divided into seconds.

Test Results for Independent Packet Loss (Forward Channel Emulation)

Table 3 represents the three different channel loss percentage where channel is emulated with 40%, 50% and 60% loss respectively. Furthermore, the measurements are based on **average** values of 10 tests results. It shows that there is no de-compression failures exist for 40% channel loss. In such condition number of received packets at de-compressor interfaces eth0 and tap0 are equal i.e. 2981. Therefore, the difference eth0 –tap0 = 0, meaning that no packet drop at TAP interface due to ROHC failure.

Independent Loss Percentage	TX-Packets (eth0)-Compressor	RX-Packets (eth0) de-compressor	RX-Packets (tap0)- de-compressor	Drop Packets (eth0)-de-compressor	Drop Packets (tap0)- de-compressor	Loss (%) eth0	Loss (%) Tap0
40%	5000	2981	2981	2019	2019	40.4	40.4
50%	5000	2486	2483	2514	2517	50.28	50.34
60%	5000	1972	1962	3028	3038	60.56	60.76

Table 3: Average value with independent packet loss pattern (forward channel emulation)

Where,

Compressor Node = HP-8460PX1, De-compressor Node = HP-8460PX2, **TX**: Transmitted Packets from PX1, **RX**: Received Packets at PX2 as described in Section 5.2.2, Figure 5.5.

Hence, we can conclude that ROHC is robust up to 40% channel loss conditions if the independent packet losses exist at the starting of communication setup over the imperfect channel.

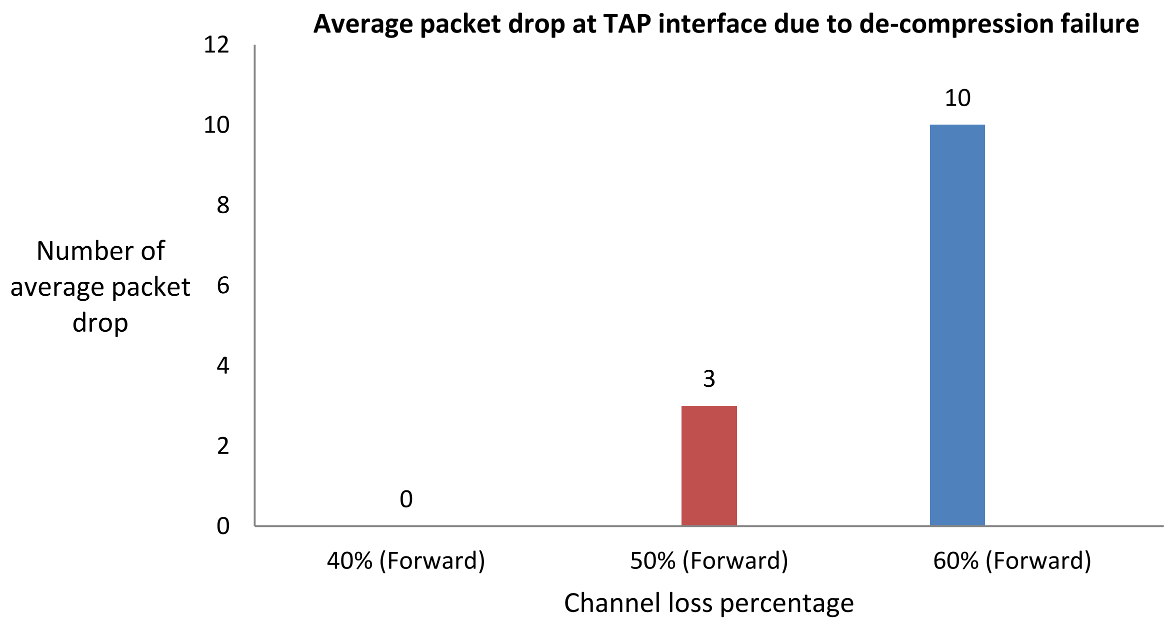


Figure 6.10: Average packet drop at TAP (tap0) interface (forward channel emulation)

However, de-compression failures exist for 50% and 60% channel loss conditions as shown in Figure 6.10. Therefore, there is an average packet drop difference between both interfaces (eth0 & tap0) at de-compressor end.

Average Packet drop difference (eth0 – tap0):

This difference represents the number of packets that are lost at TAP interface due to de-compression failure at de-compressor end. It shows the robustness of ROHC protocol under channel loss conditions.

Figure 6.10 represents the average packet drop difference values in a bar chart. It shows that 3 packets are lost for 50% and 10 packets for 60% channel loss conditions at TAP interface respectively.

Confidence Interval

A **confidence interval** (**CI**) is a type of interval estimate of a population parameter. It is used to indicate the reliability of an estimate [32].

The level of confidence of the confidence interval would indicate the probability that the confidence range captures. It does not describe any single sample. This value is represented by a percentage [32].

Confidence interval consists of a range of values (interval). In such case, we have lower and upper limits with an average value. We have already described the average values of 10 test results for forward channel emulation in Table 3. Figures 6.11, 6.12 & 6.13 represent the 95% confidence intervals of the received compressed packets at de-compressor Ethernet interface (eth0) for the independent loss percentage, as given in Table 3. It illustrates that we are 95% confident that the number of received compressed packets at the de-compressor input interface (eth0) should exist in the confidence range for any test case. Additionally, y-axis illustrates the received compressed packets at de-compressor Ethernet (eth0) interface.

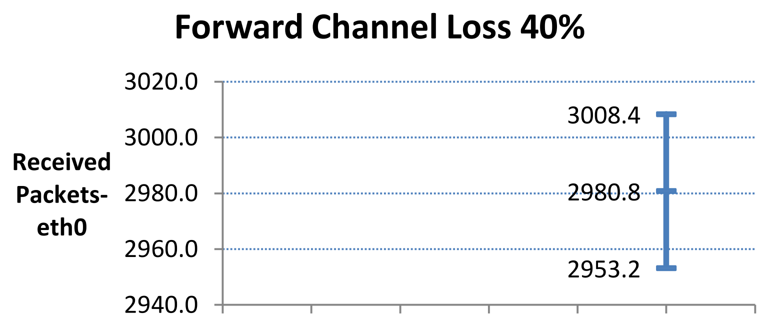


Figure 6.11: Confidence interval (95%)

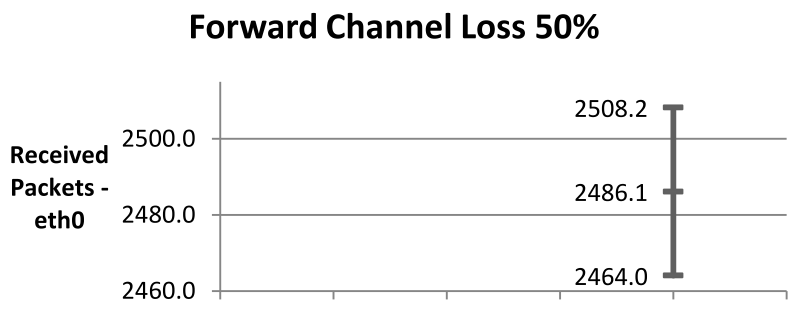


Figure 6.12: Confidence interval (95%)

For example, if we consider 50% forward channel loss condition then the confidence interval is from 2464 packets to 2509 packets. It means that it is 95% chance that the 10 test values reside in this confident range as shown in Figure 6.12.

In this way, we can estimate a test output value in a confident range for each sample. It is also observed that a few test values exist out of the confident range. For example, we have observed two test values of received compressed input (eth0) which are out of confident range for 50% packet loss i.e.2453 & 2565. The reference also support for the identical results.

After a sample is taken, the population parameter is either in the interval made or not [32].

This also might be Netem loss generation issue that is addressed in Section 7.2.2.

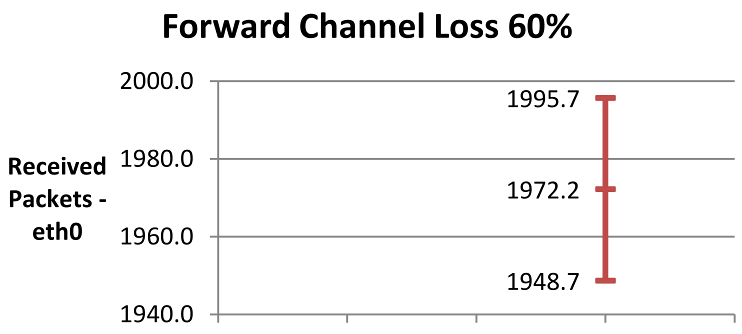


Figure 6.13: Confidence interval (95%)

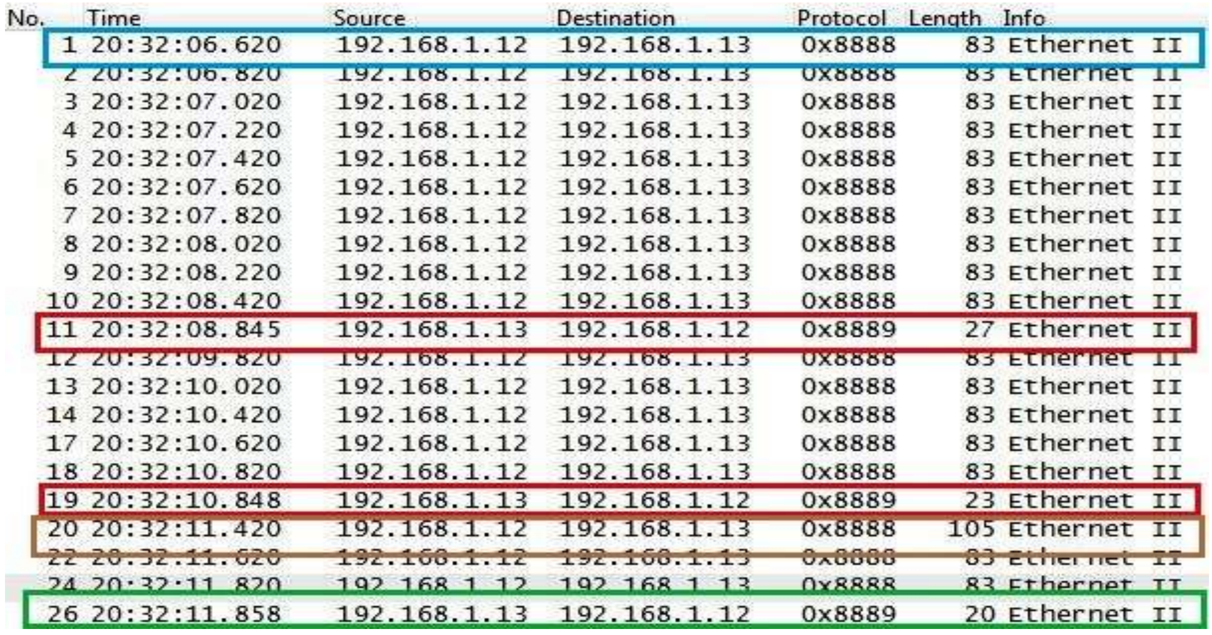
6.2.2 Forward & Reverse Channels Emulation

The two-way channel emulation is performed at the bridge Ethernet interfaces (eth0 & eth1) as already described in Section 5.2.2. We have performed the forward & reverse channels emulation in two ways which are followed.

Condition1:

In such condition, both forward and reverse channels are emulated with identical loss percentage (= 50%) as given in script no. 7 Appendix B. Figure 6.14 represents the trace analysis where the IR header packet (= 105 bytes) is lost due to forward channel loss at the starting period. The de-compressor receives the first packet (= 83 bytes) at time 20:32:06. The de-compressor cannot directly de-compress a compressed packet without reference or IR header. Therefore an error log message is generated by the ROHC library at de-compressor end in vyatta router.

```
20:32:06 vyatta rohc: ETH0 read_eth (rohc.c:779): Not able to decompress, drop packet. rohc_len -1.
20:32:10 vyatta rohc: last message repeated 14 times.
```



```
No.  Time          Source         Destination    Protocol  Length  Info
  1 20:32:06.620   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  2 20:32:06.820   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  3 20:32:07.020   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  4 20:32:07.220   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  5 20:32:07.420   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  6 20:32:07.620   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  7 20:32:07.820   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  8 20:32:08.020   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
  9 20:32:08.220   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 10 20:32:08.420   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 11 20:32:08.845   192.168.1.13   192.168.1.12   0x8889        27 Ethernet II
 12 20:32:09.820   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 13 20:32:10.020   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 14 20:32:10.420   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 17 20:32:10.620   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 18 20:32:10.820   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 19 20:32:10.848   192.168.1.13   192.168.1.12   0x8889        23 Ethernet II
 20 20:32:11.420   192.168.1.12   192.168.1.13   0x8888       105 Ethernet II
 22 20:32:11.620   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 24 20:32:11.820   192.168.1.12   192.168.1.13   0x8888        83 Ethernet II
 26 20:32:11.858   192.168.1.13   192.168.1.12   0x8889        20 Ethernet II
```

Figure 6.14: Trace analysis of forward & reverse channels independent packet loss pattern

The de-compressor node (IP address = 192.168.1.13) sends NACK towards compressor node (IP addresses = 192.168.1.12) two times as marked with red line in Figure 6.14. But the first NACK message is lost due to reverse channel loss. Afterwards, second NACK is received at the compressor node. Then, de-compressor received an IR header packet at time 20:32:11. At this moment, all received compressed packets are discarded during de-compression failure duration at Ethernet interface (eth0). Hence these packets are lost due to de-compression failure at TAP (tap0) interface.

