Budapesti Műszaki és Gazdaságtudományi Egyetem

Robust Header Compression in Wireless Networks

Robust Header Compression vezeték nélküli hálózatokban

TDK dolgozat

Készítette:

Tömösközi Máté

mérnök informatikus MSc.

szakos hallgató

Témavezető:

Dr. Ekler Péter

egyetemi adjunktus

Berlin

2013

Contents

Összefoglaló

A telekommunikáció manapság mindenki által olcsón elérhető, de vannak olyan felhasználási területek amikor a lehető legkevesebb bájt átvitelére kell törekedni. Ezek jellemzően főleg a vezeték nélküli és műholdas rendszereket használó hálózatokat érintik, melyek manapság a mindennapos kapcsolattartás elengedhetetlen részét képezik.

Az okostelefonok és egyéb mobil internetet használó eszközök elterjedésével jelentősen megnőtt a digitális adatkapcsolat alapú hálózatokra eső terhelések. Mivel az infrastruktúra bővítése igen időigényes és túlontúl költséges lehet, ezért a szolgáltatók és gyártók számára egy olcsóbb és egyszerűbb alternatívát jelent magának az adatforgalomnak az optimalizálása.

Többek között a VoIP beszélgetés során csak a protocol stack headerjeinek együttes mérete jóval meghaladhatja a tényleges adatmennyiséget. Szerencsére a „fölösleges" header bájtok tömörítésére már vannak sztenderdizált megoldások, melyek között a legújabb a Robust Header Compression (RoHC) első és második verziója.

Kérdés viszont, hogy ezek alkalmazása mennyire hatékony különböző körülmények között, illetve hogy milyen mellékhatásai vannak az összteljesítményre. Továbbá érdekes megvizsgálni, hogy az újabb – és még kevésbé elterjedt – verziók mennyire hatékonyabbak az elődeiknél. A dolgozat ezen header tömörítési technikák tárgyalása mellett bemutatja a mérési környezetet, és részletesen megvizsgálja a Robust Header Compression mindkét verziójának használata során kapott eredményeket.

Abstract

Telecommunication these days is available rather inexpensively for everyone. There are however certain areas, where there is a need to transfer as few bytes as possible. These areas include mostly the wireless and satellite systems, without which we wouldn't be able to imagine our present lives.

With the wide market penetration of smartphones and other devices that employ mobile internet, the systems that provide digital data connection are under higher load than before. Since upgrading the hardware infrastructure is slow and rather costly, the service providers and hardware producers need an alternative method to optimise their usage of the available network bandwidth.

Just during a VoIP call, the cumulative size of the protocol stack's various headers are over the actual data! Fortunately, the "useless" header bytes can be compressed with standardised techniques, of which Robust Header Compression's (RoHC) first and second versions are the newest.

Though the question remains, how do these perform in certain conditions and what are the side effects in these situations. Moreover, it's interesting to look at how much better the newer - and so far less popular - versions are than their predecessors. This study introduces these compression techniques and discusses Robust Header Compression's both versions, while giving some answers through measurement results.

Chapter 1

Introduction

Nowadays Internet based communication is quite prominent in the industry and also in the market. With the wide distribution of smartphones, the users are connected to the service providers constantly, regardless whether they are staying at a fixed point or are in motion. For real-time IP-based wireless connections this means, that we need to send packets at a constant rate, so that the users won't experience any degradation of service quality. However, building a system that could handle such a load is currently quite expensive. In the mean time this requires from hardware/software producers, that they find a cheaper solution.

During wireless communication the limited bandwidth and the relatively higher rate of errors need a special handling of the data. While, as an example, for a normal audio only session the operators can save usually more than half of the data costs by just compressing the fields in the protocol headers. Header compression usually compresses the protocol stack above the link layer. This is possible, because most of the fields in an IP traffic contain values that are constant or rarely change during a session.

As seen later in this text, we can save nearly 90% of the IP/UDP/RTP headers just by utilising *Robust Header Compression*. Let's take as an example a typical RTP packet with a full-rate GSM payload and IPv4 internet layer protocol. The overall average size of such a packet is 87 bytes, including the IP/UDP/RTP headers' 40 bytes. With a typical stream like this, it is possible to achieve an constant compression ratio of almost 60% (given an error free channel). This results in a final compressed header size that is 1/10 of the original. Table 1.1 shows the compression statistics with Robust Header Compression for a sample audio transmission using RTP.

Robust Header Compression (or shortened: RoHC) accomplishes this by maintaining com-

IP ver.	RoHC ver.	Average payload	Uncompressed header size	Compressed header size	Savings w payload	Savings header only
v4	v1	174.01 bytes	40 bytes	5.40 bytes	16%	86%
	v2			3.60 bytes	17%	91%
v6	v1	174.01 bytes	60 bytes	4.80 bytes	24%	92%
	v2			3.60 bytes	24%	94%

Table 1.1: Compression statistics for an Audio stream showing both versions of RoHC and IP

pression states on both sender and receiver sides (later referred to as compressor and decompressor). From this the receiver can reconstruct the last header easily. For further headers the sender has to update only the fields that have changed since the previous packet.

Another very important aspect of wireless communication is the round-trip time (the time it takes to send feedback back to the original sender). Usually, achieving an ideally low rate costs more for the operators than it's worth. Therefore, the compression setup has to be robust enough to handle errors without explicitly notifying the compression side via feedback.

Earlier it was also proved, that compressing a wireless communication session can have other benefits than just conserving bandwidth. Namely, it can increase the voice/video quality on an unreliable channel (see [7] and [15]). This is due to the lower likelihood of the compressed packets suffering bit errors. And there is the added benefit of having smaller packets in general, plus the extra CRC verification contained in the compressed data as well.

The compression statistics shown in table 1.1 were put together based on the measurements discussed in this text and can come very handy when considering header compression techniques. Having an understanding of what integrators can gain by utilizing them, while knowing what the pros and cons are as well, is crucial when incorporating either of these into a system. The primary aim of this text is to explain and quantify the differences between the two compression methods in a way, which haven't been done before.

Chapter 2

Goals

Considering all the benefits discussed in the introduction, one could assume, that if the providers/producers just integrate header compression into every possible service/device, we don't need to worry about bandwidth for a long time. However that is not so easily accomplished, since not every packet stream is efficiently compressible, plus we also need to consider transmission errors and runtime resource costs.

This text tries to provide a detailed explanation of how the most current header compressions work, while showing what they can accomplish regarding the compression gain in different simulated scenarios.

After a brief overview about the research of header compressions in the next part, the fourth chapter introduces Robust Header Compression in detail. This includes short explanations about the different data compression methods, compressed packet types, states, etc employed by RoHC.

The fifth chapter goes deep into the measurements and shows how the compressions perform on different captured-RTP streams and under varying transmission conditions. The novelty of these measurements lies in the in-depth discussion of the compression efficiency differences between Robust Header Compression's version 1 and 2. Since version 2 is relatively new and therefore not yet integrated into many user devices, these result could help in evaluating the potential benefits of adapting RoHCv2.

In the sixth and also last chapter a short summary concludes this text. At the end there is an appendix with some detailed figures that function as an extension to the ones found in chapter five.

In order to support and prove, that Robust Header Compression meets its expectations and

version 2 performs even better, I have built a test bed with which I was able to measure RoHC's performance and even compare the newer version to its predecessor under the exact same conditions.

As a developer at *acticom GmbH*, I have worked on both implementations of acticom's RoHCv1 and RoHCv2 solutions. Since I had easy access to both API this way, I was in a unique position where I could build a specialized environment to investigate both of them, and compile this text with which the interested Reader can gain insight into the inner workings of the world of Robust Header Compression(s).

Here, I would also like to show my thanks to acticom GmbH for their support for providing the API-s that were needed to complete the testing setup. Unfortunately, because of copyright reasons, the implementation of the measurement environment cannot be made publicly available.

Chapter 3

Predecessors and related work

Header Compression is a widely researched and applied field. There are a number of standards which we can consider the direct or indirect predecessors of Robust Header Compression. Most of all CRTP, ROCCO and ECRTP.

The first IP header compression scheme was the *Compressed Transport Control Protocol* (CTCP or VJHC). It was proposed by Van Jacobson [8] and only considers TCP protocol. CTCP combines TCP and IP headers together for better results and lower complexity. The compression algorithm itself employs *delta coding*. Here delta means the differences between two packets, and this is what will be transmitted on the channel. The advantage of this approach is the high compression ratio. Unfortunately, it's very susceptible to bit errors, which results in the dropping of many of the following packets by the receiver. There is no error detection built into this method, instead it relies on the lower and higher level protocols' protection schemes.

An improvement on this was introduced by *Perkins* [13]. The delta coding for the neighbouring packets is replaced by a reference frame, much like modern video compressions. In this case, the first packet of a frame is sent sent as is, and the following packets use the delta coding to refer to the first one. This results in better tolerance to errors compared to CTCP, albeit produces less compression gain. An improvement on this approach by *Calveras* ([3] and [4]) proposes a dynamic frame length scheme as a function of the channel state. Both of these approaches suffer from desynchronization when the first (uncompressed) packet is lost, which results in the corruption of all the packets in the same frame. A proposed improvement is available by Rossi ([16] and [17]).

The next step was to compress RTP transmissions, so the *Compressed Real Time Protocol* (CRTP) [6] was developed. *RObust Checksum-based COmpression* (ROCCO) [19] is a refine-

ment of CRTP, which improves the header compression performance for highly error-prone links and long round-trip times. Similarly, *Enhanced Compressed RTP* (ECRTP) [5] is a refinement of CRTP.

RoHC builds on these predecessors and focuses on robust and efficient header compression over highly error-prone links with long round-trip times as well. Version 1 of RoHC [2] was built around the concept of being extensible later on (see IP [9], UDP-Lite [10] and TCP profiles [12]), while RoHC version 2 [11] follows simplicity over extensibility.

Since its inception in 2001, Robust Header Compression version 1 has been under scrutiny by – among others – [18], [14] and [1]. These all show, that version 1 can reach an average compression ratio of ~85% without problems. However, there are seldom any publications that would show RoHCv1 and RoHCv2 together.

In December of 2013, together with *F.H.P Fitzek* (Aalborg University) and *P. Seeling* (Michigan University), we will be publishing an article at *IEEE GlobeCom 2013* under the title "Performance Evaluation and Comparison of RObust Header Compression (ROHC) ROHCv1 and ROHCv2 for Multimedia Traffic", which is partially based on some of the measurements also presented here, in it, however, we are only focusing on the direct performance improvement of v2 over v1 and with much less detail than in this text.

Chapter 4

Robust Header Compression

Version 1 of RoHC has been around for a while and became a part of the WiMAX[1] and 3GPP-UMTS[2] standards. Because RoHCv2 is only available since 2008, it's currently being incorporated into products like LTE testing tools.

The main benefits of using RoHC over other packet compression techniques include the following:

- Efficient compression of header data with protection for sensitive (dynamically changing) header fields.
- Quick recovery from inconsistent sender/receiver state combinations caused by transmission errors.
- Ability to repair certain inconsistent sender/receiver state combinations caused by transmission errors[3].
- Open for extension to other Level 4 protocols[4].

In the standard protocol setup, RoHC is integrated between the IP-based network layer and the link layer. Figure 4.1 shows this on the well-known OSI model. RoHC heavily relies on the

[1] Worldwide Interoperability for Microwave Access, part of the IEEE 802.16 family of wireless-networks standards.

[2] Universal Mobile Telecommunications System, third generation cellular system based on GSM, part of IMT-2000 standard set.

[3] RoHC is, of course, able to cope with different channel characteristics in terms of transmission delay, jitter and bit-error rate.