Note: We have observed this behavior in two tests out of 10 tests. It depends on Netem queue which generates the loss pattern randomly.

Test Result Graph

IO graph is another way of representation which shows the NACK and ACK existence with respect to time for a test result in Figure 6.15. It only shows the de-compression failure part which happens at the starting period of the test duration. It is observed that there is no de-compression failure exists for the remaining period instead of starting in 10 test cases. Additionally, the maximum data rate is 2.56 Kbit per second.

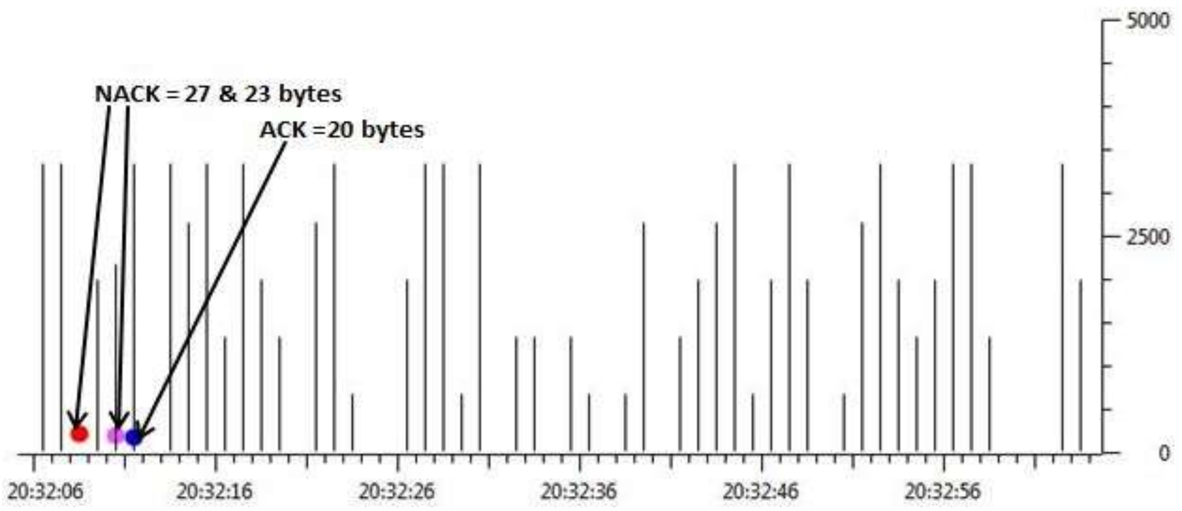


Figure 6.15: Graph independent packet loss pattern (forward & reverse channels emulation)

Where,
X-axis is divided into seconds. Similarly Y-axis represents bit /seconds.

Test Results for Independent Packet Loss (Forward & Reverse Channels Loss Emulation)

Tables 4 and 5 compare the packet drop in two emulation scenarios at de-compressor interfaces. It is observed that packet losses are greater with forward & reverse channel emulation than only forward channel emulation. This happens because NACK loss many times.

Forward channel loss	**Reverse channel loss**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at eth0 (ETHERNET-II)**	**Average Packet drop at De-compressor (PX2)RX.eth0 (ETHERNET-II)**	**Loss (%) eth0**
50%	0%	5000	2486	2514	50.28
50%	50%	5000	2408	2592	51.84

Table 4 : Average values at eth0 with independent packet loss (forward & reverse channels emulation)

Forward channel loss	**Reverse channel loss**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at tap0 (UDP)**	**Average Packet drop at De-compressor (PX2)RX at tap0 (UDP)**	**Loss (%) tap0**
50%	0%	5000	2483	2517	50.34
50%	50%	5000	2404	2596	51.92

Table 5: Average values at tap0 with independent packet loss (forward & reverse channels emulation)

For example, the loss percentage is increased from 51.84% to 51.92% at TAP (tap0) interface. It is not a big difference but it exists.

Figure 6.16 is further representation of Table 4 & 5. It shows the average packet drop number during de-compression failures for both scenarios as explained in Table 5.

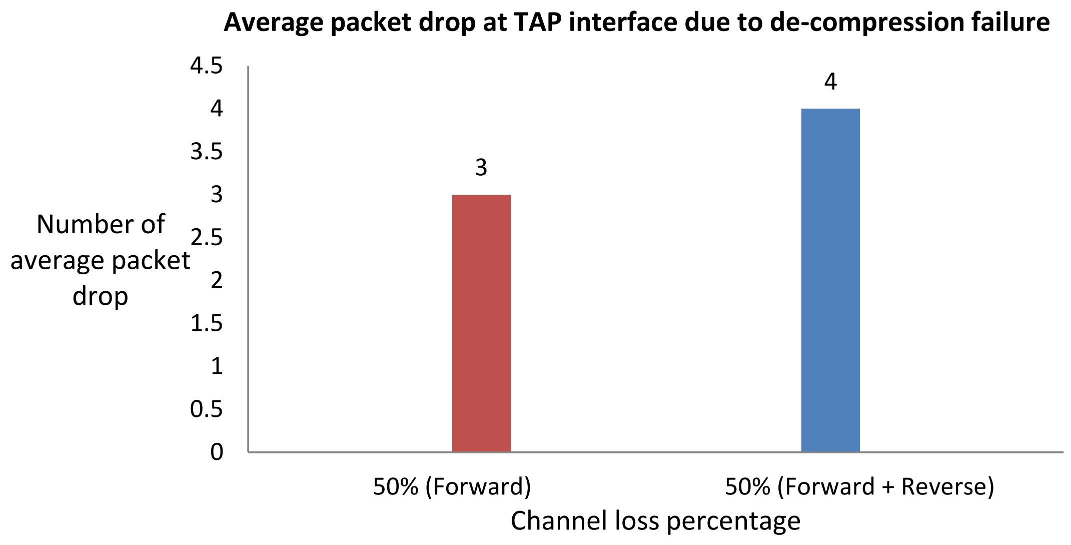


Figure 6.16: Average packet drop at TAP (tap0) interface with forward & reverse channels emulation

It is shown that number of packet loss at de-compressor end is greater for forward & reverse channel emulation case. The bar chart shows that 1 additional packet is dropped as compared to forward channel emulation.

Confidence Interval

Figure 6.17 represents the 95% confidence interval of the received compressed packets at de-compressor Ethernet interface (eth0) for forward & reverse channels loss emulation, as given in Table 4.

It illustrates the confidence range is from 2351 to 2465 packets with an average of 2408. Furthermore, we have observed three test values of received compressed input at de-compressor end (interface eth0) which are out of confident range .i.e.2294, 2473 & 2485.

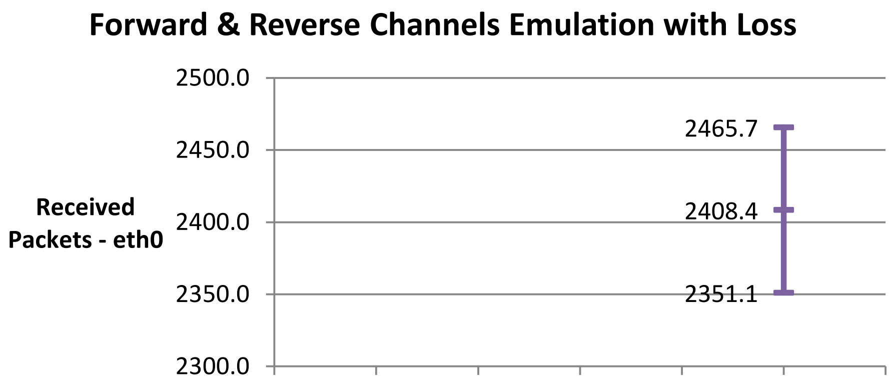


Figure 6.17: Confidence interval (95%)

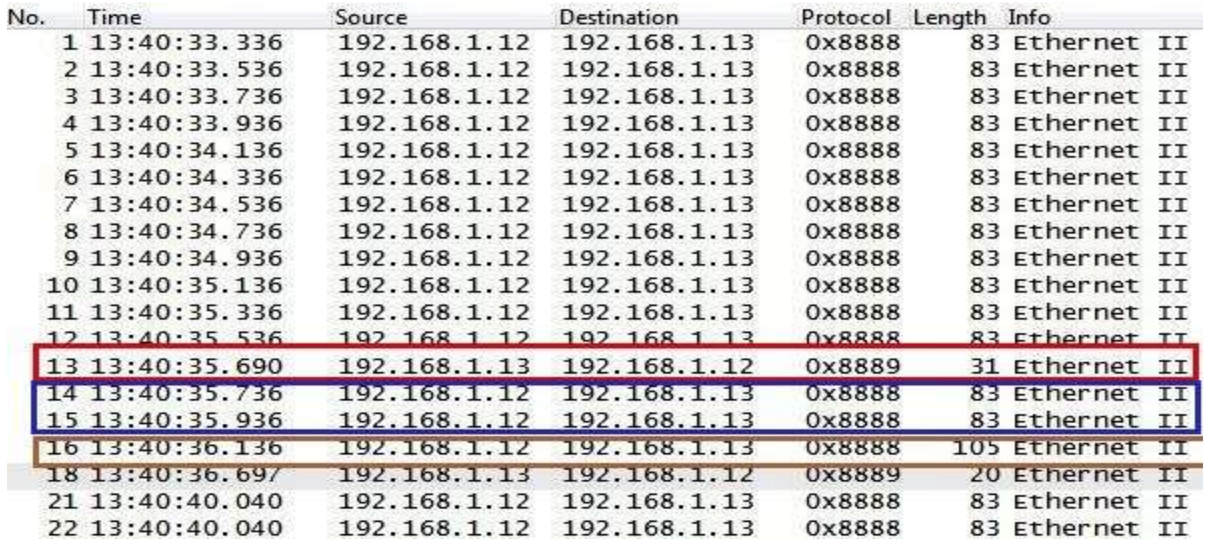
Y-axis illustrates the received compressed packets at de-compressor Ethernet (eth0) interface in Figure 6.17.

Condition2:

We are also interested to use ROHC protocol over satcom satellite channel for one hop communication. A satcom communication link has RTT (round trip times) in the range of 100 to 200 ms [33]. Therefore, it is decided to test reverse channel with delay emulation. We have tested the ROHC with a reverse channel delay of 100 ms, 200 ms and 250 ms.

In this condition, Forward channel is emulated with loss percentage (50%) and reverse channel with delay (250 ms) as given in script no.8 Appendix B. In such condition, the number of packet loss at TAP interface is higher than forward channel loss scenario in de-compression failure state.

De-compressor sends the NACK (31 bytes) towards compressor node at time 13:40:35 second as marked with red line in Figure 6.18. NACK is delayed due to reverse channel emulation therefore it arrived late at compressor node (IP address= 192.168.1.12).



No.	Time	Source	Destination	Protocol	Length	Info
1	13:40:33.336	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
2	13:40:33.536	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3	13:40:33.736	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
4	13:40:33.936	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
5	13:40:34.136	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
6	13:40:34.336	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
7	13:40:34.536	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
8	13:40:34.736	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
9	13:40:34.936	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
10	13:40:35.136	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
11	13:40:35.336	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
12	13:40:35.536	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
13	13:40:35.690	192.168.1.13	192.168.1.12	0x8889	31	Ethernet II
14	13:40:35.736	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
15	13:40:35.936	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
16	13:40:36.136	192.168.1.12	192.168.1.13	0x8888	105	Ethernet II
18	13:40:36.697	192.168.1.13	192.168.1.12	0x8889	20	Ethernet II
21	13:40:40.040	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
22	13:40:40.040	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II

Figure 6.18: Trace analysis of forward channel independent loss & reverse channel delay

At the same moment, compressor is unaware from the de-compression failure state and it continues send two compressed header packets (= 83 bytes) at time 13:40:35 second. Then, these compressed packets are discarded at the de-compressor Ethernet interface (eth0) as marked with blue line in Figure 6.18. Therefore, these packets are considered as loss at TAP (tap0) interface due to de-compression failure. This behavior leads to higher packet loss at de-compressor TAP (tap0) interface than forward channel emulation scenario due to reverse channel packet delay.

Test Result Graph

Figure 6.19 demonstrates the graphical representation of the de-compression failure event which starts at time 13:40:33 seconds in the trace analysis Figure 6.18. Figure 6.18 illustrates that two additional compressed packets (= 83 byte for each) are lost after NACK at de-compressor end due to reverse channel delay.

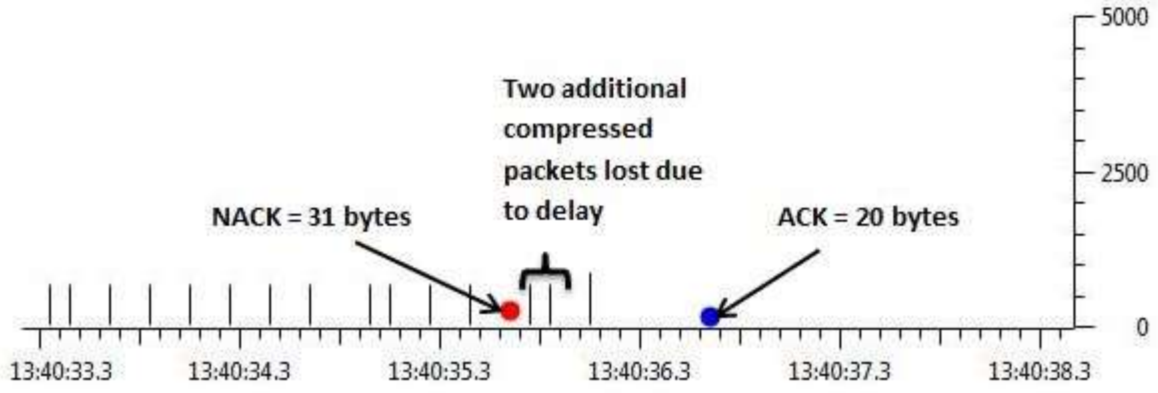


Figure 6.19: Graph independent forward channel packet loss & reverse channel delay emulation

Where,

X-axis is divided into desi (0.1) seconds. Similarly Y-axis represents bit / desi-second.

Test Results for Forward Channel Independent loss & Reverse Channel Delay Emulation

Table 6 is the comparison of forward channel emulation and forward & reverse channel emulation where reverse channel is emulated with packet delay. The measurements are based on average value of 10 tests where each test consists of 1000 seconds.

In this way, we have compared the loss percentage at de-compressor Ethernet interface (eth0). It is shown that loss percentage with 250 ms delay is higher than zero second delay in reverse channel condition (condition 1). It shows that overall loss is 50.28% without delay in reverse channel. But it exceeds to 51.18% for 250 ms delay in reverse channel due to NACK reaches at compressor node with delay.

Forward channel loss	**Reverse channel delay**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at eth0 (ETHERNET-II)**	**Average Packet drop at De-compressor (PX2)RX.eth0 (ETHERNET-II)**	**Loss (%) eth0**
50%	0	5000	2486	2514	50.28
50%	250 ms	5000	2441	2559	51.18

Table 6: Average values at eth0 interface with forward channel loss & reverse channel delay emulation

Table 7 is the loss percentage comparison at de-compressor end for TAP interface (tap0). The loss percentage is 50.34 % without reverse channel delay which rises up to 51.26% for 250 ms reverse channel delay case.

Forward channel loss	**Reverse channel loss**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at tap0 (UDP)**	**Average Packet drop at De-compressor (PX2)RX at tap0 (UDP)**	**Loss (%) tap0**
50%	0	5000	2483	2517	50.34
50%	250 ms	5000	2437	2563	51.26

Table 7: Average values at tap0 interface with forward channel loss & reverse channel delay emulation

The average packet drop is 3 with zero second reverse channel delay. Similarly, the average packet drop is exceeded to 4 with a 250 ms reverse channel delay with the equal 50% forward channel loss.

From the observations of Sections 6.2.1 & 6.2.2, we can conclude that ROHC is more robust for independent packet loss over channel because de-compression failure starts after 40% channel loss.

Confidence Interval

Figure 6.20 represents the 95% confidence interval of the received compressed packets at de-compressor Ethernet interface (eth0) for the independent forward & reverse channels loss percentage (50%), as given in Table 6. Y-axis illustrates the received compressed packets at de-compressor Ethernet (eth0) interface.

It illustrates the confidence range is from approximately 2328 to 2553 packets with an average of 2441. Furthermore, we have observed three test values of received compressed input at de-compressor end (interface eth0) which are out of confident range .i.e. 2276, 2318, & 2687.

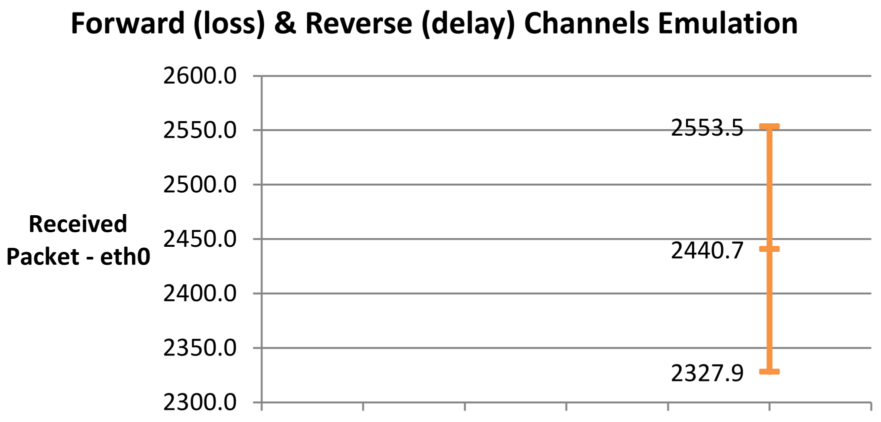


Figure 6.20: Confidence interval (95%)

6.3 Burst Packet Loss Pattern

The burst packet loss emulation is performed in terms of consecutive packet loss which has longer periods of high loss density. A longer burst packet loss pattern can affect the ROHC operation because the de-compressor lost the interpretation interval or reference value (v_ref) due of lack of input for certain time duration as explained in Section 3.2.4.

Burst pattern emulation generates the de-compression failure state in two ways. Firstly, in the beginning of communication setup that is identical to Section 6.2. Therefore, it is not essential to repeat it again. Secondly, de-compression failures or de-synchronization during normal ROHC operation which will be explained in this section.

Similarly, we have performed 10 tests for each channel emulation scenario. The time duration of each test is 2000 seconds. MGEN generated the 10000 packets for one test duration at the compressor end as given in Appendix C.

Burst is defined in terms of mean burst length [27]. We have performed several tests with various burst length such as 5, 10, 20, 30 & 35 seconds. Furthermore, we have found decompression-failure happens only with burst duration of more than 30 seconds. All the experimental results are given in Section 6.3.1 & 6.3.2 where a burst length is equal to 35 seconds.

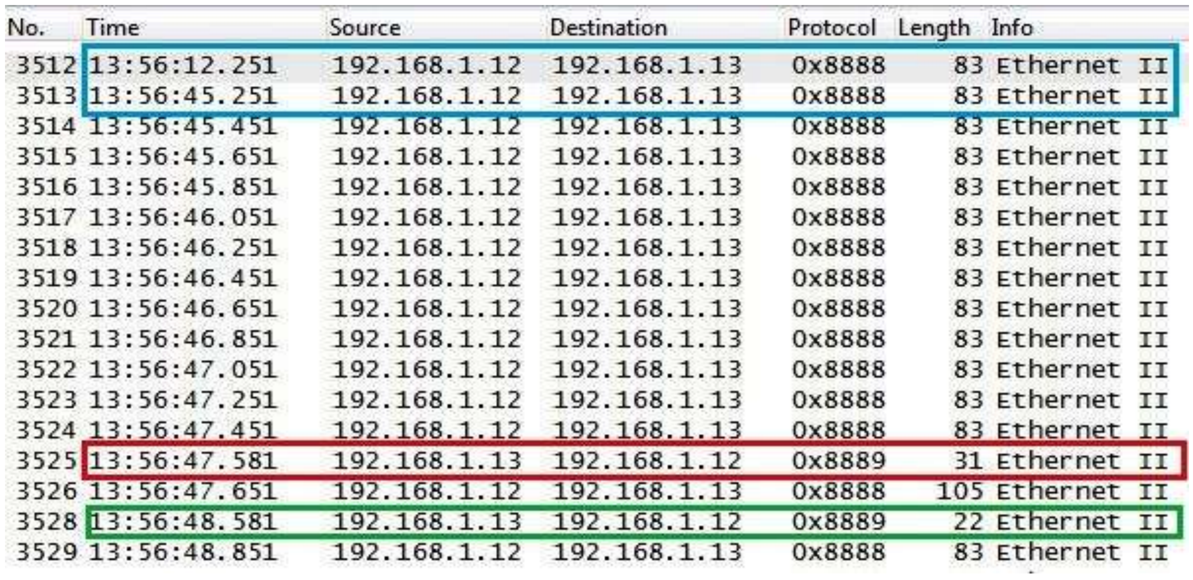
MGEN generates the packet at a rate = 5 packets/second at compressor node as given in Appendix C. Therefore, maximum 2.56 Kbit are transmitted in 1 second. We have evaluated the performance in two conditions which are followed.

6.3.1 Forward Channel Emulation

In such condition, forwards channel is only emulated for packet loss with a mean burst length of 35 seconds as given in scrip no. 3 in Appendix B where bridge PX3 Ethernet interface (eth1) is emulated for this purpose.

The trace analysis is described for 50% channel loss in Figure 6.21 where compressed packets are captured at the de-compressor (eth0) Ethernet interface. It shows a consecutive packet loss for time duration of 33 seconds where the last successfully decompressed packet has received at time 13:56:12 and the current compressed packet is received at time 13:56:45. At this stage, the de-compressor (IP address = 192.168.1.13) attempts to de-compress the compressed header but the CRC fails. The reason is interpretation interval time duration which is already explained in Sections 5.2.1 & 5.3.3. Moreover, the RFC describes this situation such as.

“In the Full Context state: When the CRC check of k_1 out of the last n_1 decompressed packets have failed, context damage SHOULD be assumed and a NACK SHOULD be sent in O-mode. The de-compressor moves to the static context state and discards all packets until an update (IR) which passes the CRC check is received” (RFC, 3095, Page 59).



No.	Time	Source	Destination	Protocol	Length	Info
3512	13:56:12.251	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3513	13:56:45.251	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3514	13:56:45.451	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3515	13:56:45.651	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3516	13:56:45.851	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3517	13:56:46.051	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3518	13:56:46.251	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3519	13:56:46.451	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3520	13:56:46.651	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3521	13:56:46.851	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3522	13:56:47.051	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3523	13:56:47.251	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3524	13:56:47.451	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II
3525	13:56:47.581	192.168.1.13	192.168.1.12	0x8889	31	Ethernet II
3526	13:56:47.651	192.168.1.12	192.168.1.13	0x8888	105	Ethernet II
3528	13:56:48.581	192.168.1.13	192.168.1.12	0x8889	22	Ethernet II
3529	13:56:48.851	192.168.1.12	192.168.1.13	0x8888	83	Ethernet II

Figure 6.21: Trace analysis of forward channel burst packet loss pattern

The de-compressor received the first packet after consecutive packet loss at time 13:56:12; then it tries to de-compress the header and failed. The ROHC LIB continues generate a log message in Vyatta router log file during the de-compression failure duration.

13:56:12 vyatta rohc: ETH0 read_eth (rohc.c:779): Not able to decompress, drop packet. rohc_len -1
13:56:47 vyatta rohc: last message repeated 11 times

The, the de-compressor tries additional de-compression attempts but failed. The RFC 3095 explain such condition as follow.

"When the mismatch is caused by prolonged loss, the de-compressor might attempt additional de-compression attempts" (RFC, 3095, Page 59).

The consecutive / burst loss pattern is also known as wraparound as reference with RFC.

"A de-compression failure event occurs due to consecutive packet loss, also known as wraparound" (RFC, 3095, Page 58, 59).

The de-compressor continues attempt for approximately 2 seconds. Finally, it sends NACK (= 31 bytes) towards compressor (IP address = 192.168.1.12) at time 13:56:47 as shown in Figure 6.21. Afterwards, the compressor updates a new reference header in its context table. It sends IR header (= 105 byte) towards de-compressor at the same time. The de-compressor also updates its context table with new reference header for next de-compression attempt and replies with an ACK (= 22 byte). Then, the compressor starts sends compressed header in SO state again.