[4] Especially in case of version 1.

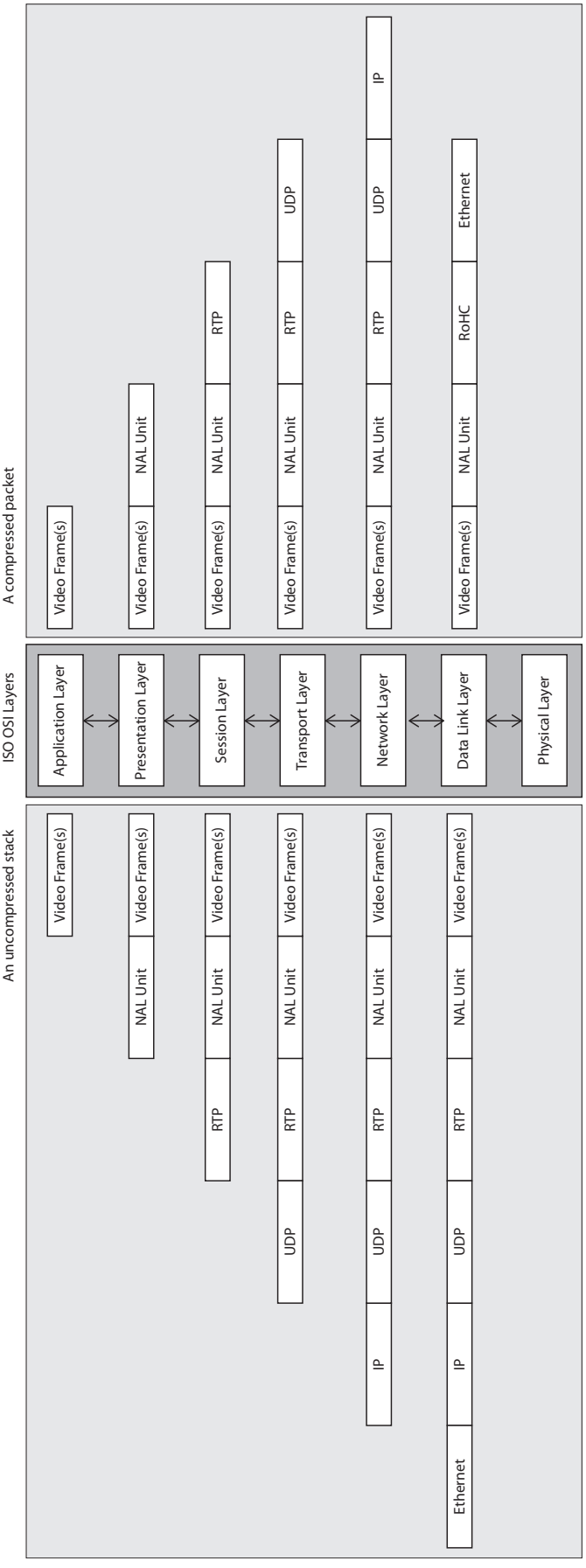


Figure 4.1: RoHC's place in the protocol stack

link layer to provide a point-to-point transport service that doesn't have any reordering on the channel. Also, for a more reliable compression, RoHC provides a feedback functionality, which can be either interspersed (i.e. having a dedicated channel) or piggybacked onto a – backwards going – compressed packet.

4.1 Description of the compression mechanism used by RoHC

RoHC's compression scheme uses states to maintain knowledge of witch one of the compressed packet types can be sent over a link (compressor) or interpreted by the receiver (decompressor). These states can be thought of as a finite machine like structure that is present in both versions (although the actual implementation details are left to the implementer, the RFC recommends this).

4.1.1 RoHC states

The 3 states of the compressor are as follows (illustrated on figure 4.3):

1. Initialization and Refresh state (IR): The compressor has to establish a new context for the session. Compression gain is very low.

2. First Order state (FO): The compressor has to synchronise the dynamically changing fields with the decompressor.

3. Second Order state (SO): The optimal compression happens here, only the most relevant data needs to be sent.

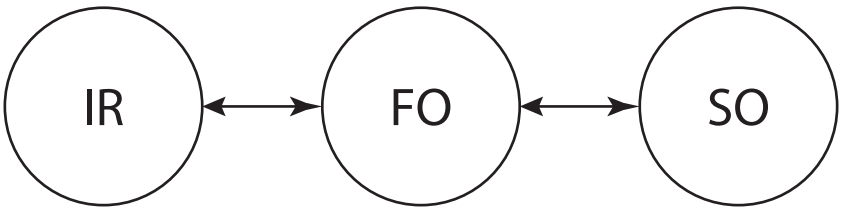


Figure 4.2: Compressor states

The decompressor uses a similar structure (figure 4.3):

1. No Context (NC): The decompression context hasn't been established yet or has to be repaired by reinitialization.

2. Repair Context (RC): The decompression context is corrupted and has to discard smaller packet types until it is refreshed by dynamic data.

3. Full Context (FC): The decompression context is up to date, and only requires smaller packet types to be transmitted in order to recreate headers.

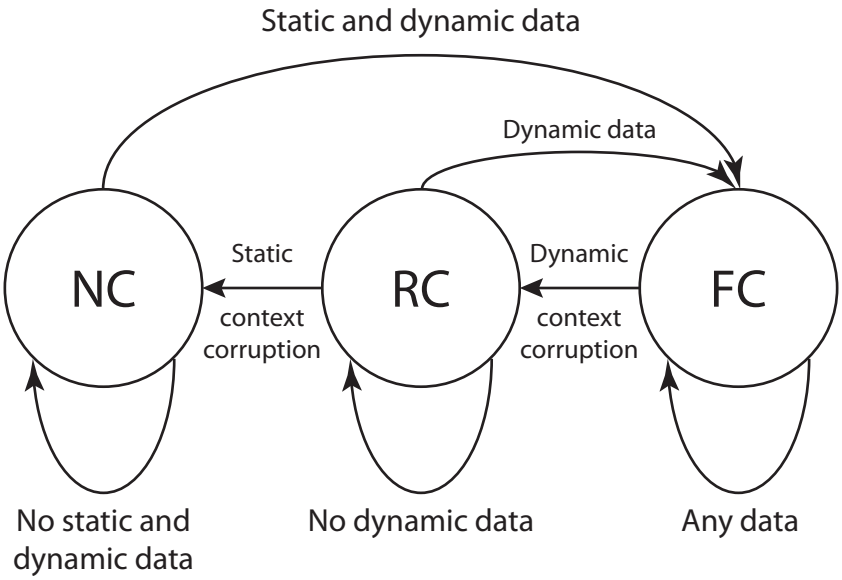


Figure 4.3: Decompressor states

In case of the first states (IR and NC), there is not enough or no data available at all for compression and decompression. For the compressor this means, that it has to (re)establish a new context at the decompressor side. Therefore the compressed packet – which will be sent out next – must contain all information. In case of the decompressor, it cannot forward a decompressed packet to the upper layers, unless it receives a full refresh from the compressor. To achieve this, the decompressor is – usually – able to notify its compressor pair by sending back a negative feedback.

Ideally, after the initial context is established, the compressor and decompressor always stay in the third states (SO and FC). If a desynchronisation occurs, the machine state must transit to a lower one (FO and RC). If the state goes to the first one, the whole context must be updated before compression can recommence. However, this can be quite undesirable, if the corruption is only present for a dynamically changing field (e.g. timestamp, which changes for every packet). Therefore, this scheme provides a ”middle” state (FO, RC), which is able to resume compression with only a partial context refresh.

4.1.2 Contexts

To achieve better compression ratio, the RoHC compressor organises the incoming packets into different contexts according to their characteristics. These characteristics are defined by the *static fields* (see later) that never or very rarely change during the transmission's lifetime. A good example for one of these "context defining" fields is the IP source and destination addresses present in the IP header.

By separating the packets according these fields, the compressor has to send – ideally – only once most of the "static" fields, and can focus on the more "dynamic" data found in the headers. All of these separated contexts are identified by a *context id* (CId) after initialisation.

This "dynamic data" – also referred to as *dynamic* and *irregular* fields – usually change from packet to packet, but (normally) in a well defined way. For example, the *IPv4 Id number* or the *RTP sequence number* increments by 1 for every packet that follows. This behaviour can be exploited by transmitting an initial value and storing it in the decompressor context. Later, this initial value can be incremented according to the difference (delta) between the stored and the new value (see the description of the LSB compression later).

These contexts are bound tightly to the compressor (decompressor) states described in the previous subsection. If a context gets corrupted or is out of sync with the compressor, only that specific context needs to be repaired or reinitialized. Exactly for this reason do the feedback packets contain the context identifier as well. Also, since we assume, that the static fields never or very rarely change, not all of the context data need to be updated during an error (usually the dynamic and irregular fields are the ones that are prone to errors). This is the reason, why the repair and first order states are being provided.

4.1.3 Profiles

Besides the separation of the packet streams based on the headers' characteristics, the used compression profile is also a very important factor.

As seen in table 4.1, the profiles define how many protocols above the link layer are compressed. It is not a requirement, that – for example – an RTP packet has to be compressed using one of the RTP profiles. It can be compressed using just the IP or IP/UDP(-Lite), albeit we save less bytes. On the other hand, this is not true in reverse. A non-RTP packet cannot be compressed via an RTP profile, therefore the integrator has to use the right profile, or the API has to employ a packet classification algorithm.

Profile identifier	Compressed protocols	v1 RFC (date)	v2 RFC (date)
0x0000	uncompressed	3095 (2001)	
0x0001 or 0x0101	IP/UDP/RTP	3095 (2001)	5525 (2008)
0x0002 or 0x0102	IP/UDP	3095 (2001)	5525 (2008)
0x0003 or 0x0103	IP/ESP	3095 (2001)	5525 (2008)
0x0004 or 0x0104	IP	3843 (2004)	5525 (2008)
0x0006	IP/TCP	4996 (2007)	
0x0007 or 0x0107	IP/UDP-Lite/RTP	4019 (2005)	5525 (2008)
0x0008 or 0x0108	IP/UDP-Lite	4019 (2005)	5525 (2008)

Table 4.1: RoHC compression profiles

The profile identifiers – as seen in tabe 4.1 – are needed for the context initialization. The first two digits represent the RoHC version (0x00 for RoHCv1 and 0x01 for RoHCv2) and the last two the profile. However, since the compressed packets are only transmitting the last byte of the identifier, it is not possible for a RoHC instance to decide which compression version is needed for decompression. Therefore, this has to be negotiated by a different protocol when an environment uses both versions[5] at the same time.

While the RFC for version 2 doesn't specify uncompressed and IP/TCP profiles, the v1's RFS-s can be used without modifications, since both of them are self contained.

It is also worth mentioning, that the dates seen beside the v1 RFC numbers show, that RoHCv1 was designed in an extensible way, whereas v2's sole RFC contains all profiles[6].

4.1.4 Field chains

Normally, a compressed packet contains a Context Identifier (CId), a discriminator (compressed packet type, see next section), a profile indicator (only during initialization), a CRC value and one or more of the following chains:

- Static chain: Transmits the static fields of a stream. These are the fields that are usually constant during a session and therefore are needed to be transmitted only once. Examples include the IP addresses and various flags and structure dependent information (next

[5] For example LTE test equipments are doing this.

[6] The idea was to make it much simpler to implement.

header field). Usually, during a session different values can be present in the same stream for these fields, but are none the less part of the same flow. In this case, multiple contexts (states) will be created, that can accommodate all the possible combinations.

- Dynamic chain: The dynamic chain contains fields, that are usually constant, but are prone to change during a session. Examples include the Hop Limit values, RTP CSRC list and various flags.