All packets are discarded at Ethernet (eth0) interface which were received during de-compression failure duration at the de-compressor end. These packets are considered as packet drop at TAP (tap0) interface due to header compression failure at de-compressor end.

Test Result Graph

Wireshark IO graph represents the compressed data traffic for 50% channel loss test case in Figure 6.22. It also constitutes de-compression failure states with red marked.

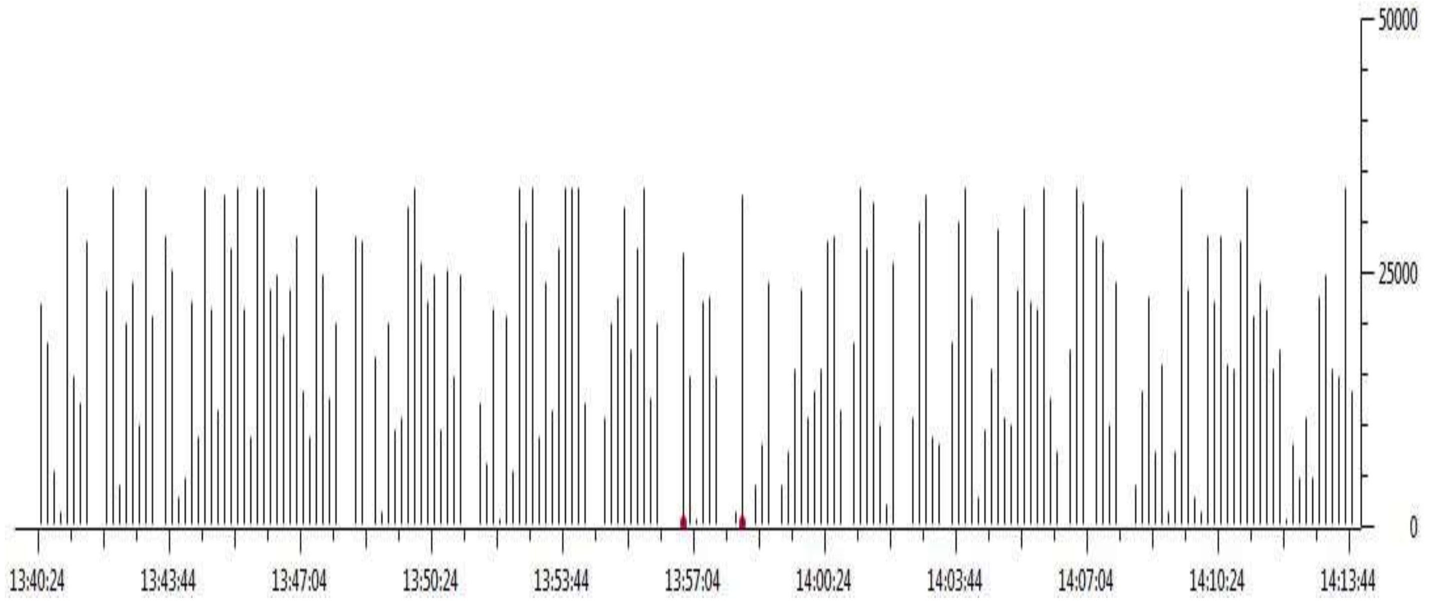


Figure 6.22: Graph burst packet loss pattern (forward channel)

Where,
X-axis is divided in to ticks where one tick = 10 seconds. Similarly Y-axis represents bit /tick.

The de-compression failure state is independently described in Figure 6.23 with better resolution. Figure 6.23 is the identical representation of trace analysis in Figure 6.21. It shows the packet loss duration of 33 seconds. Additionally, NACK and ACK are represented with red and blue dots respectively.

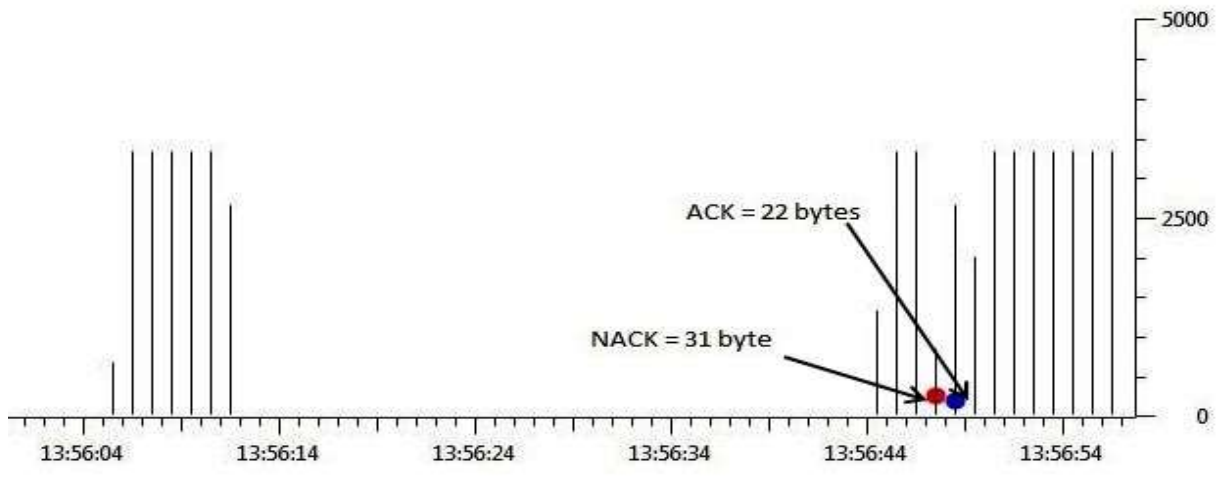


Figure 6.23: Decompression failure state over burst packet loss (forward channel emulation)

Where, X-axis is divided in to seconds. Similarly Y-axis represents bit /second.

Test Results for Burst Packet Loss (Forward Channel Emulation)

Table 8 represents the five different channel loss percentages where channel is emulated with 15%, 20%, 30%, 40% and 50% loss with a mean burst length (35 second). Furthermore, the measurements are based on average values of 10 tests for each channel loss value. It shows that there is no de-compression failures exist for 15% channel loss. In such condition number of received packets at interface eth0 and tap0 are equal i.e. 1377 packets.

Packet Loss Percentage	TX-Packets (eth0)-Compressor	RX-Packets (eth0) de-compressor	RX-Packets (tap0)- de-compressor	Drop Packets (eth0)-de-compressor	Drop Packets (tap0)- de-compressor	Loss (%) eth0	Loss (%) Tap0
15%, burst=35	10000	8623	8623	1377	1377	13.8	13.8
20%, burst=35	10000	7764	7757	2236	2243	22.36	22.43
30%, burst=35	10000	7164	7151	2836	2849	28.36	28.49
40%, burst=35	10000	5864	5843	4136	4157	41.36	41.57
50%, burst=35	10000	5143	5109	4857	4891	48.57	48.91

Table 8: Average values with burst loss pattern (forward channel emulation)

De-compression failure started from 20% channel packet loss. We can observe that ROHC is robust up to 15% channel loss with burst duration more than 30 seconds. Additionally, the burst duration has 100% loss density. Furthermore, average packet drop at TAP (tap0) interface during failure instant increases with channel loss percentage at the de-compressor end as shown in Figure 6.24. We have maximum drop of 34 packets at TAP (tap0) interface with 50% channel loss.

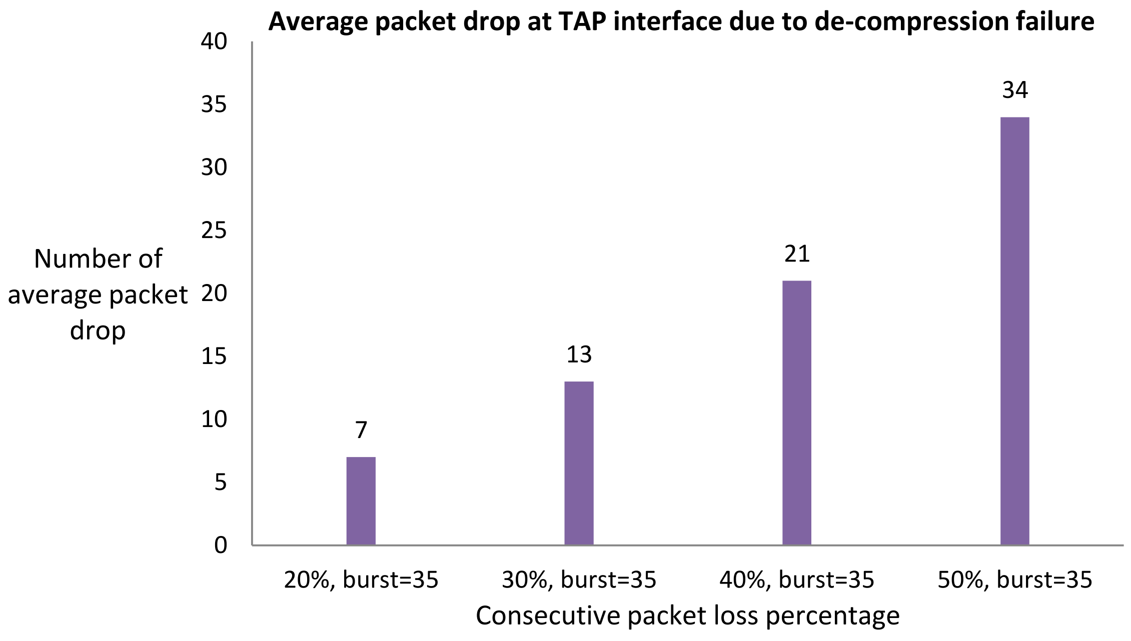


Figure 6.24: Average packet drop at TAP (tap0) interface (forward channel emulation)

Confidence Interval

Figure 6.25 illustrate the 95% confidence interval of the received compressed packets at de-compressor Ethernet interface (eth0) for the forward channels emulation with burst packet loss pattern with loss percentages, as given in Table 8. X-axis represents the loss percentage whereas y-axis illustrates the received compressed packets at de-compressor Ethernet (eth0) interface.

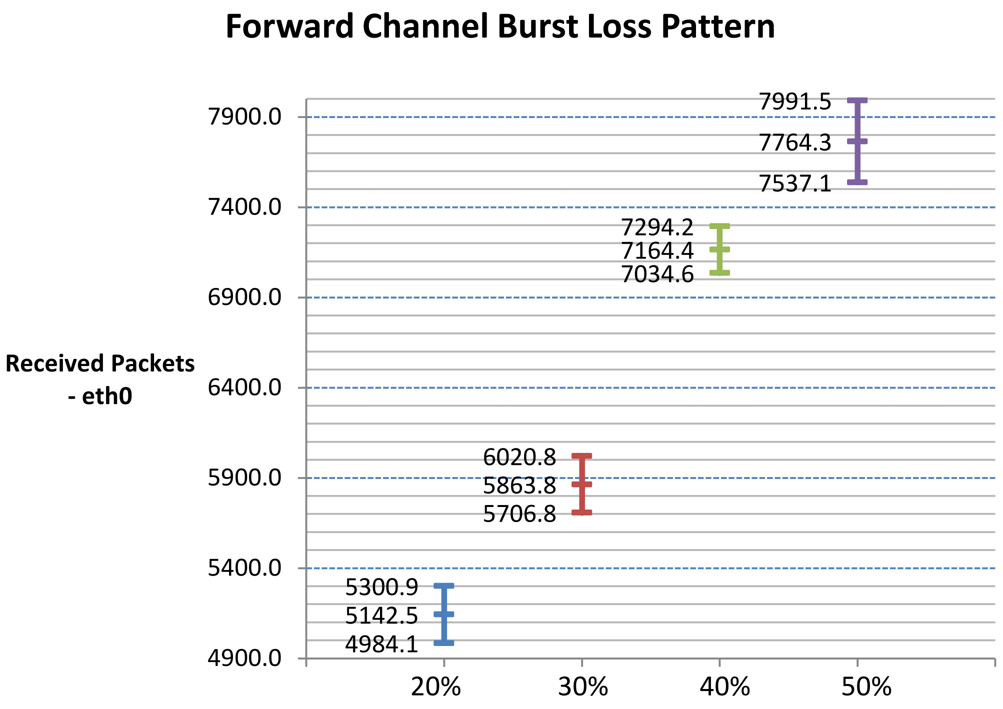


Figure 6.25: Confidence interval (95%)

It illustrates the confidence range for each packet loss percentage with an average values.

6.3.2 Forward & Reverse Channels Emulation

Forward & reverse channels burst loss emulation is performed with the bridge Ethernet interfaces (eth0 & eth1) as already described in Section 5.2.2. It is also performed in two different ways for burst loss pattern. Bridge interfaces (eth0 & eth1) are emulated for this purpose as described in Section 5.2.2. The emulation set-ups are followed here.