- Irregular chain: This chain contains fields that are changing from packet to packet and/or are required to be transmitted always or at certain intervals. Examples include the IP Id field, the RTP Sequence Number, etc. One field, that is always transmitted, is the MSN. Basically, this field is unique to the compression (doesn't necessarily equates to an uncompressed field) and increases (usually by one) for every packet in the stream. It is used to derive other irregular fields' values as well (see the section about LSB compression later on).

- Static-known: This is not a chain per se, but fits into this list. There are some fields that are required to be set to certain values, otherwise the packet won't be compressible. The good thing about the static-know fields is, that they are not needed to be transmitted, because they are always assumed to be set to a given value. Obviously, this proves to be a limiting factor for the possible applications of RoHC. For example, packets with fragmentation aren't compressible by RoHC (nor should they be. If we have a setup where fragmentation occurs, header compression loses its value[7]) and so, fields that has to do with fragmentation are assumed to be always 0.

4.2 RoHCv1 modes

A unique feature in RoHCv1 (compared to version 2), is the presence of certain operational modes. These modes govern, how the compressor handles context corruption using feedback and pre-emptive context refreshes. In case of version 1, these modes can be thought of as states in the compressor and decompressor instances. In the beginning, they start with the basic unidirectional mode and later they can transit to a bidirectional (compressed channel and

[7] The compression gain from compressing headers would be too small compared to the whole size of the payload.

feedback channel) as required (see figure 4.4).

In contrast to v1, the new version of RoHC doesn't explicitly specify these modes, but similar behaviours can be achieved with the right configuration of the entities, although transitions during run-time are not supported according to the RFC.

These modes are briefly described in the following part.

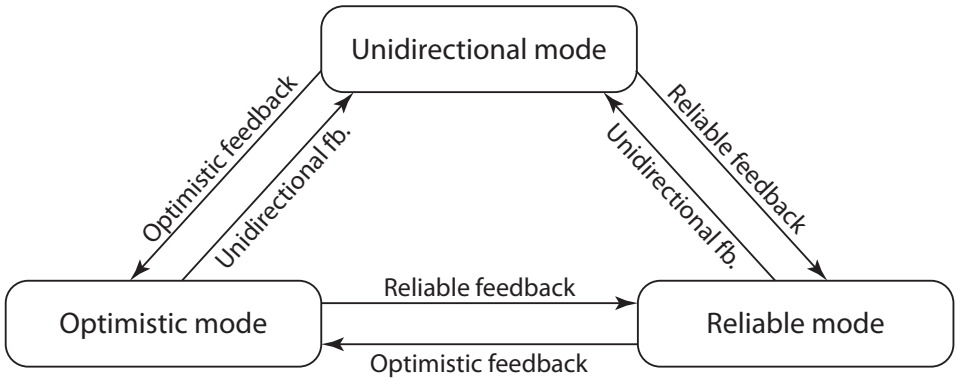


Figure 4.4: RoHCv1 modes

4.2.1 Unidirectional Mode

In unidirectional mode there are no feedback packets sent back to the compressor. By supporting this mode, RoHC can be run on links without "return" channels.

However, without feedback there's no way for the compressor to be sure, whether a compressed packet was successfully decompressed on the receiver side. To account for this, the compressor can use the *optimistic approach*, which periodically repeats context initialization and sends a changed reference value multiple times.

4.2.2 Bidirectional Optimistic Mode

The bidirectional optimistic mode uses feedback packets that are sent from the decompressor to the compressor in order to accelerate state transitions at the compressor and to avoid the periodic fallbacks to the first and second states.

Due to the mostly weak CRC protection, this mode is still relatively prone to context damage and therefore utilizes the optimistic approach as well.

4.2.3 Bidirectional Reliable Mode

This mode uses a more powerful CRC protection and a very tight coupling between the two endpoints by relying heavily on feedback received from the decompressor.

4.3 Compressed packet types

A short description of the defined compressed packet types follows. It's important to note, that between version 1 and 2, only the IR and feedback packet types resemble each other and the rest were completely redesigned.

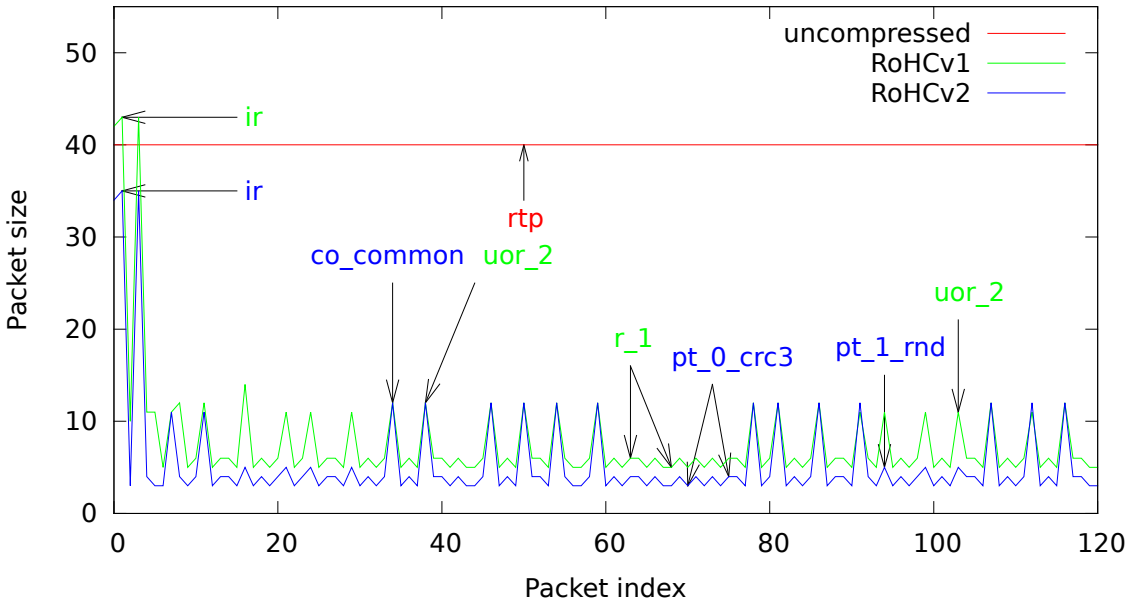


Figure 4.5: Packet types during compression

As an example, figure 4.5 notes some of the packet types employed and their respective sizes during compression of a test stream.

4.3.1 Basic packet structure

All compressed packets contain the a context identifier. The context identifier can either be 0, 1 or 2 bytes long. In Small-CId mode, a 0[8] or 1 byte long CId field is at the start of a packet, which is followed by – in most compressed packets – the first octet of the compressed

[8] If the CId in question is 0, than it doesn't need to be transmitted in the compressed packet when using Small-CIds.

packet type. With Large-CIds, after the aforementioned first octet is the 1 or 2 bytes long CId information[9].

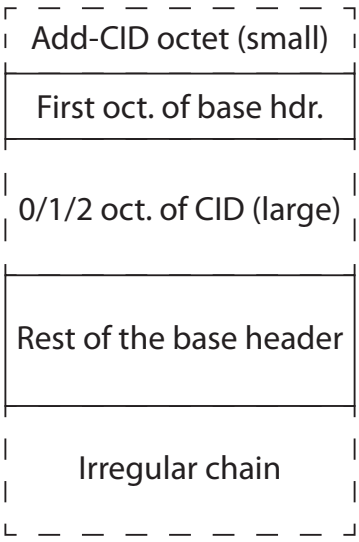


Figure 4.6: Format of the general compressed packet

Following the CId information of the first octet come the compressed packet-type dependent fields. After them, the last bytes of the packet contain the various compressed chains (if any). In case of an IR packet, this means the static and dynamic chains, for an IR-Dyn or co_repair, it's only the dynamic chain, and for the rest of the packet types, it's the irregular chain.

Figure 4.6 shows this layout on a general compressed header packet.

4.3.2 Version 1 packets

Version 1's packets don't contain any chains except for the two IR-s. Aside from these two, the compressed packets are defined by mode–type–property combinations. The mode identifies the mode in which the compression is currently working (see the section before), the type basically states how much information is contained in a packet and the optional property extends the basic packet with some additional data (for example CRC, timestamp or IP Id.).

As seen on table 4.2, only those packets are referenceable, which contain CRC prtection. This referencing means, that the packet data can be used to safely update the decompressor context, otherwise, the received data can only be used for the current packet's decompression.

In version 1, there are only static and dynamic chains defined. These are used to transmit

[9] With small-CIds, the compression only has an id pool of 16 different numbers, whereas using large-CIds, we have up to 16384 possible choices.

the – initial – values of constant (e.g. not changing fields like IP addresses, next header, etc.) and non-constant (timestamp, IP Id, etc.) SN functions.

The non-IR packet types don't contain any chains, instead they append the extension related data, encapsulation data and also the UDP checksum value at the end. This is denoted in table 4.3 by the generalised *extensions* keyword.

Mode	Type	Property	Size[10]	CRC	Referenceable	Chains or extensions
	IR		42	8-bits[11]	Yes	Static (and dynamic)
	IR	DYN	26	8-bits	Yes	Dynamic
R	0		5	None	No	Extensions
R	0	CRC	6	7-bits	Yes	Extensions
UO	0		5	3-bits	Yes	Extensions
R	1		9	None	No	Extensions
R	1	ID	9	None	No	Extensions
R	1	TS	9	None	No	Extensions
UO	1		9	3-bits	Yes	Extensions
UO	1	ID	9	3-bits	Yes	Extensions
UO	1	TS	9	3-bits	Yes	Extensions
UOR	2		10	7-bits	Yes	Extensions
UOR	2	ID	10	7-bits	Yes	Extensions
UOR	2	TS	10	7-bits	Yes	Extensions

Table 4.2: RoHCv1 compressed packet types

4.3.3 Version 2 packets

The packets types of version 2 can be separated into two groups. The first group contains packets, that are only sent during initialization (ir) and during context repair (ir and co_repair). For ideal compression, these packets are sent very rarely. The second group contains smaller packets that are most commonly used to update the decompressor contexts. Their sizes range from a 2 bytes (pt_0_crc3) to a maximum of about 12 bytes (co_common).

[10] Average observed size in different streams.

[11] The CRC is calculated over the compressed header in case of the IR packet, for the others it is done over the uncompressed instead.

Table 4.3 shows all the possible packet types for version 2.

Name	**Profile**	**Size**[12]	**CRC**	**Chains**	**Fields**[13]
ir	All[14]	34	8-bits[15]	Static & dyn.	
co_repair	All w\o UCP[16]	25	7 & 3-bits[17]	Dynamic	
co_common	All w\o UCP	11	7 & 3-bits[17]	Irregular	IP & RTP
pt_0_crc3	All[18]	3	3-bits	Irregular	
pt_0_crc7	All w\o UCP	4	7-bits	Irregular	
pt_1_seq_id	All w\o UCP	5	3-bits	Irregular	IP Id
pt_1_rnd	RTP	5	3-bits	Irregular	RTP TS
pt_1_seq_ts	RTP	5	3-bits	Irregular	RTP TS
pt_2_seq_id	All w\o UCP	5	7-bits	Irregular	IP Id
pt_2_rnd	RTP	6	7-bits	Irregular	RTP TS & m[19]
pt_2_seq_both	RTP	6	7-bits	Irregular	IP Id & RTP TS & m
pt_2_seq_ts	RTP	6	7-bits	Irregular	RTP TS & m

Table 4.3: RoHCv2 compressed packet types

[12] Average observed size during the compression of different streams.

[13] The MSN field is present in every packet and is ommitted from this column for the sake of brevity.

[14] The uncompressed profile uses IR packets in name only (for initialization), therefore the contents of the size and chains cell of this row do not apply for UCP.

[15] The CRC is calculated over the compressed header in case of the IR packet, for the other it is done over the uncompressed instead.

[16] UCP denotes the Uncompressed Profile (0x0000).

[17] This CRC is calculated over the control fields (i.e. MSN).

[18] Again, for uncompressed profile there is no ”classical” pt_0_crc3 packet (also called pt_normal in this case) like for the rest of the profiles.

[19] RTP marker bit.

4.4 Utilized compression techniques

In this section follows a short description of the compression techniques used by RoHC.

4.4.1 Self-describing variable length

The *SDVL* algorithm reduces the number of bits needed to transmit a whole field value by omitting the leading zeros. In RoHC there are 5 variants of SDVL compression: 7-bit, 14-bit, 21-bit, 28-bit and 32-bit[20].

This technique is usually employed for fields that rarely need to be updated and potentially contain leading zeros. For example: time stride for timerbased compression (see later) or the RTP sequence number (used in combination with LSB compression).

4.4.2 Window-based Least Significant Bit

The *(W-)LSB* compression relies on that the a given field's value usually increases by a small constant value. In this case, the compressor can use the fact, that only the last digits or bits change during an iteration and it suffices to transmit these.

This has the potential advantage that – in consideration of the dynamic field behaviour – the number of bits needed to transmit the changes are minimized.

Depending on how many bits we use for the LSB, we can define a window around the minimal and maximal values that can be represented this way (hence the name).

In reality, we cannot transmit negative changes with LSB, because RoHC defines it in a way, that the decompressor needs to account for the arithmetical carry. In which case, it isn't possible to distinguish between a compressed LSB value that forces carrying and a decreasing value.

The Window-based LSB compression algorithm partially contributes to the robustness of RoHC. W-LSB compression is therefore used throughout RoHC, for example, the compression of the IP Id, MSN, timestamp, etc are handled by it.

[20] To distinguish them form each other appropriate discriminators are used with the following sizes (in order): 1-bit, 2-bit, 3-bit, 4-bit and 1-byte.

4.4.3 List compression

Some data in the headers that belong to a stream, contain a variable number of items that are – from time to time – reused. The RTP CSRC list is exactly like this.

Using list compression, the compressor and the decompressor maintains a small ”database” of identifiers and values. If the compressor is sure, that the decompressor is aware of a certain value, it needs only to transmit its id (instead of the whole uncompressed value).

In case of the RTP CSRC, the compressor sends a list of 4- or 8-bits long list id-s, and the decompressor replaces these with the respective uncompressed values from its internal list.

4.4.4 Timer based compression

Under specific conditions the RTP timestamp can be compressed using the local machine clock. Using this approach, the two compression endpoints synchronise the initial timestamp value and from thereon, the decompressor updates the timestamp stored in its context by the measured elapsed time between two packets.

However, this requires the transmission link to have low delay and and that both sides have similar clock resolutions. To account for the packet transmission time, the LSB of the compressed timestamp has to be sent in the compressed packets as well.

This feature is optional in a RoHC implementation. If it is not support by a decompressor instance, the associated compressor needs to be notified through feedback.

4.4.5 Inferred values

This is not a traditional compression scheme in itself, however it is an integral part of RoHC and is needed to achieve best performance.

It is used when no data is needed to be transmitted for a compressed field. In case of a dynamic field, we can deduce the new value by incrementing the last known one with the difference between two preceding MSN values[21] and incrementing with that.

In RoHCv2, this is used for example in the pt_0_crc3 or pt_0_crc7 packets.

[21] Master Sequence Number: this field identifies the compressed packets, therefore it is always present. For each compressed packet, its value is increased by 1.

4.5 Decompressor feedback

Feedback is indispensable for RoHC when operating in a bidirectional mode. Feedback packets can be either transmitted through a dedicated channel (interspersed) or put before a compressed packet (piggybacked). In the latter case, it is even possible to transmit multiple feedbacks, when needed.

With feedback, the decompressor is able to signal one of the following conditions:

- Acknowledgement (*ACK*): The decompressor is up-to-date and is able to decompress every packet. It is in the FC state.
- Negative-Acknowledgement (*NACK*): The decompressor lost synchronisation with the compressor and can only decompress certain packets[22]. It is in the RC state.
- Static Negative-Acknowledgement (*Static-NACK*): The decompressor doesn't have any data available to be able to decompress any packet other than the IR. It is in the NC state.

There are two different feedback packet types: *FEEDBACK-1* and *FEEDBACK-2*. The FEEDBACK-1 is always interpreted as an ACK, while FEEDBACK-2 can contain one of the three. With FEEDBACK-2 the decompressor is able to send feedback options, that can signal, for example, the local clocks resolution (for timerbased compression) or insufficient local resources (i.e. memory).

Each feedback has to contain a valid Acknowledgement number (MSN), which identifies the last received and correctly decompressed packet. If this is unavailable, a FEEDBACK-2 option can be used to signal the missing value.

[22] Packets that contain full information about the dynamic fields.

4.6 Comparison of version 1 and 2

To conclude the description part of this text, a short feature comparison is presented here that outlines the differences between the two versions.

Version 1	**Version 2**
Designed to be extensible RTP, IP and UDP-Lite are in their separate RFC-s	Designed to be simple All in one RFC. No new UCP and TCP profiles.
Runtime modes (U-O-R)	No explicit runtime modes
Compressed packets are used based on the active compression mode	Compressed packets are used based on their fields and CRC protection strength
Segmentation supported	Segmentation is not supported
Supports 2 levels of encapsulation	Supports infinite levels of encapsulation
Uses lists for extensions, tunnelling, etc.	Uses static, dynamic and irregular chains
Employed by 3GPP-UMTS, WiMAX, ...	So far, mostly employed by LTE test tools

Chapter 5

Compression Measurements

After the previous chapter's description of RoHC's inner workings, the Reader is probably interested in how this compression actually performs in action. With the results presented in the following chapter, it will be possible – for example – to decide the feasibility of adapting Robust Header Compression into a potential project.

The Reader can also ascertain the actual *robustness* of RoHC with the insight that can be gained after looking at the simulated lossy link's results.

The novelty of the presented material here is, that it shows the difference between the two RoHC versions for the exact same traffic. Since version 2 wasn't designed to achieve better compression, these results would potentially be a deciding factor when choosing between the two versions.

5.1 Description of the Testbed

The following results were achieved on an AMD Athlon II X3 system running Ubuntu 12.04. All executions were limited to only one logical thread using the *taskset* command to limit parallel executions. This was done, so the captured *CPU Timestamp (TSC)* would have coherent values over multiple execution of the same measurement scenario.

The tests were implemented in a combination of *Bash*, *C*, *Python* and *GNUplot*. C providing the API access and the execution of the measurement scenarios, Python transforming the results of the latter to graphs with GNUplot and generating certain error patterns used with the C API, and a Bash script connecting all together and executing the different measurement setups.

The uncompressed packet streams were either generated by the C application online or

were captured using packetdump/wireshark and stored in, and later read from pcap files. The measured values were generated in a human readable CSV format, which can be interpreted by GNUplot or MS Excel for generation of the graphs.

The API provided by *acticom GmbH* was linked during compile time without debug symbols and maximum compiler optimization using *Link Time Code Generation* of *GCC*. The compression simulations were run in user space and the compression informations were extracted using the RoHC API's statistical interface.

All compression gains were calculated using the following formula:

$$H_s(i) = \frac{H_u(i) - H_c(i)}{H_u(i)}$$

where i is the index of the packet, H_u the uncompressed header size and H_c is the compressed packet's size.

In addition to the bandwidth savings, the compression performance is also evaluated by means of the compression/decompression methods' complexity. This complexity is measured through CPU timestamping using this simple formula:

$$T(i) = T_f(i) - T_s(i)$$

So, the complexity (time) required to compress (decompress) the i^{th} packet is the delta between the start and the finish time of the compression (decompression).

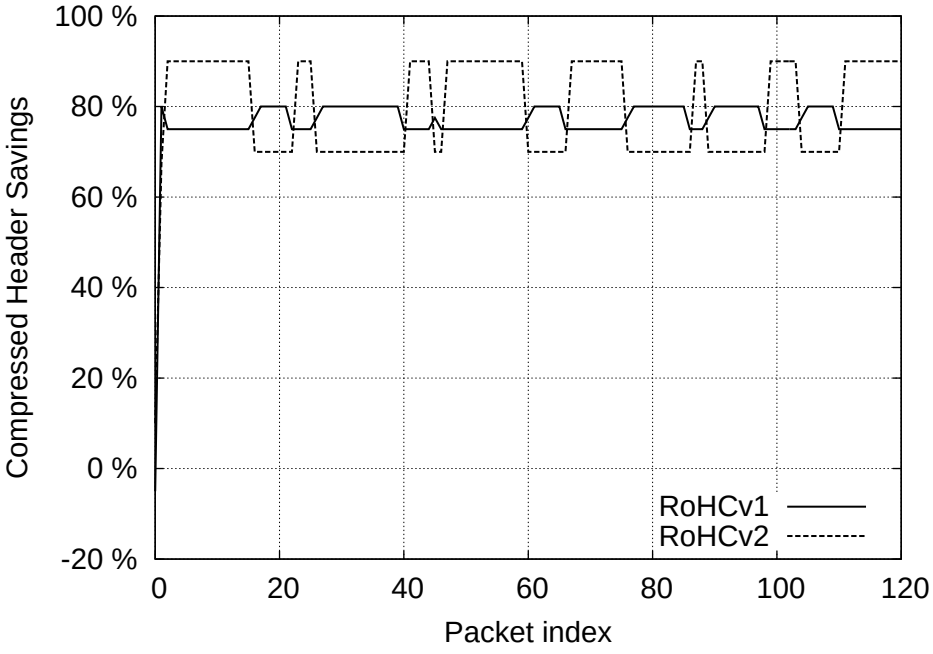
5.2 Results with captured streams

The following section presents measurement results for different application streams using acticom GmbH's RoHC v1 and v2 implementations.

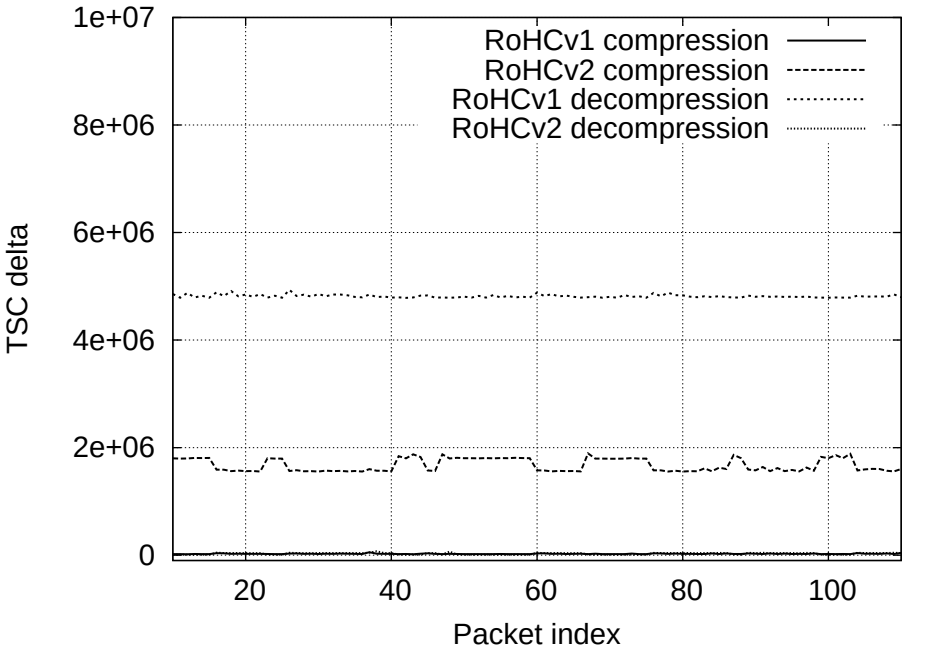
5.2.1 VLC Stream

This stream was generated using *VLC* and was transmitted on a local Ethernet based wired-network. The stream contained one RTP flow which transmitted and transcoded an audio file – provided by acticom GmbH – in a way, that it had similar characteristics to *Full-Rate GSM*.