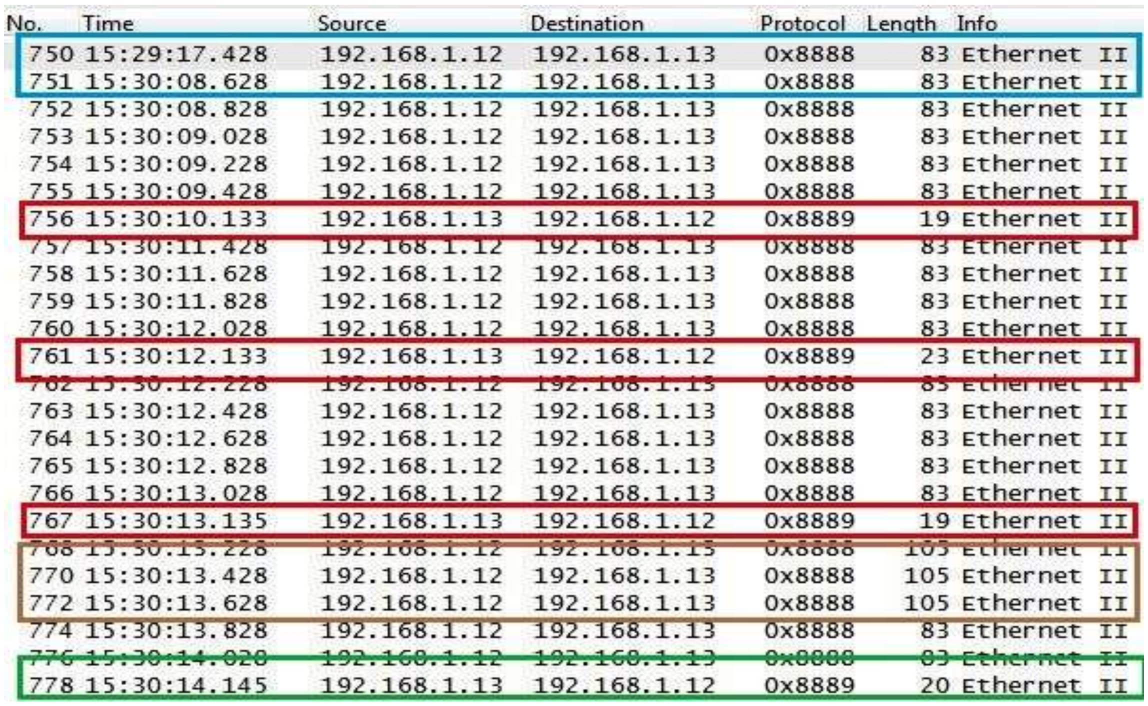
Condition 1:

Forward & reverse channels are emulated with burst loss patterns. The emulation details are given in Appendix B script 4 where,

Forward channel emulation pattern: Loss = 50%, burst = 35 second
Reverse channel emulation pattern: Loss = 50%

Figure 6.26 describes the de-compression failure instant. The de-compressor (192.168.1.13) receives the first compressed packet after consecutive packet loss of 50 seconds in FC state at time 15:30:08 as marked with blue rectangular in Figure 6.26. Afterwards, the de-compressor tries de-compression attempts but failed. It sends two NACK 19 bytes and 23 bytes towards de-compressor at time 15:30:10 and 15:30:12. But these are failed due to feedback channel packet loss. The de-compressor tries to de-compress packet after two NACKs but it failed every time. De-compressors third NACK (at time = 15:30:13) is received by the compressor (192.168.1.12) which sends an IR header for the recovery.

Note: The burst packet loss is 50 seconds which is combination of two burst. It is depended on Netem error generator. It might be one 35 sec burst and other 15 seconds or an equal size bursts.



```
No.  Time           Source         Destination    Protocol  Length  Info
750  15:29:17.428   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
751  15:30:08.628   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
752  15:30:08.828   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
753  15:30:09.028   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
754  15:30:09.228   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
755  15:30:09.428   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
756  15:30:10.133   192.168.1.13   192.168.1.12   0x8889       19  Ethernet II
757  15:30:11.428   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
758  15:30:11.628   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
759  15:30:11.828   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
760  15:30:12.028   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
761  15:30:12.133   192.168.1.13   192.168.1.12   0x8889       23  Ethernet II
762  15:30:12.228   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
763  15:30:12.428   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
764  15:30:12.628   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
765  15:30:12.828   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
766  15:30:13.028   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
767  15:30:13.135   192.168.1.13   192.168.1.12   0x8889       19  Ethernet II
768  15:30:13.228   192.168.1.12   192.168.1.13   0x8888      105  Ethernet II
770  15:30:13.428   192.168.1.12   192.168.1.13   0x8888      105  Ethernet II
772  15:30:13.628   192.168.1.12   192.168.1.13   0x8888      105  Ethernet II
774  15:30:13.828   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
776  15:30:14.028   192.168.1.12   192.168.1.13   0x8888       83  Ethernet II
778  15:30:14.145   192.168.1.13   192.168.1.12   0x8889       20  Ethernet II
```

Figure 6.26: Trace analysis of forward & reverse channel burst packet loss pattern

All compressed packets are discarded at de-compressor Ethernet (eth0) interface which were received during de-compression failures instant. These packets are considered as packet drop due to header compression at TAP (tap0) interface. These number of packets loss are greater here than forward channel emulation in Section 6.3.1 for 50% channel loss percentage.

Test Result Graph

The trace analysis graph shows the decompression failure event in Figure 6.27.

It shows the two NACK which are request for error recovery generated by de-compressor and a feedback message for the acknowledgement. The maximum data rate is 2.56kbit/second that is illustrated by y-axis in Figure 6.27.

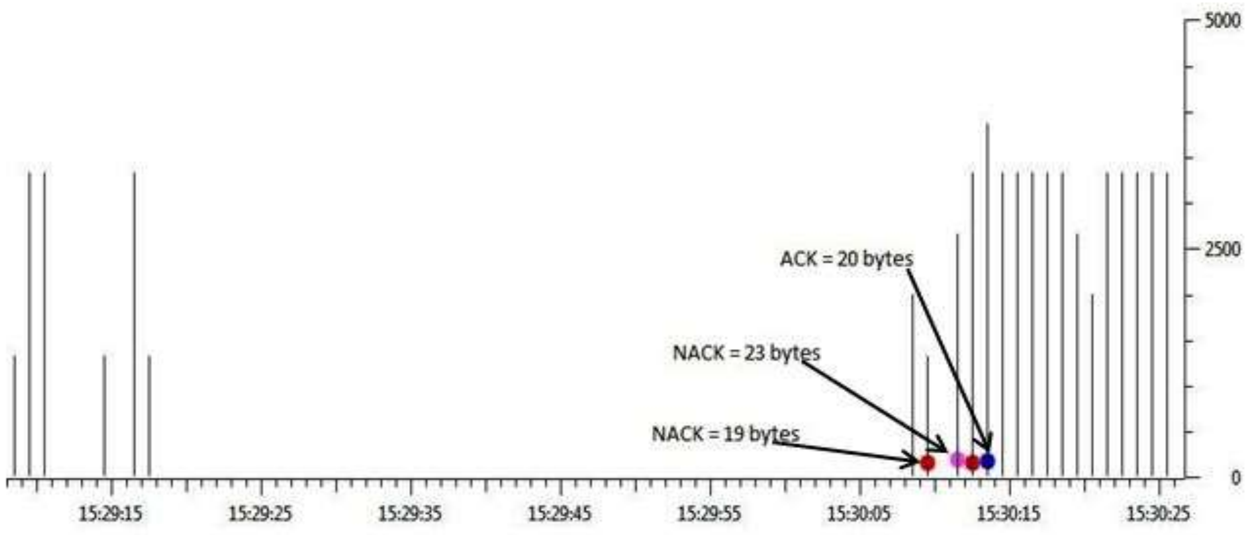


Figure 6.27: Decompression failure state for burst packet loss (forward & reverse channel emulation)

Where,
X-axis is divided in to seconds. Similarly Y-axis represents bit /second.

Test Results for Burst Packet Loss (Forward & Reverse Channels Loss Emulation)

Tables 9 and 10 compare the packet drop in two emulation scenarios at de-compressor interfaces i.e., forward channel emulation and forward & reverse channels emulation.

Forward channel loss	**Reverse channel loss**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at eth0 (ETHERNET-II)**	**Average Packet drop at De-compressor (PX2)RX.eth0 (ETHERNET-II)**	**Loss (%) eth0**
50%, burst = 35	0%	10000	5143	4857	48.57
50%, burst = 35	50%	10000	5090	4910	49.10

Table 9 : Average values at eth0 interface for burst loss pattern (forward & reverse channel emulation)

Forward channel loss	**Reverse channel loss**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at tap0 (UDP)**	**Average Packet drop at De-compressor (PX2)RX at tap0 (UDP)**	**Loss (%) tap0**
50%, burst = 35	0%	10000	5109	4891	48.91
50%, burst = 35	50%	10000	5045	4955	49.55

Table 10 : Average values at tap0 interface for burst loss pattern (forward & reverse channel emulation)

It is observed that packet losses are greater with forward & reverse channel emulation scenario at eth0 & tap0 interfaces than forward channel emulation. The reason is already explained that the loss of NACK many times due to reverse channel loss.

For example, the loss percentage increases from 49.10% to 49.55% at TAP (tap0) interface for two way channels emulation. Furthermore, it is observed that packet loss increases with a longer loss burst duration in the reverse channel.

Figure 6.28 is further representation of Tables 9 & 10. It compares the average number of packets drop during de-compression failures in both scenarios. The bar chart shows that 11 additional packets are dropped as compared to forward channel emulation. It is shown that ROHC performance effects with the loss percentage in forward and reverse channels.

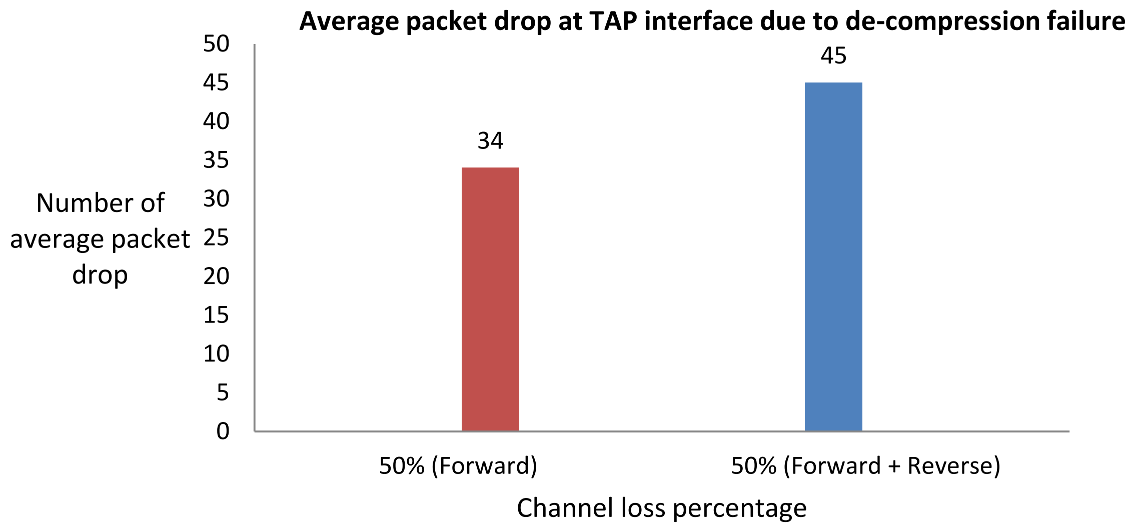


Figure 6.28: Average packet drop at TAP (tap0) interface (forward & reverse channels emulation)

Confidence Interval

Figure 6.29 illustrates the 95% confidence interval of the received compressed packets at de-compressor Ethernet interface (eth0) for forward & reverse channels loss emulation, as given in Table 9.

It illustrates the confidence range is from approximately 4908 to 5273 packets with an average of 5090. Furthermore, we have observed three test values of received compressed input at de-compressor end (interface eth0) which are out of confident range .i.e., 4708, 4849 & 5429.