Figure 5.1a shows that in this scenario both RoHC v1 and v2 fluctuate between ~75% and ~90%. While v2 indeed achieves better results in compression gain, it produces, at certain



(a) Header Savings



(b) Compression complexity

Figure 5.1: Measurements of a VLC stream

times, lower compression gain where v1 reaches its peak performance. This is due to the semi-static handling of the *RTP marker bit*. This bit usually signifies the start of a talkspurt, and for the rest of the time is set to 0. Both versions of RoHC assume, that the marker is 0, unless explicitly specified by the compressed packet that it is 1 instead. Unfortunately, RoHCv2 is less flexible when compressing this bit, since the pt_0 type packets (the smallest compressed packet types) cannot transmit this field. Consequently RoHCv2 cannot achieve optimal compression for a stream that has the marker bit set to always 1 (see figure 1 in the appendix).

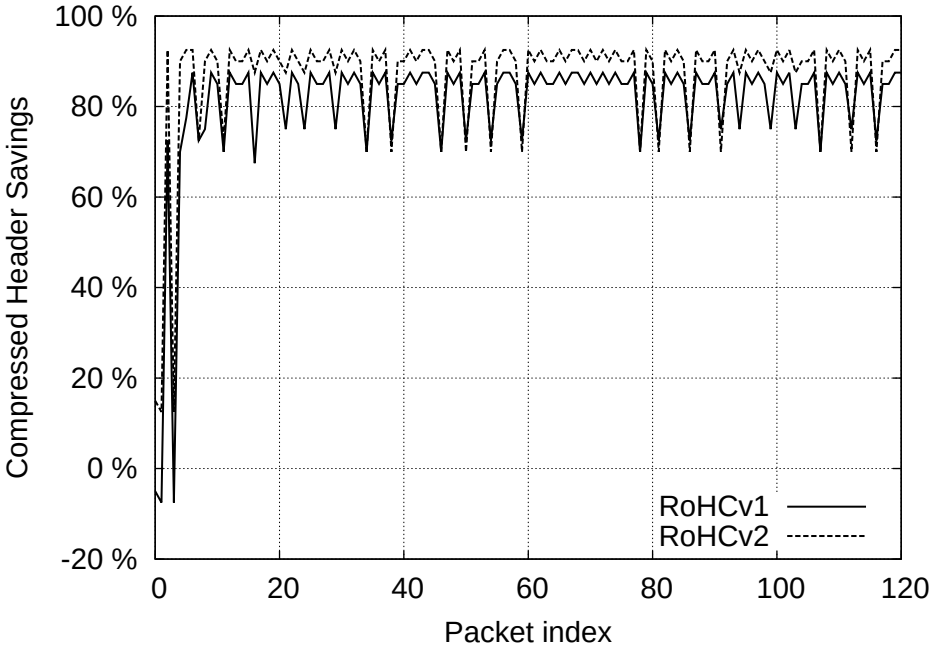
Figure 5.1b shows the CPU timestamp difference for the compression/decompression of both versions. The interesting thing here is the asymmetry regarding the compression complexity of RoHCv2 and the decompression complexity of v1. It takes significantly longer for v2 to compress than to decompress, while v1 is slower at decompression than compression. Also, the decompression of v2 and the compression of v1 is quite negligible compared to the other computations.

5.2.2 Ekiga Stream

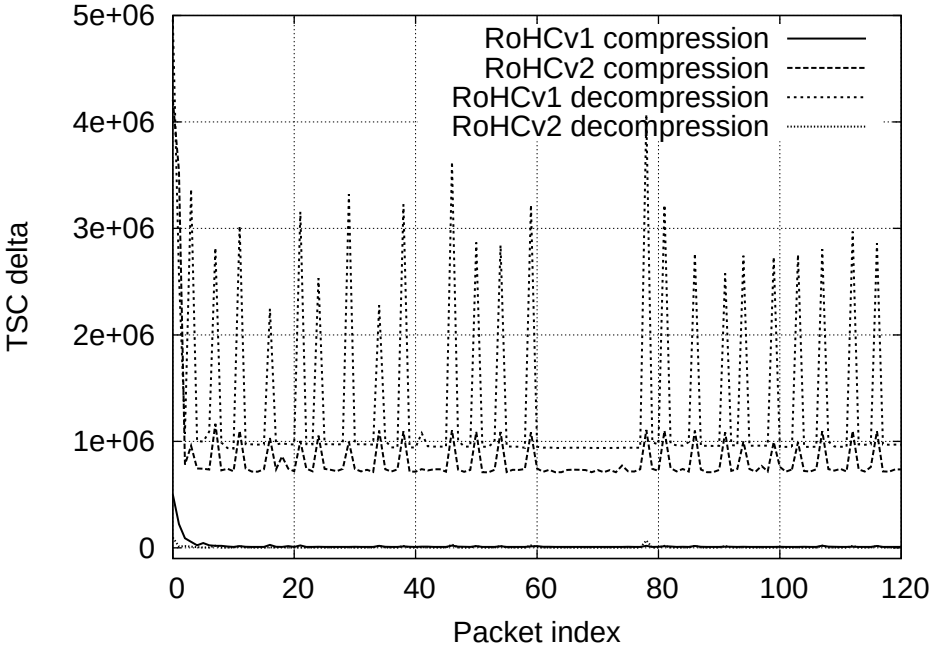
The following test is more close to an actual application of RoHC, than the one done with VLC. This stream was generated by the Ekiga opensource softphone application using an echo call, and capturing the resulting RTP streams. This call contained two audio streams and one video stream as seen on table 5.1.

In this case (figure 5.2b) both version perform very similarly, however v2 has a clear advantage of about 3-4%. The greater dips in the performance are due – again – to the marker bit of the third CId, which is set throughout the call (see figure 2 in the appendix). Also, the initialization's usual ramp up period is longer, because we have – in this case – three different contexts that need to be initialized.

The large spikes on figure 5.2b – again – occur when the set marker needs to be transmitted. The higher spikes of v2 are attributed to the currently unoptimized API of v2.



(a) Header Savings



(b) Compression complexity

Figure 5.2: Measurements of the Ekiga loopback

Fields	First (CId 0)	Second (CId 1)	Third (CId 2/3)
First occurence	#1	#2	#4
Src	0x5640a223:2828	0xc0a82114:13e6	0xc0a82114:13ea
Dst	0xc0a82114:13e6	0x5640a223:2828	0x5640a223:4aac
Ip Id	0	0	0
SN start	0xcb90	0xf5b7	0xd82c
SN delta	1	1	1
TS start	0x1540	0x10e0	0xb45a
TS delta	160	160	constantly fluctuating
Marker	always unset	always unset	always set
RTP payload	0x08 ITU-T G.711 PCMA	0x08 ITU-T G.711 PCMA	0x9f ITU-T H.261

Table 5.1: Overview of the main protocol header field characteristics for the different CId-s for the Ekiga scenario.

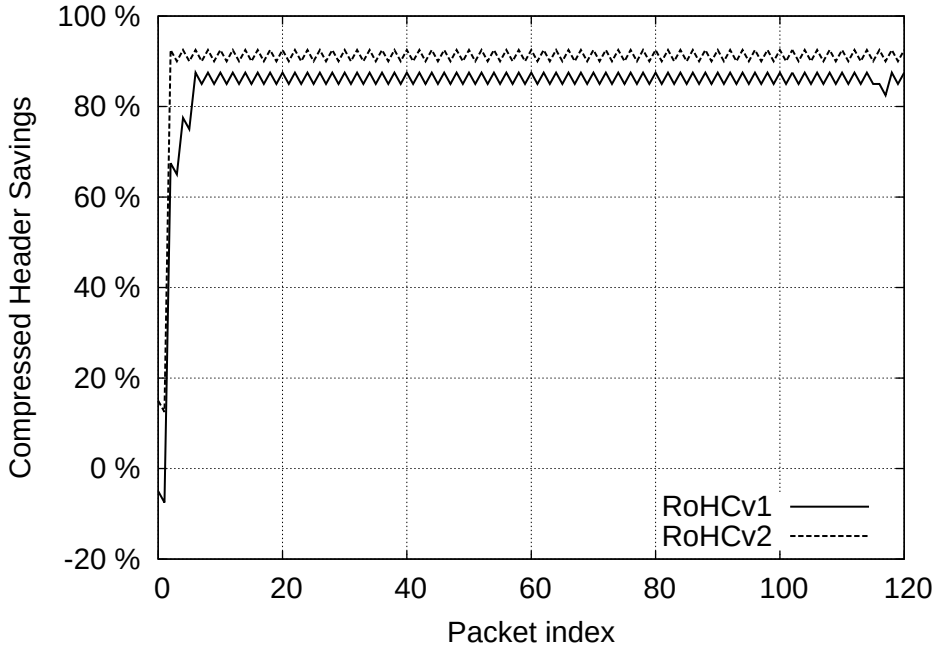
5.2.3 Asterisk Call

The last error free scenario is generated by an *Asterisk* based *VoIP call* which features a full-rate GSM talk. The calls were made using two N71 phones over local WiFi and a central Asterisk server. This scenario models a real life application of RoHC closely.

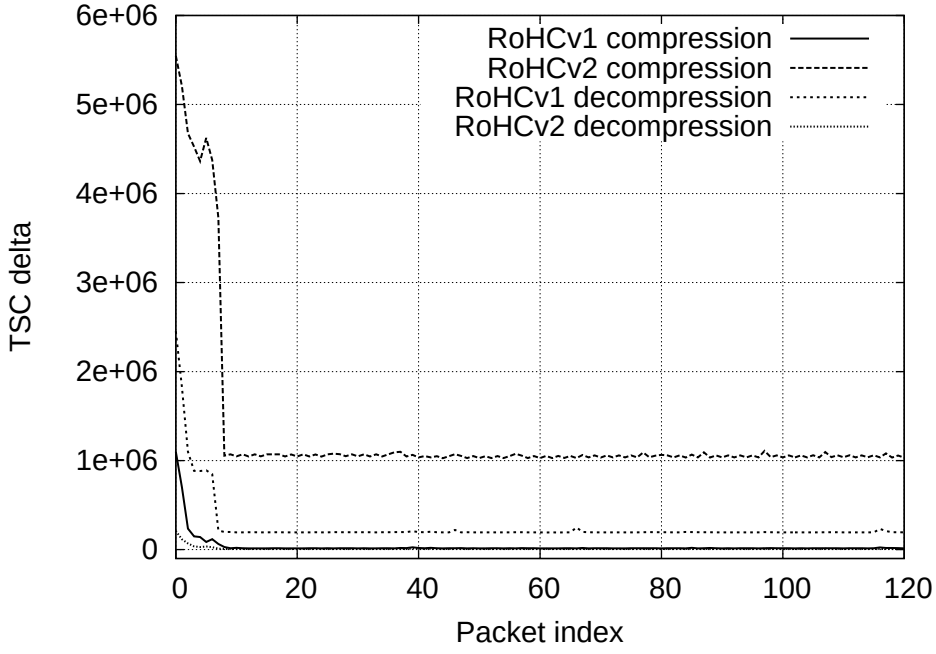
On figure 5.3b v2 clearly performs better, while v1 at the start needs more packets to adopt and reach top compression efficiency. The difference between the two compressions is exactly 2 bytes. In case of v1, this means a r_0 and for v2 a pt_0_crc3 type packet. The ”saw” shape on this diagram is caused by the presentation of all contexts on the same curve. When separated by CIds, the curves show smooth and constant shape (see figure 3 in the appendix)[1].

The complexity diagram 5.3b still shows, that the compression of v2 is the most resource ”hungry” operation.

[1] The CId 0 packets are smaller by 1 byte, because the 1 byte long context identifier doesn't need to be transmitted for this context.



(a) Header Savings



(b) Compression complexity

Figure 5.3: Measurements of a VoIP call

5.3 Results using errors

This section shows the results obtained by introducing errors to an RTP stream. The simulations of errors were done using both correlated and uncorrelated methods. The errors investigated here are the complete skipping of compressed packets during transmission (packet losses). This was simulated by skipping some of a previously captured stream's packets after compression.

The approach utilized here was the following: Before running the compressions, a packet loss profile was generated by one of the error models. This profile was generated for a multitude of different loss probabilities ranging from 0.01 to 0.7. These profiles were then combined with the previously used Ekiga stream. During the measurements interspersed feedback was enabled.

5.3.1 Uncorrelated errors

In the first error scenario, the error profiles were generated uncorrelated (e.g. the probability of packet loss is independent of the previous packet), therefore the errors don't propagate.

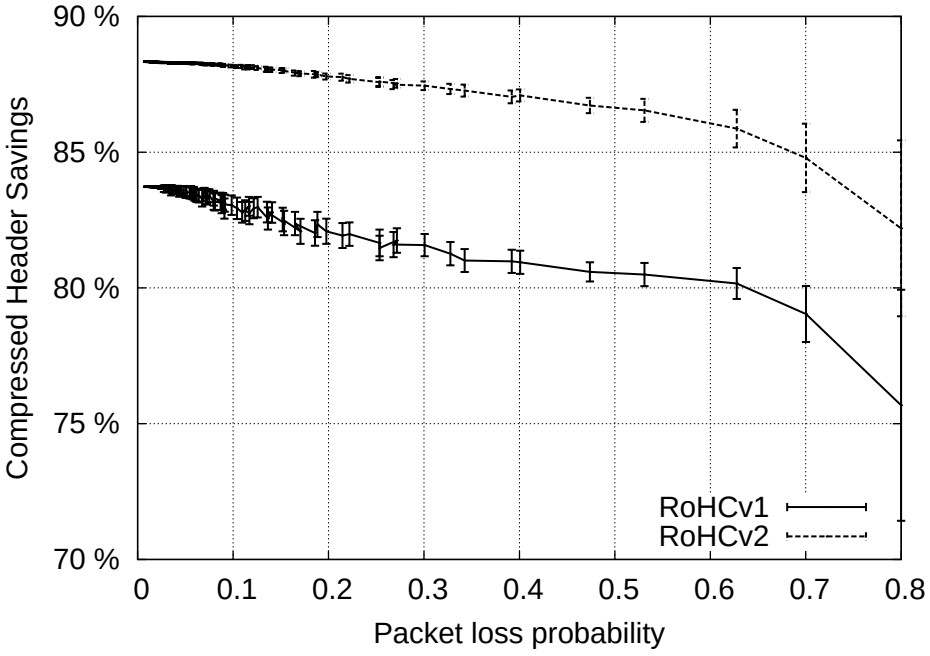


Figure 5.4: Uncorrelated error scenario

As seen on figure 5.4, the efficiency of compression keeps dropping steadily as the loss probability increases. RoHCv2 clearly dominates in compression gain with ~4-5% better results and smaller confidence intervals. Also, v2 has a smoother curve. For both versions, a sudden drop occurs after 0.6 loss probability is reached. Fortunately, this part of the diagram is not

really relevant, since links with so high loss rates would be particularly useless for everyday needs.

5.3.2 Correlated errors

In this setup a simplified Markov-chain is used to simulate correlated errors. This model has only two states, a ”good” and a ”bad” one. If the chain is in the good state, all packets are transmitted and no errors occur. However, as long as the model is in the bad state, the packets are being skipped after compression.

All-in-all, there are 4 transitions in the model (as seen on figure 5.5): the transition from good state to good state, from good to bad, from bad to bad and from bad to good. All of these have a transition probabilities assigned to them.

In the measurement setup, only two parameters are used: these are the transitions between the two states. The other two transitions are calculated as the remainder[2].

By varying these parameters, the real-life channel's burstiness can be simulated.

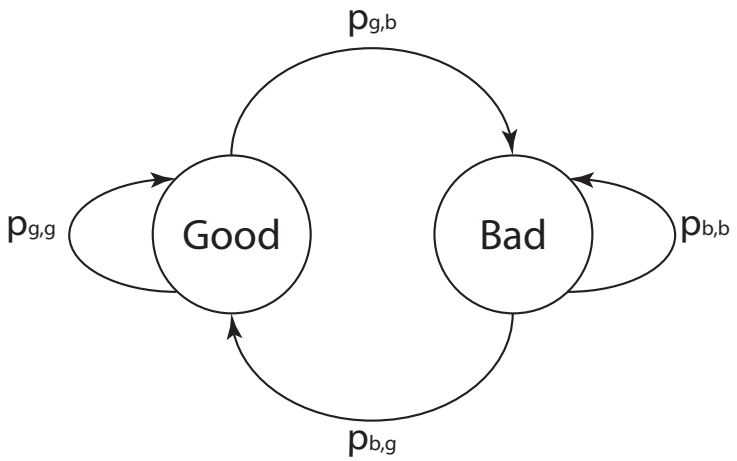


Figure 5.5: Gilber-Elliot model

The result shows a similar curve to the uncorrelated case. The reason is, that RoHC is not affected by how many packets are lost during transmission. The fields that need to be updated – for this stream – remain the same. Basically, the number of packets lost in a row, doesn't correlate with decreasing compression gain.

[2] $p_G, G := 1.0 - p_G, B$ and $p_B, B := 1.0 - p_B, G$.

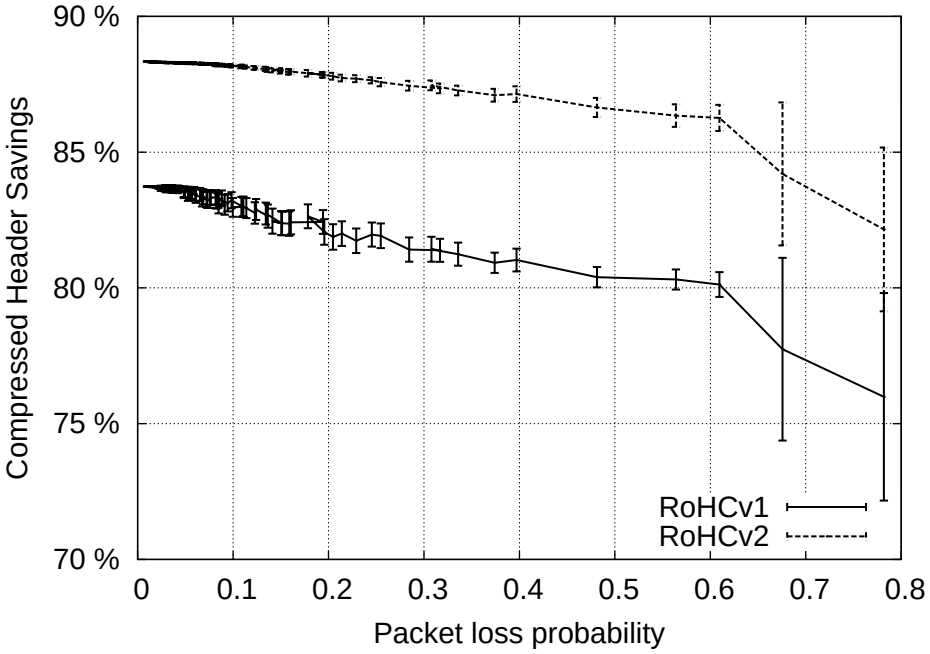


Figure 5.6: Correlated error scenario

5.4 Analysis of impact fields

It is also interesting to look at the streams before compression, since not all packet flows are perfect for compression with RoHC and therefore they could decrease the compression gain.

The following header fields are all dynamic or irregular in nature an therefore impact the compression quality the most:

- IP Id (2 bytes): This field applies only to version 4 of the Internet Protocol. It is primarily used to identify fragments of a datagram. Since fragmentation is not allowed to be compressed by RoHC, this field should be optimally left 0, but that is usually not the case.

- UDP Checksum (2 bytes): This field is an error-checking number and is not optional when used with IPv6. It is always sent uncompressed in every packet (when not 0).

- RTP Marker bit (1 bit): The usage of the marker bit is defined by the application layer. In a VoIP scenario, this is usually used to signal the start of a talkspur. Consequently, it is rather rarely set to 1. RoHC assumes, that it is set 0, unless otherwise specified by the packet.

- RTP Sequence Number (2 bytes): The sequence number is incremented by 1 for each

RTP packet and is used for packet loss detection.

- RTP Timestamp (4 bytes): The timestamp is usually used for audio-video transmissions and defines the interval between two frames. For compression it is best when this interval remains constant between packets.

Figure 5.7 shows how an IPv4/UDP/RTP packet's fields are classified in regard of "dynamicality".

In this part, the impact fields are evaluated in a setup (except the UDP checksum) in which – by certain probabilities – a fluctuation is simulated in the difference between the same field's value over following packet's. This effectively results in a degradation of compression quality, since because of this, the compressor is usually forced to switch from a sequential field behaviour to another transmission method.

In the measurement scenario a the fluctuation probability is applied by constantly increasing its value from ~0.0 to ~1.0. This fluctuation itself is nothing more than a switching between the two values used for the fields' delta. However, this is quite sufficient for the evaluations' purpose.

We can also interpret the measurements presented here, as an extension of the correlated/uncorrelated error scenario presented in the previous section, but with the packet losses occurring before compression.

5.4.1 IP Id delta

In this measurement RoHC's tolerance to the fluctuation of the IPv4 identification field is tested.

As seen on figure 5.8, the average compression ratio of the two different versions are very close to each other. RoHCv1's vary between 89% and 90%, while RoHCv2 is around 91% throughout.