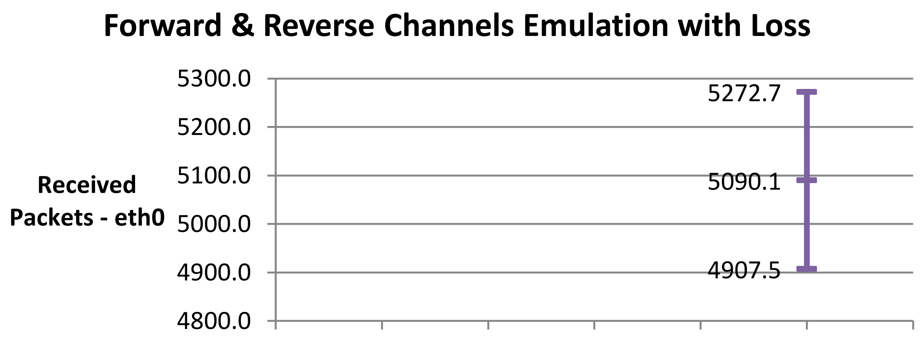


Figure 6.29: Confidence Interval (95%)

Condition 2:

In this condition, Forward channel is emulated with loss and reverse channel with delay. The emulation details are given in scriptno.5 Appendix B where,

Forward channel emulation pattern: Loss = 50%, burst = 35 second
Reverse channel emulation pattern: Delay = 300 ms

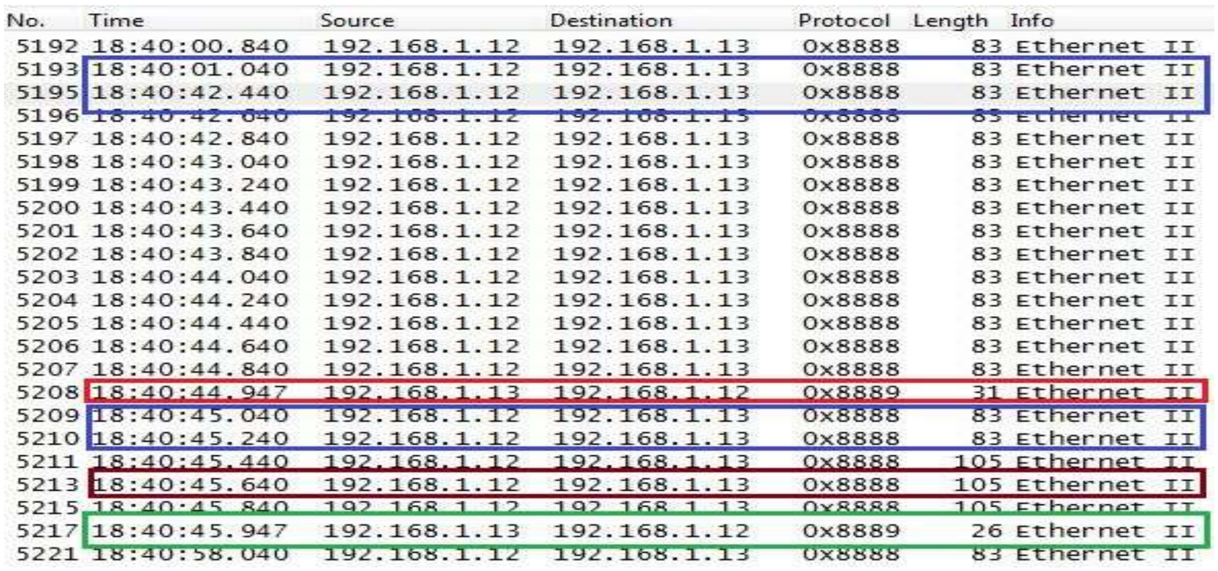
Note: There is not a special reason to increase 50ms delay than previous delay condition in Section 6.2.2. The purpose is to observe different behavior of ROHC devices (compressor & de-compressor) with 300 ms delay. But the ROHC devices operation is normally identical for delay at de-compression failures instant for both cases i.e., individual loss & burst loss.

The de-compression failure event is described with Figure 6.30 which starts at time 18:40:42 after a consecutive packet loss of 41 seconds.

Another important aspect of our test is to observe the effect of delay attribute during the de-compression failure state. The de-compressor (IP address = 192.168.1.13) sends NACK towards compressor (IP address = 192.168.1.12) node at time 18:40:44. NACK is delayed with a configured value of 300ms. At the same moment, compressor is unaware from de-compression failure therefore it sends two compressed header packets which can be seen after NACK as marked with blue rectangle in Figure 6.30.

In this way these two packets are also discarded at de-compressor Ethernet (eth0) interface. In a normal operation, de-compressor gets IR header after NACK without delay as marked with brown rectangle in Figure 6.20.

Hence, the number of packet drop will increased at de-compressor TAP (tap0) interface due to de-compression failure.



```
No.   Time            Source          Destination     Protocol Length Info
 5192 18:40:00.840    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5193 18:40:01.040    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5195 18:40:42.440    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5196 18:40:42.640    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5197 18:40:42.840    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5198 18:40:43.040    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5199 18:40:43.240    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5200 18:40:43.440    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5201 18:40:43.640    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5202 18:40:43.840    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5203 18:40:44.040    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5204 18:40:44.240    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5205 18:40:44.440    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5206 18:40:44.640    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5207 18:40:44.840    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5208 18:40:44.947    192.168.1.13    192.168.1.12    0x8889       31 Ethernet II
 5209 18:40:45.040    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5210 18:40:45.240    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
 5211 18:40:45.440    192.168.1.12    192.168.1.13    0x8888      105 Ethernet II
 5213 18:40:45.640    192.168.1.12    192.168.1.13    0x8888      105 Ethernet II
 5215 18:40:45.840    192.168.1.12    192.168.1.13    0x8888      105 Ethernet II
 5217 18:40:45.947    192.168.1.13    192.168.1.12    0x8889       26 Ethernet II
 5221 18:40:58.040    192.168.1.12    192.168.1.13    0x8888       83 Ethernet II
```

Figure 6.30: Trace analysis of forward channel burst loss pattern & reverse channel delay

Test Result Graph

Figure 6.31 demonstrates the graphical representation of the de-compression failure event which starts at time 18:40:01 seconds in the trace analysis Figure 6.30. Figure 6.31 illustrates that two additional compressed packets (= 83 byte for each) are lost after NACK at de-compressor end due to reverse channel delay.

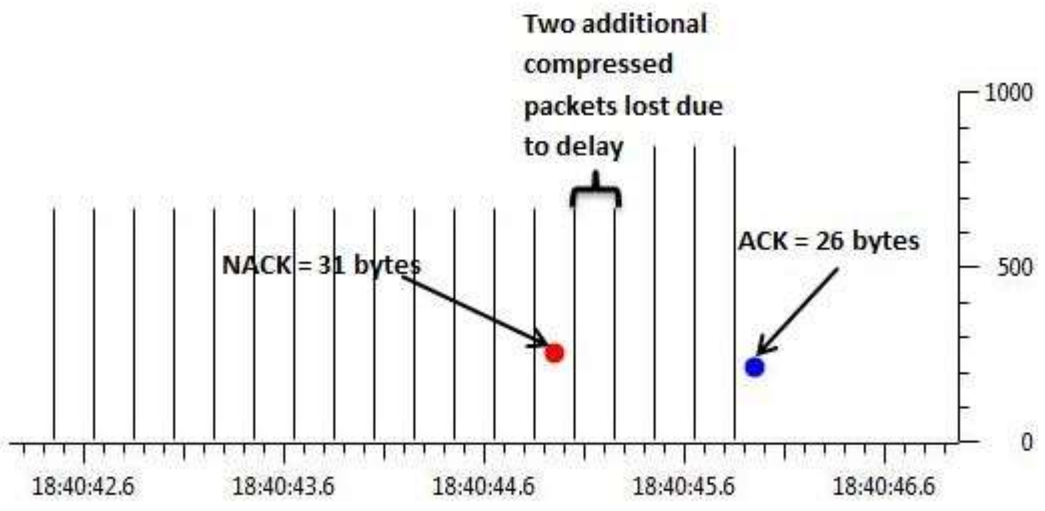


Figure 6.31: Graph forward channel burst packet loss & reverse channel delay emulation)

Where,

X-axis is divided into desi (0.1) seconds. Similarly Y-axis represents bit / desi-second.

Test Results for Forward Channel Burst Packet Loss & Reverse Channel Delay Emulation

Table 11 compares the loss percentage at Ethernet (eth0) interface for reverse channel emulation with zero and 300ms delay respectively. The forward channel has fixed loss percentage with a 35 seconds burst length for both reverse channel delay conditions.

Forward channel loss	Reverse channel delay	Compressor(PX1) TX at eth0 (ETHERNET-II)	De-compressor (PX2)RX at eth0 (ETHERNET-II)	Average Packet drop at De-compressor (PX2)RX.eth0 (ETHERNET-II)	Loss (%) eth0
50%, burst = 35	0	10000	5143	4857	48.57
50%, burst = 35	300 ms	10000	5004	4996	49.96

Table 11: Average values at eth0 interface with forward channel loss & reverse channel delay emulation

Forward channel loss	**Reverse channel loss**	**Compressor(PX1) TX at eth0 (ETHERNET-II)**	**De-compressor (PX2)RX at tap0 (UDP)**	**Average Packet drop at De-compressor (PX2)RX at tap0 (UDP)**	**Loss (%) tap0**
50%, burst = 35	0	10000	5109	4891	48.91
50%, burst =35	300 ms	10000	4967	5033	50.33

Table 12: Average values at tap0 interface with forward channel loss & reverse channel delay emulation

It is found that loss percentage is higher for two way channel emulation where reverse channel is emulated with 300ms delay than forward channel emulation (Section 6.3.1).

Table 12 represents that the packet drop or loss percentage at de-compressor TAP (tap0) interface is higher than forward channel emulation scenario.

Figure 6.32 represents that packet loss with reverse channel delay emulation is higher than forward channel emulation. The average packet drop increased at TAP (tap0) interface which equals to 37 packets.

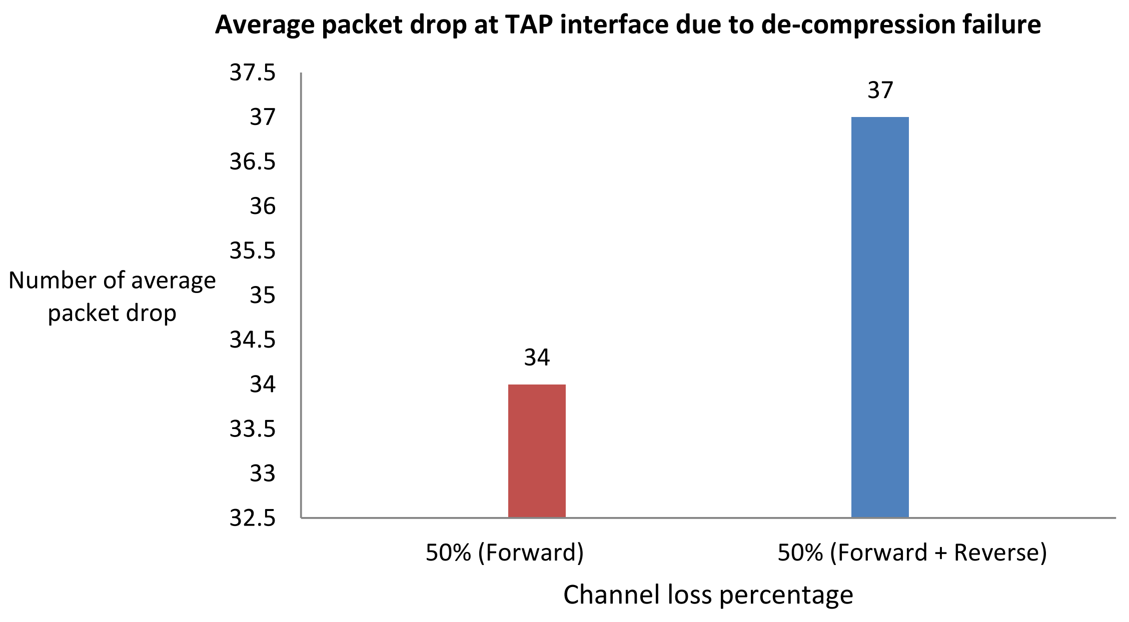


Figure 6.32: Average packet drop at TAP (tap0) interface

Confidence Interval

Figure 6.33 represents the 95% confidence interval. Y-axis illustrates the received compressed packets at de-compressor Ethernet (eth0) interface as given in Table 11.

We have observed three test values of received compressed input at de-compressor end (interface eth0) which are out of confident range .i.e. 4468, 4784, & 5458.

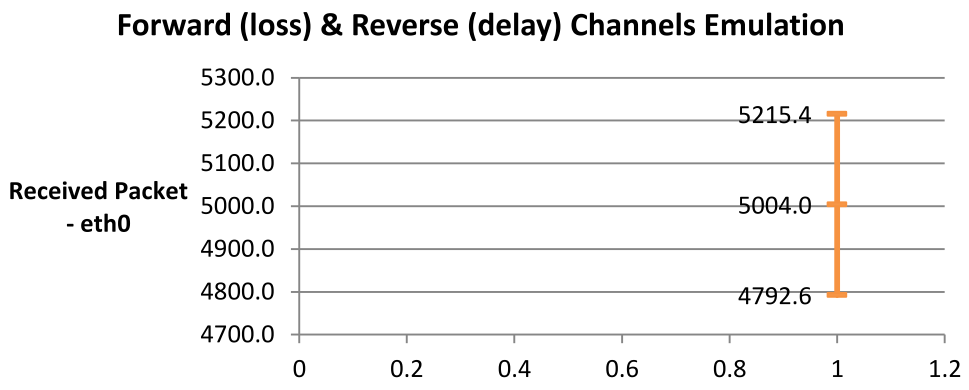


Figure 6.33: Confidence Interval (95%)

Chapter 7 Discussions

The thesis work started with the theoretical knowledge of the ROHC protocol. RFCs 3095, 5225 and 5795 are studied carefully for this purpose. Furthermore, I have studied the related research work and understand the different header compression approaches as described in Section 2.2.

The core task of the thesis work is to build a testbed scenario where we can evaluate the ROHC protocol performance in terms of its robustness against link failures. Vyatta Linux routers were already configured with the ROHC Library software. Netem is considered for the packet emulation testbed scenarios as described in Section 5.2. This approach reflects the ROHC operation over an imperfect channel. Netem is an open source tool which modifies the Linux traffic control *TC* with respect to configured parameters. We have faced certain limitations with Netem queue discipline and problems with testbed design for the compressed traffic emulation through this thesis work. This took extra time. Section 7.1 describes the testbed design problems. On the other hand, Section 7.2 describes the Netem limitations.

7.1 Testbed Design Problems

The testbed design problems are followed.

7.1.1 Netem Egress Queue

At the first step, Netem egress queue is configured at the compressor node for traffic emulation with the following *tc* command.

vyatta@vyatta:~$sudo *tc* qdisc add dev eth0 root netem LOSS

Where,

LOSS: = loss {random PERCENT (CORRELATION)}, e.g. loss 10%

In such condition, the htb queue is replaced with Netem queue in the Linux traffic control. Hence, the Netem queue behaves like packet first-input first-output (pFIFO) queue. In this way, we are able to emulate the uncompressed traffic at the compressor end but not the compressed one.

There are following Linux traffic queue details which represent the htb (root) queue discipline before Netem queue integration.

qdisc htb1: dev eth0 root refcnt 2 r2q 10 default 16 direct_packets_stat 0
qdisc rohc 11: dev eth0 parent 1:11

In this queue (htb) hierarchy, the filter (TOS) packets are sent towards rohc sub-queue. The rohc 11: sub-queue further passes the packets stream towards ROHC library for compression.

Afterwards, the addition of Netem queue gives the following Linux queue discipline here.

qdisc Netem 8001: dev eth0 root refcnt 2 r2q limit 1000

It means that the generated packets are directly sent in the Netem queue where emulation took place. Netem queue does not send the packets towards ROHC library for compression because it does not filter the packets for compression. Finally, the un-compressed emulated data is forwarded to Ethernet driver and transmitted at Ethernet (eth0) interface at compressor end.

Hence, our major task is emulation of the compressed packet stream which is not achieved with this approach.

7.1.2 Netem Ingress Queue

Netem ingress queue disc is usually used for packet emulation of incoming traffic at receiving node. In our case Netem ingress queue can emulate the incoming traffic and sends it towards ROHC library for decompression at the de-compressor end. In this way, we can develop a channel packet loss or delay situation for the de-compressor node for ROHC performance evaluation.

Netem ingress queue disciplines cannot directly configure on Ethernet (eth0) interface. It is needed to use 'Intermediate Function Block ', known as IFB [34]. This network device allows us to attach queue disciplines to the incoming packets at the receiver node. Actually, it creates a logical interface ifb0, as described in the script no.1 in Appendix B. This approach redirects the incoming traffic from Ethernet (eth0) interface to IFB interface (ifb0). In this way, we can emulate the incoming compressed traffic at ifb0 interface. Then, the emulated traffic could deliver to the ROHC library for de-compression.

We believed that the interface type eth0 should replace with ifb0 in the ROHC execution script as given in Appendix A. In this way, incoming compressed traffic will emulate first at Ethernet interface (eth0). Afterwards, the emulated packet stream will send towards ROHC library via IFB interface (ifb0). Consequently, the interface type is replaced from eth0 to ifb0 in ROHC execution script .i.e., $IF = ifb0.

Consequently, TAP (tap0) interface have been lost itself at the de-compressor node. In this condition it was not able to receive and observe the de-compressed traffic at the de-compressor node. It was only able to receive the compress traffic at de-compressor. Moreover, incoming compressed packets emulation was possible.

Finally, we decided to modify the queue disciplines at the source node. In this way, we can emulate the compressed traffic at the compressor node and evaluate the robustness at de-compressor end.

7.1.3 Netem Traffic Source Emulation

Queue discipline is modified in the ROHC execution script (Appendix A) for the traffic source emulation scenario as described in script no.02 Appendix B. In this way, the generated traffic (MGEN) is emulated first and then it further passes towards rohc sub-queue. Finally, the emulated packet stream is compressed in the ROHC library. The de-compressor node receives the emulated packet stream which reflects the real imperfect channel scenario. On the other hand, we have observed the behavior of ROHC devices (compressor & de-compressor) under packet loss and delay attributes. But this approach has one drawback which produces the unacceptable results.

It is evaluated that the overall packet loss percentage at receiver (de-compressor) end is greater than Netem emulated loss percentage at sender (compressor) end. For example, Netem queue is configured at 50% loss for the MGEN generated packet stream but the loss percentage at compressor Ethernet (eth0) interface is 75% as shown in Table 13. The problem is identical with other values like 45%, 55% and 60% etc.

Netem queue Loss	MGEN generated packets	Transmitted Packets at compressor end	Drop packets at Compressor end	Received Packets Decompressor	Loss (%)
50%	5000	1248	3752	1248	75
55%	5000	1004	3996	1004	79.9

Table 13: Traffic source emulation

Hence, this approach is also not acceptable due to loss percentage inconsistency at sender and receiver ends.

7.2 Netem Limitations

7.2.1 Weak Correlated Loss Generator (CLG)

We have found that the bridge emulation scenario is a consistent approach for channel emulation. Since, independent loss pattern does not de-synchronize the ROHC devices unless it happens at the starting of communication setup. On the other hand, the loss burst patterns are the best way to develop a de-compression failure state during normal operation where we can evaluate the ROHC robustness. The correlation (in percentage) attribute is added with random loss at bridge Ethernet (eth1) interface. Netem script is given in Appendix B (Netem Bridge independent Packet Loss with Correlation).

We have observed a weak correlated loss generation with Netem. Table 14 represents this weakness, where we have 6.98% over all packet loss at de-compressor Ethernet (eth0) interface for the configured loss of 40% with 90% correlation at bridge interface (eth1). Similarly, overall packet loss of 92.4% for a configured 60% loss with 90% correlation at bridge PX3 Ethernet interface (eth1).

Loss Correlation	TX-Packets (eth0)-Compressor	RX-Packets (eth0)-Decompressor	RX-Packets (tap0)-Decompressor	Drop Packets (eth0)-Decompressor	Drop Packets (tap0)-Decompressor	Loss (%) Eth0	Loss (%) Tap0
40% 90%	5000	4652	4652	349	349	6.98	6.98
60% 90%	2432	186	146	2246	2286	92.4	94

Table 14: Weakness with the Netem correlation generation

Afterwards, the problem is investigated with Netem mailing-list community. We have found that the stated problem is a Netem implemented bug and already addressed for possible solution at http://netgroup.uniroma2.it/twiki/bin/view.cgi/Main/NetemCLG.

Finally, the Linux kernel and iproute2 is update with Netem CLG implementation patch. The modified version of *TC* allows the user to add loss features to the outgoing interface according to several models such as GI (General and Intuitive) model [27].

7.2.2 Packet Loss Generation Reliability

It is observed that Netem queue does not produce accurate packet loss percentage for a test case which matches with the configured parameters. The packet loss generation percentage deviates from the configured input values. It does not repeat every time but it exists. Additionally, it is observed that there are certain test result values that is lower than confidence interval lower limits and sometimes it become higher than confidence interval upper limits as described in Sections 6.2.2 & 6.3.2.

It might be a timing problem which is explained with a research paper reference such as.

Linux timer granularity affects the real-time nature of Netem, choice of pseudo-random number generator (PRNG) impacts emulation results. Additionally, Linux is not a real-time system and this provides some constraints on the performance of a real-time simulator such as Netem [35].

7.3 Other Issues

It is observed in our testbed design that the sender (Compressor) node drops the receiver (de-compressor) node Ethernet address when the Netem queue emulates the loss in packet stream. At the same instant, all outgoing data packets are discarded which were waiting for transmission at outgoing Ethernet (eth0) interface at sender end.

In case of higher packet loss (time>=40 sec) due to Netem queue at bridge (PX3) node, compressor node moves to downward (IR state) transition itself without error recovery request from de-compressor end. The problem is resolved with a static entry in ARP table. In such way, the destination (de-compressor) node MAC address is permanently added in the sender ARP table with the following command.

```
vyatta@vyatta:~$sudo arp –v –I eth0 –s 192.168.1.13 10:1f:74:f2:89:cb
```

Chapter 8 Conclusions & Future Work

In this thesis work, we have done the performance evaluation of the robust header compression (ROHC) with transport protocol (UDP) & packet format (IPv4) for one hop wireless Ad hoc network. The performance is judged in terms of robustness against transmission errors when it is being used over imperfect channel situations.

The main contribution of this thesis is the emulation of hybrid wired channel with packet loss and delay conditions which are caused by imperfect wireless channel and signal propagation. Furthermore, bridge emulation lab scenario develops a real-life testbed for ROHC performance assessment.

Consequently, we found that ROHC protocol is very robust over the imperfect channel in terms of two loss patterns. Firstly, it is robust up to 40% channel loss for independent packet loss pattern and only fails due to IR header packet lost at the starting period of communication setup.

Secondly, it is robust up to 29 second packet lost time duration for the burst or consecutive packet loss pattern over the channel.

The main conclusion is that ROHC can useful for wireless Ad hoc networks which has higher channel loss and packet delay (in seconds) medium. It is possible to reduce packet loss due to bit error rate by inherently sending smaller packets and reduce delays introduced by packet losses on the wireless link by ROHC [2]. Additionally, it is also effective for satellite communication applications where we have higher BER and RTT.

Future work would be performance evaluation for higher data rates with hybrid testbed. Additionally, evaluate the protocol performance in a multihop hybrid wired network. Furthermore, ROHC performance evaluation with a wireless testbed for single & multihop hop networks respectively.

References

[1] "The Journal of Military Electronics & Computing," RTC, Group, Inc, November 2011. [Online]. Available: http://www.cotsjournalonline.com/articles/view/102158. [Accessed May 2013].

[2] "The concept of robust header compression, ROHC," EFFNET, Bromma, Sweden, 2004.

[3] "ROHC Library Wiki," [Online]. Available: http://rohc-lib.org/wiki/doku.php?id=supported-systems. [Accessed May 2013].

[4] Ching Shen Tye, Dr.G. Fairhurst, "A Review of IP Packet Compression techniques," in *Proceedings of PostGraduate Networking*, Aberdeen University, Scotland, 2003.

[5] V. Jacobson, "Compression TCP/IP headers for Low speed serial Links," *RFC1144,* February 1990.

[6] M. Degermark, B. Nordgren, S. Pink, "IP Header Compression," *RFC2507,* February 1999.

[7] "Internet Engineering Task Force (IETF)," Network Working Group, "RFC 3095: Robust Header Compression", July 2001. [Online]. Available: http://www.rfc-editor.org/rfc/pdfrfc/rfc3095.txt.pdf. [Accessed April 2013].

[8] Kaddoura. M, Schneider. S, "scalable and robust end-to-end header compression techniques for wireless ad hoc networks," in *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, 2004.

[9] Hai Wang, Dawei Niu, Wendong Zhao, Yuan Yan, Gup Yan, "A Robust Header Compression Method for Ad hoc Network," in *International conference on networking and services. ICNS*, 2006.

[10] Jesus Arango, Syed Al, Daniel Hampel, "Header Compression for Ad hoc Networks," in *Military Communication Conference, MILCOM 2005*, 2005.

[11] Tatiana K. Madsen , Frank H. P. Fitzek , Gian Paolo Perrucci , Ramjee Prasad, "Joint approach for IP overhead reduction reduction in wireless Ad hoc networks," in *9th International symposium on wireless personal multimedia communications*, 2006.