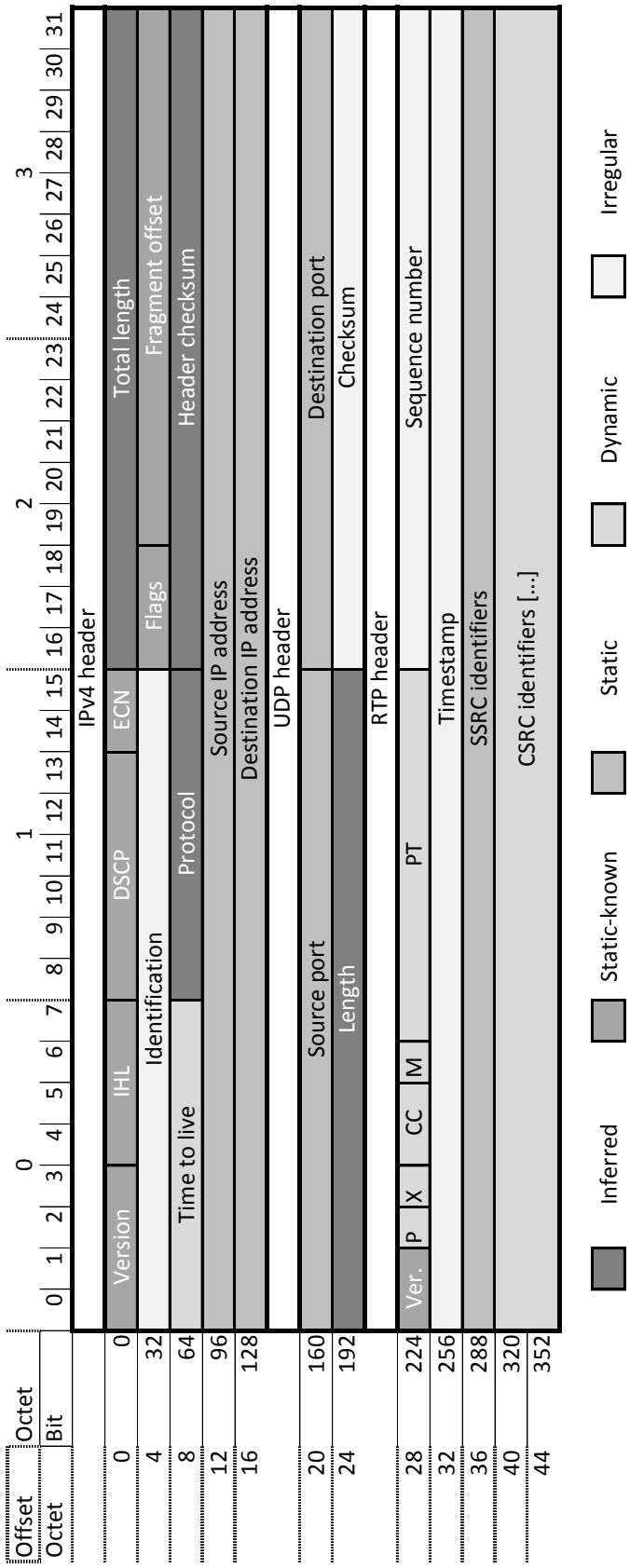


Figure 5.7: An IPv4/UDP/RTP packet's fields and compressableness

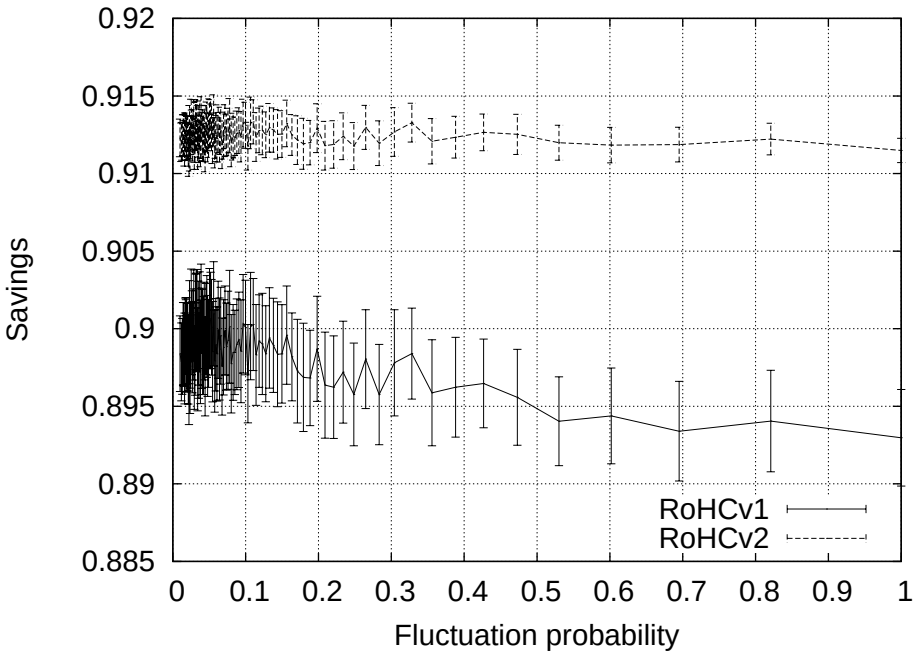


Figure 5.8: Fluctuation of the IP Id

It can be concluded, that the IPv4 identification field doesn't impact in any significant way the overall compression ratio, though v2 has a smaller confidence interval and a lot smoother curve.

5.4.2 RTP Marker Bit

On figure 5.9 we can see how the compression behaves when the marker bit fluctuates (from 1 to 0 and vice versa).

Since there's no significant difference between the two versions' efficiency to compress the marker bit, it can be concluded, that the marker alone cannot degrade the compression efficiency.

However it must noted, that v2 can only transmit the marker change using the pt_1_rnd or, pt_1_seq_ts, etc. packets, which have a bigger size by 1 byte than the optimal pt_0_crc3 packet. Hence the confidence interval difference on figure 5.9.

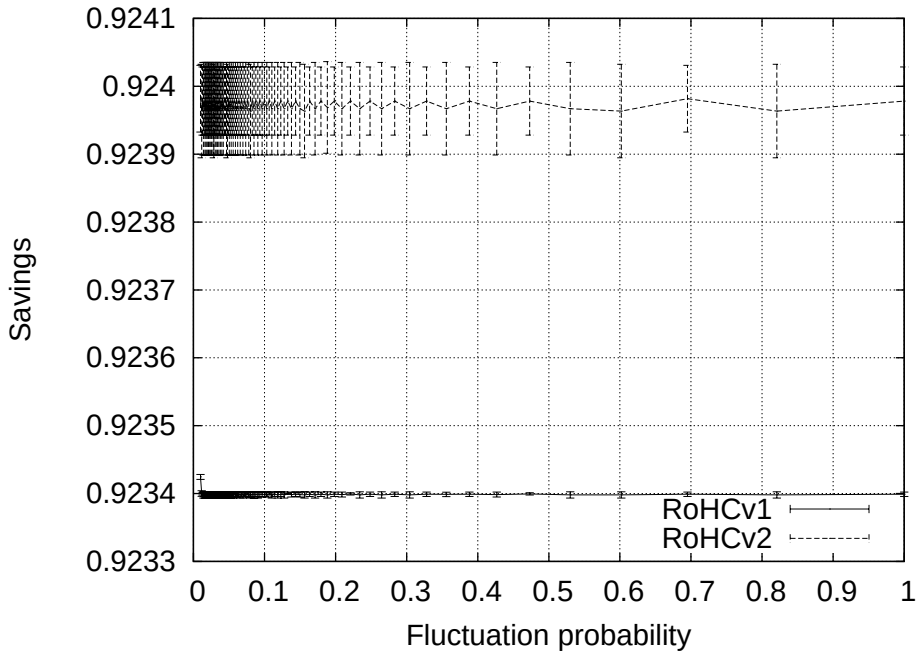


Figure 5.9: Fluctuation of the RTP Marker bit

5.4.3 RTP Sequence Number

In this part the RTP sequence number is evaluated.

On figure 5.10 we can see, that there is a 3% difference in gain between the two versions. The interesting thing is, that v2 has a lower compression ratio for small fluctuation probabilities. This is due to larger packets being used by version 2 when transmitting a sequence number change.

The reason for this is, that in RoHCv2, the internal MSN[3] has the same value as the RTP sequence number number[4]. However, the LSB compression relies on this MSN and the LSB fields need to be updated as well.

Plus, it can be seen on the figure 5.10, that the situation reverses after 0.34 and RoHCv2 reaches better results than v1. This can be interpreted as v2 being more tolerant toward frequent changes in sequence number delta than v1.

[3] Master Sequence Number, it identifies the compressed packet, much like the RTP SN the uncompressed RTP packet.

[4] This is done in order to have 1 less field in the compressed packet.

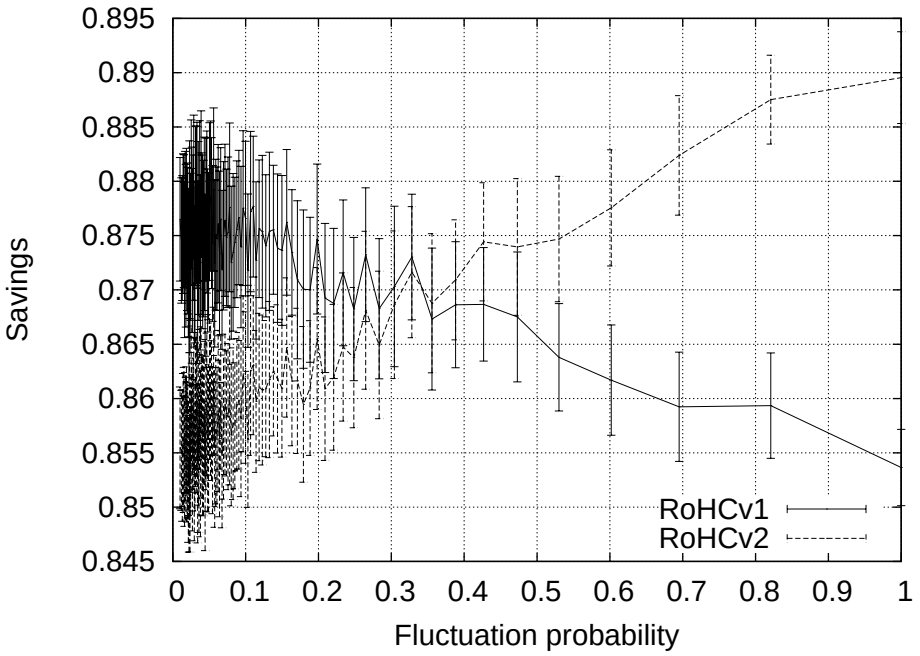


Figure 5.10: Fluctuation of the RTP Sequence number

5.4.4 RTP Timestamp

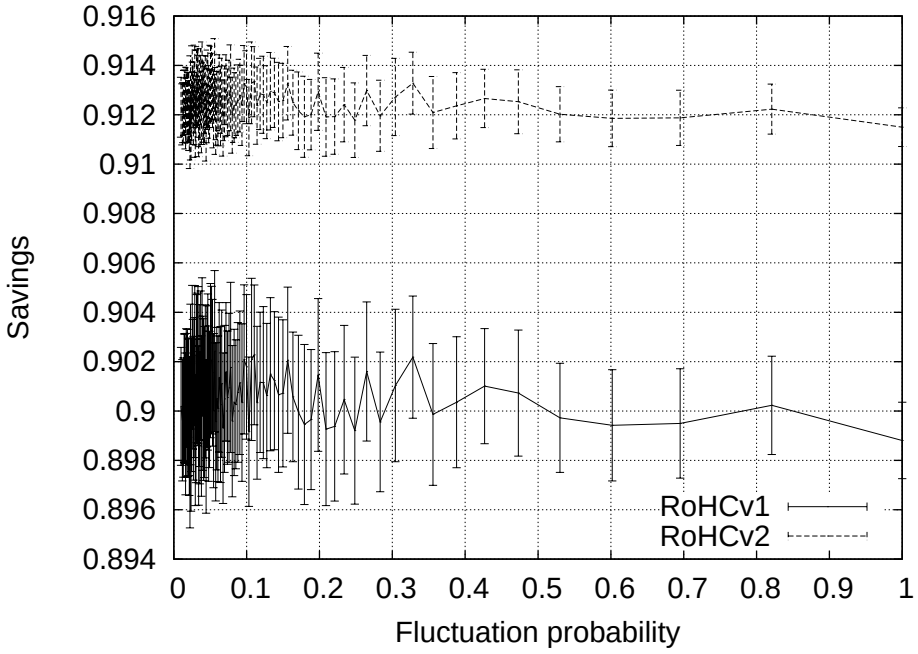
Lastly, the RTP timestamp is evaluated in the non-timer-based compression mode.