[12] "Robust Header Compression (charter-ietf-rohc-05)," 2010. [Online]. Available: http://datatracker.ietf.org/doc/charter-ietf-rohc/. [Accessed 2013].

[13] K. Sandlund,G. Pelletier, L-E. Jonsson, "Internet Engineering Task Force (IETF)," Network Working Group, "RFC 5795: Robust Header Compression", 2010. [Online]. Available: http://www.rfc-editor.org/rfc/pdfrfc/rfc5795.txt.pdf. [Accessed 2013].

[14] G. Pelletier, K. Sandlund, "Internet Engineering Task Force (IETF)," Network Working Group, "RFC 5225: Robust Header Compression", 2008. [Online]. Available: http://www.rfc-editor.org/rfc/pdfrfc/rfc5225.txt.pdf. [Accessed 2013].

[15] French space agency (CNES), Thales Alenia Space (TAS) and Viveris Technologies, "Robust Header Compression (ROHC) Library," [Online]. Available: https://launchpad.net/rohc. [Accessed 2013].

[16] A. Keller, "Mannual tc Packet Filtering and netem," 2006. [Online]. Available: http://tcn.hypert.net/tcmanual.pdf. [Accessed April 2013].

[17] "Mannual Reference pages - BPF(4)," [Online]. Available: http://www.gsp.com/cgi-bin/man.cgi?topic=bpf. [Accessed April 2013].

[18] "Universal TUN/TAP device driver.," [Online]. Available: https://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt. [Accessed April 2013].

[19] B. hubert, "tc(8) - Linux man page," [Online]. Available: http://linux.die.net/man/8/tc. [Accessed April 2013].

[20] Marius Næss Olsen, Mariann Hauge, "Integration of Robust Header Compression (ROHC) in a Linux based tactical router – an extension to Linux traffic control," Oslo, 2011.

[21] M. A. Brown, "Classful Queuing Disciplines (qdiscs)," [Online]. Available: http://linux-ip.net/articles/Traffic-Control-HOWTO/classful-qdiscs.html#qc-htb-borrowing. [Accessed May 2013].

[22] M. D. a. devik, "HTB Linux queuing discipline manual - user guide," 05 05 2002. [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm. [Accessed April 2013].

[23] T. Distler, "Netem WAN Emulation: How to Setup a Netem Box," [Online]. Available: http://tdistler.com/2011/06/10/netem-wan-emulation-how-to-setup-a-netem-box. [Accessed April 2013].

[24] M. Devera, "tc-htb(8) - Linux man page," [Online]. Available: http://linux.die.net/man/8/tc-htb. [Accessed April 2013].

[25] A. N. Kuznetsov, "tc-tbf(8) - Linux man page," [Online]. Available: http://linux.die.net/man/8/tc-tbf. [Accessed April 2013].

[26] "Networks and Communication Systems Branch, MGEN," Naval Research Labortory (NRL) Protocol Engineering Advanced Networking (PROTEAN) Research Group, [Online]. Available: http://cs.itd.nrl.navy.mil/work/mgen/index.php. [Accessed April 2013].

[27] S. Salsano, F. Ludovici, A. Ordine, D.Giannuzzi, "Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel," niversity of Rome “Tor Vergata”, Rome, 2012.

[28] S. Salsano, "NetemCLG (Correlated Loss Generator): fixing loss model of Netem," [Online]. Available: http://netgroup.uniroma2.it/twiki/bin/view.cgi/Main/NetemCLG. [Accessed May 2013].

[29] D. Barvaux, "Compliance with RFCs," [Online]. Available: http://rohc-lib.org/wiki/doku.php?id=library-compliance-rfcs. [Accessed May 2013].

[30] D. Barvaux, "Library ROHC features," [Online]. Available: http://rohc-lib.org/wiki/doku.php?id=library-todo. [Accessed May 2013].

[31] "Indepth: Packet Loss Burstiness," VOIP Troubleshooter.com, [Online]. Available: http://www.voiptroubleshooter.com/indepth/burstloss.html. [Accessed May 2013].

[32] "Confidence interval," [Online]. Available: http://en.wikipedia.org/wiki/Confidence_interval. [Accessed May 2013].

[33] "Optimized SATCOM," LTI DataComm, [Online]. Available: http://www.ltidata.com/technologies/optimization/satcom.html. [Accessed May 2013].

[34] "The Linux Foundation," [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. [Accessed May 2013].

[35] S. Hemminger, "Network emulation with NetEm," in *Linux Conf Au*, 2005.

[36] "Effnet AB," [Online]. Available: http://www.effnet.com/sites/effnet/pages/uk/default.asp. [Accessed 2012,2013].

[37] "FFI: Forsvarets forskningsinstitutt," [Online]. Available: http://www.ffi.no/en/About-FFI/Sider/default.aspx. [Accessed 2013].

[38] "VOIPTROUBLESHOOTER.COM," [Online]. Available: http://www.voiptroubleshooter.com/indepth/burstloss.html. [Accessed 04 May 2013].

[39] "acticom mobile networks," [Online]. Available: http://www.acticom.de/robust-header-compression-2/. [Accessed 04 May 2013].

[40] "The experts in IP Header Compression," EFFNET AB, [Online]. Available: http://www.effnet.com/sites/effnet/pages/uk/default.asp. [Accessed May 2013].

Appendix A

ROHC Queue Discipline Execution Script [14]

```
#!/bin/bash
RATE=10mbit
# Determine if we are running standalone or in a vyatta distro
if [ -f kernel_module/sched_rohc.ko ]; then
# Not vyatta, checking networking interface to use, first try 'emX', then 'pXpY'
KPATH=kernel_module
TC=../iproute/tc/tc
ROHC=./rohc-user/rohc
RES=`ip link|grep em.:`
if [ $? -eq 0 ]; then
# Found match for onboard interface, now find slot number
IF=`echo $RES|cut -d: -f2|sed 's/^ //'`
else
RES=`ip link|grep p.p.:`
if [ $? -eq 0 ]; then
# Found match for external interface, now find bus slot number
IF=`echo $RES|cut -d: -f2|sed 's/^ //'`
else
echo "Could not find suitable intefaces to use:"
ip link
exit
fi
fi
elif [ -f /lib/modules/`uname -r`/kernel/net/sched/sch_rohc.ko ]; then
echo "Assuming we are running in vyatta"
KPATH=/lib/modules/`uname -r`/kernel/net/sched
IF=eth0
TC=tc
ROHC=/opt/vyatta/sbin/rohc
fi
# Verify that the extracted interface name exists
RES=`ip link show $IF`
if [ $? -ne 0 ]; then
echo "Interface $IF does not seem to exist, please check setup or this script"
exit
fi
echo "Using kernel path $KPATH for sch_rohc.ko and sched_rohc.ko"
echo "Using interface $IF"
# Delete old qdisc on device
```

[14] Courtesy of Thales AS Norway.

```
echo "Delete TC rules"
$TC qdisc del dev $IF root
sleep 1
echo "Kill running rohc process"
killall rohc
sleep 1
echo "Removing kernel module sch_rohc"
rmmod sch_rohc
echo "Removing kernel module sched_rohc"
rmmod sched_rohc
echo "Removing old rohc_tap device"
rm /dev/rohc_tap
#if [ ! -f /dev/net/tap0 ]; then
# echo "Creating tap0"
# mknod /dev/net/tap0 c 10 200
# tunctl -u root -t tap0
#fi
echo "Inserting kernel module sched_rohc"
insmod $KPATH/sched_rohc.ko
echo "Inserting kernel module sch_rohc"
insmod $KPATH/sch_rohc.ko
ID=`cat /proc/devices|grep rohc|cut -d" " -f1`
echo "ROHC device major is $ID"
echo "Creating rohc_tap device"
mknod /dev/rohc_tap c $ID 2
echo "Starting ROHC app"
$ROHC /dev/rohc_tap $IF &
echo "Setting up TC Qdisc and filters"

# Add outgoing limiting using HTB on device

$TC qdisc add dev $IF root handle 1: htb default 16
$TC class add dev $IF parent 1: classid 1:1 htb rate $RATE ceil $RATE
$TC class add dev $IF parent 1:1 classid 1:11 htb rate 1mbit ceil $RATE

# Add ROHC to htb class 1:11
$TC qdisc add dev $IF parent 1:11 handle 11: rohc limit 5

# Add TC filter that matches TOS 0x28 and applies ROHC for that traffic
$TC filter add dev $IF protocol ip parent 1:0 prio 1 u32 match ip tos 0x28 0xff flowid 1:11

# Turn off rp_filter, it may stop source IP from $IF lan to enter via tap0
sleep 2
echo "Turning off rp_filter for tap0 and 'all'"
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
echo 0 > /proc/sys/net/ipv4/conf/tap0/rp_filter
echo "Done"
```

Appendix B Netem Scripts

1. Netem Ingress Queue Discipline

Netem with **ingress queue** problem (ifb interface)

```
vyatta@vyatta:~$sudo ip link set dev ifb0 up
vyatta@vyatta:~$sudo tc qdisc add dev eth0 ingress
vyatta@vyatta:~$sudo tc filter add dev eth0 parent ffff: protocol ip u32 match u32 0 0 flowid 1:1 action mirred egress redirect dev ifb0
 vyatta@vyatta:~$sudo tc qdisc add dev eth0 root netem LOSS
```

2. Netem Traffic source Queue Discipline

```
$TC qdisc add dev $IF root handle 1: netem
$TC qdisc add dev $IF parent 1:1 handle 10: htb default 16
$TC class add dev $IF parent 10: classid 10:1 htb rate $RATE ceil $RATE
$TC class add dev $IF parent 10:1 classid 10:11 htb rate 1mbit ceil $RATE prio 1

# Add ROHC to htb class 1:11
$TC qdisc add dev $IF parent 10:11 handle 11: rohc limit 5

# Add TC filter that matches TOS 0x28 and applies ROHC for that traffic
$TC filter add dev $IF protocol ip parent 10:0 prio 1 u32 match ip tos 0x28 0xff flowid 10:11
```

3. Netem Bridge Burst Packet Loss for Forward Channel Emulation

```
vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth1 root netem gimodel P Burst
```

P = loss percentage, Burst = Mean burst duration

e.g.: `vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth1 root netem 50 35`

4. Netem Bridge Burst Packet Loss for Forward & Reverse Channels Emulation

```
vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth1 root netem gimodel P Burst
```

P = loss percentage, Burst = Mean burst duration

```
vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth0 root netem gimodel P Burst
```

5. Netem Bridge Burst Packet Loss for Forward Channel & Delay for Reverse Channel

```
vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth1 root netem gimodel P Burst
```

P = loss percentage, Burst = Mean burst duration

```
vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth0 root DELAY
```

DELAY: = delay TIME [JITTER [CORRELATION]]

6. Netem Bridge Independent Packet loss for Forward Channel Emulation

```
vyatta@vyatta:~$sudo tc qdisc add dev eth1 root netem LOSS [PERCENTAGE]
```

7. Netem Bridge Independent Packet loss for Forward & Reverse channels Emulation

```
vyatta@vyatta:~$sudo tc qdisc add dev eth1 root netem LOSS [PERCENTAGE]

vyatta@vyatta:~$sudo tc qdisc add dev eth0 root netem LOSS [PERCENTAGE]
```

8. Netem Bridge Independent Packet Loss for Forward Channel & Delay for Reverse Channel

```
vyatta@vyatta:~$sudo tc qdisc add dev eth1 root netem LOSS [PERCENTAGE]

vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth0 root DELAY

DELAY: = delay TIME [JITTER [CORRELATION]]
```

9. Netem Bridge Independent Packet Loss with Correlation

```
vyatta@vyatta:~$sudo tc qdisc add/change/del dev eth1 root netem LOSS [PERCENTAGE] [CORRELATION]]
```

Appendix C

MEGN Script

```
# START MGEN Version 5.02, rate =5, time = 1000 seconds
10.0 ON 1 UDP SRC 5000 DST 192.168.1.13/5001 PERIODIC [5 64] TOS 0X28
1010.0 OFF 1
#end
```

```
# START MGEN Version 5.02, rate =5, time = 2000 seconds
10.0 ON 1 UDP SRC 5000 DST 192.168.1.13/5001 PERIODIC [5 64] TOS 0X28
2010.0 OFF 1
#end
```

Appendix D

Bridge Configuration Script

```
vyatta@vyatta:~$configure
vyatta@vyatta# set interfaces bridge br0
vyatta@vyatta# set interfaces ethernet eth0 bridge-group bridge br0
vyatta@vyatta# set interfaces ethernet eth1 bridge-group bridge br0
vyatta@vyatta# commit
```