On figure 5.11a shows, that both versions have mostly the same curve, while RoHCv2 produces the usual compression ratio. In this case, the fluctuation is – similarly to the previous measurements – is switching between a delta of 1 and 2.

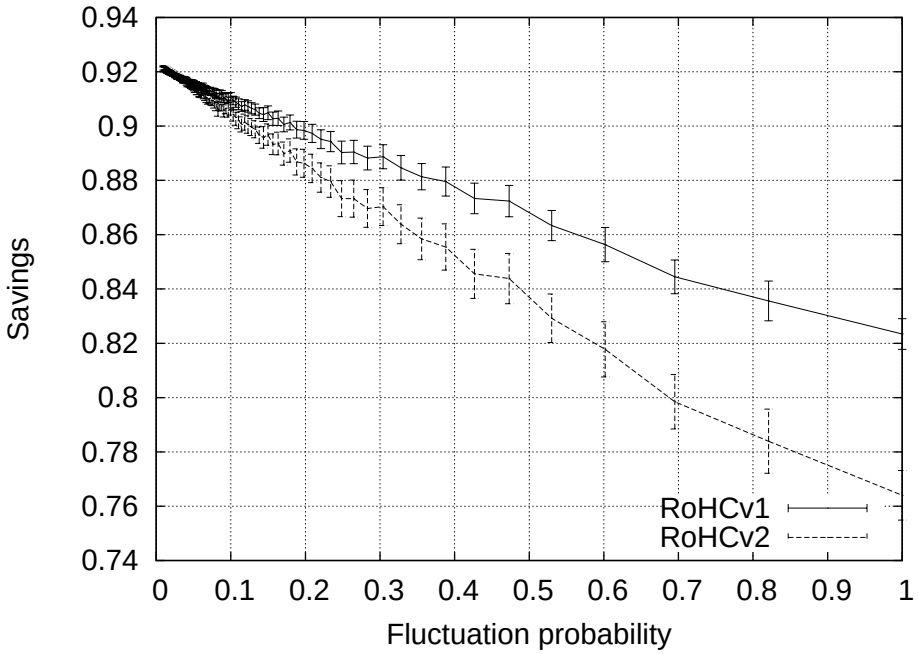
However on figure 5.11b, the delta is increased by the fluctuation probability[5]. This curve confirms one of my earlier observations, that (acticom's) RoHCv2 implementation is less tolerant to varying timestamp behaviours.

Varying timestamps are usually not good for compression to begin with, but in such case, RoHCv1 can perform better by 5-10%!

[5] Here x shows the probability of the RTP timestamp delta increasing by one. On the left side the delta increases rarely, on right side it increases for almost every packet by 1.



(a) Flipping delta



(b) Constantly increasing delta

Figure 5.11: Fluctuation of the RTP Timestamp

Chapter 6

Conclusion and future work

After reading through this text, the Reader should have some basic knowledge about Robust Header Compression, how it works and what compression gains it can provide. With the results presented in the previous chapter, the Reader can also decide which version would suit his or her needs in a – hypothetical – environment.

If an *extensible* and well *adapted* compression is needed, RoHCv1 would clearly satisfy these needs, while v2 provides an *all-in-one* package that is *simpler* to implement, than version 1, but with at least the same performance.

Based on the results shown in chapter 3, we can also state that RoHC clearly benefits us with ~90% compression gain. In almost all test scenarios RoHCv2 showed 4-7% better results than v1. We have also seen, that this comes with some resource costs, especially for v1's decompression and v2's compression.

Looking at the impact field analysis, RoHC's robustness also gives us lots of benefits when the incoming streams are not optimal for compression.

It is important to state again, that all of the measurements were done in users space, which includes the compression (decompression) processes as well. Since normally RoHC is integrated into the kernel's protocol stack, it would be beneficial for future experiments to have a custom kernel module available. Aside from this, it would be interesting to measure the optimistic approach and the feedback performance as well.

Appendices

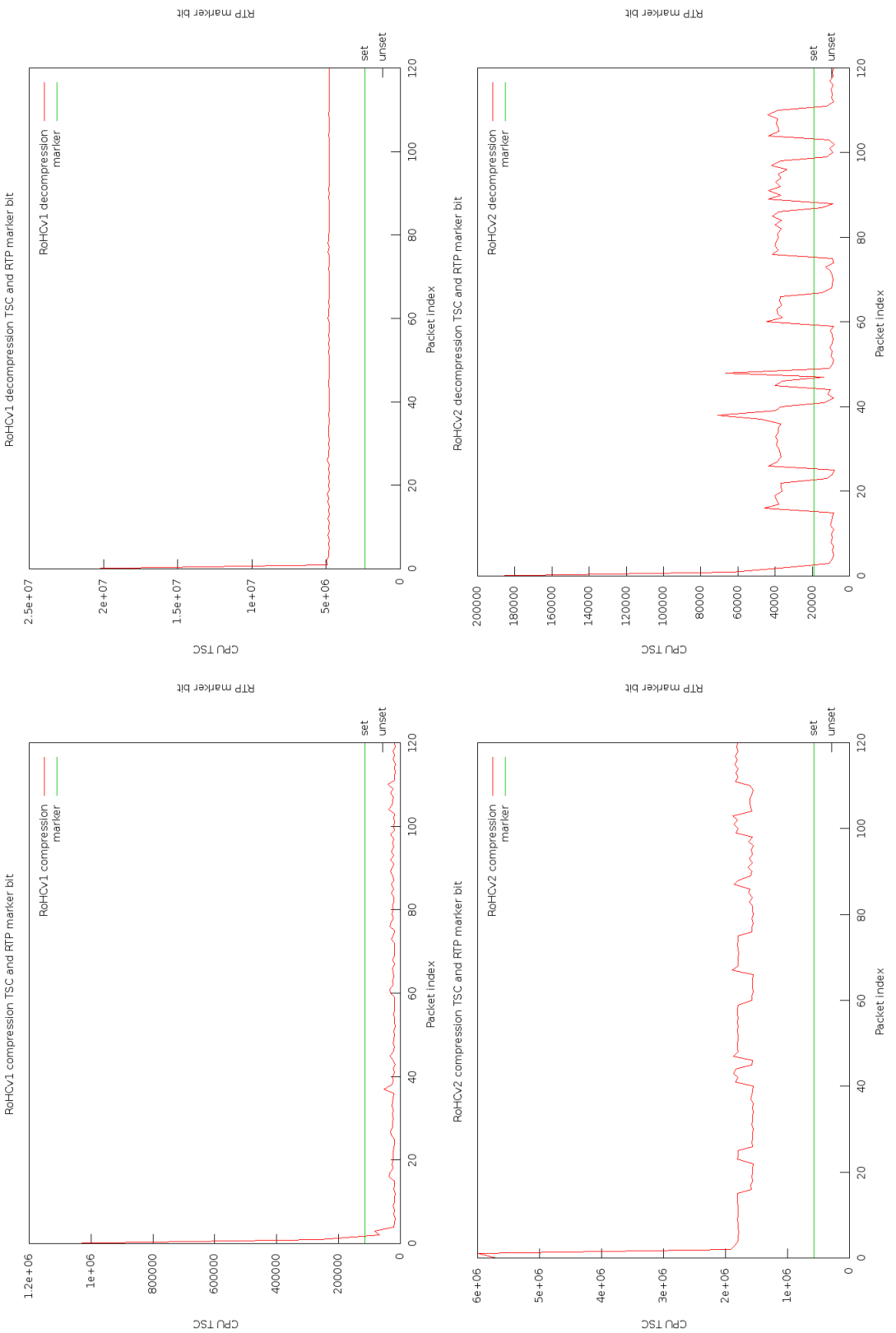


Figure 1: VLC stream complexity for each context with marker-bit

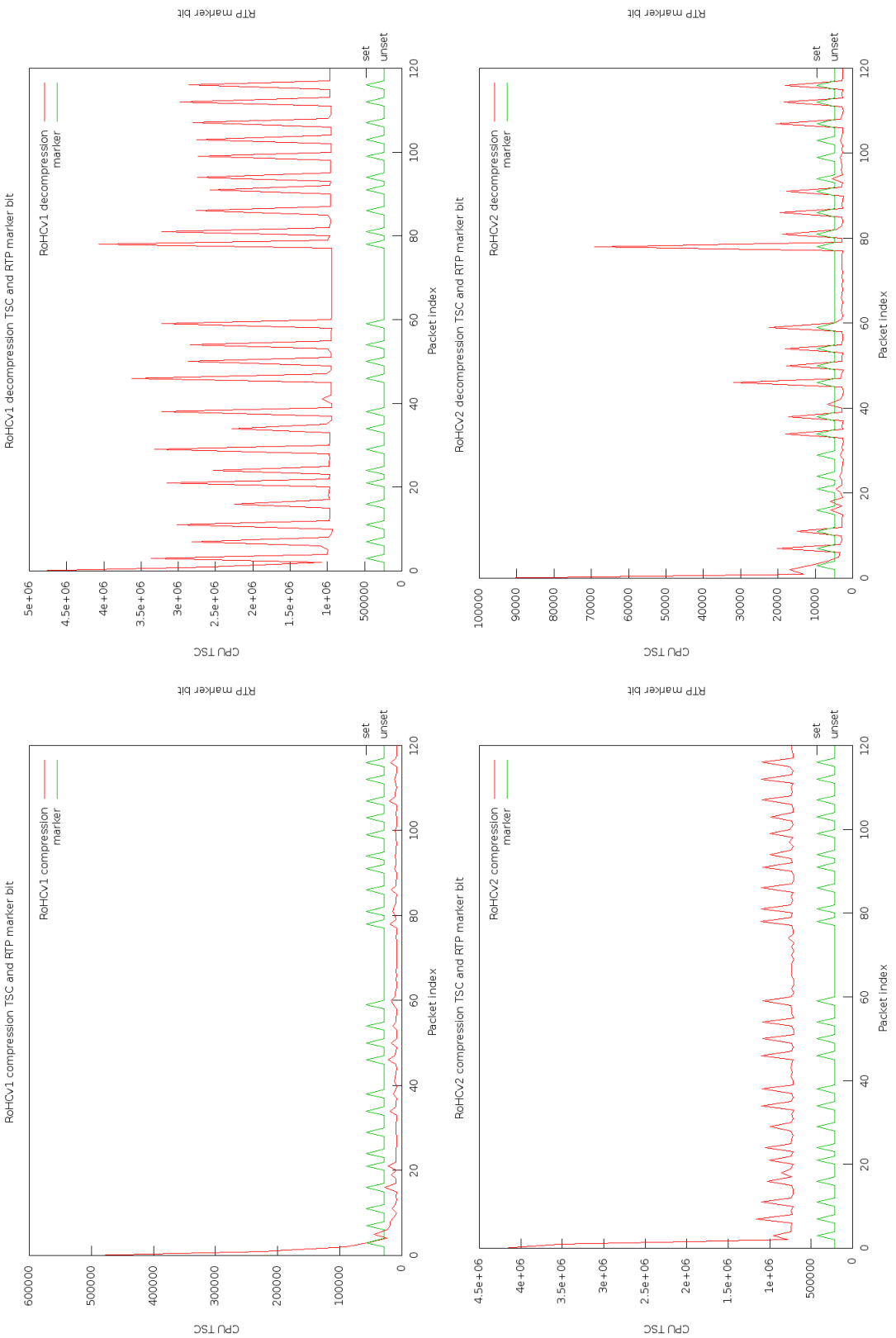


Figure 2: Ekiga stream complexity with with marker-bit for each context

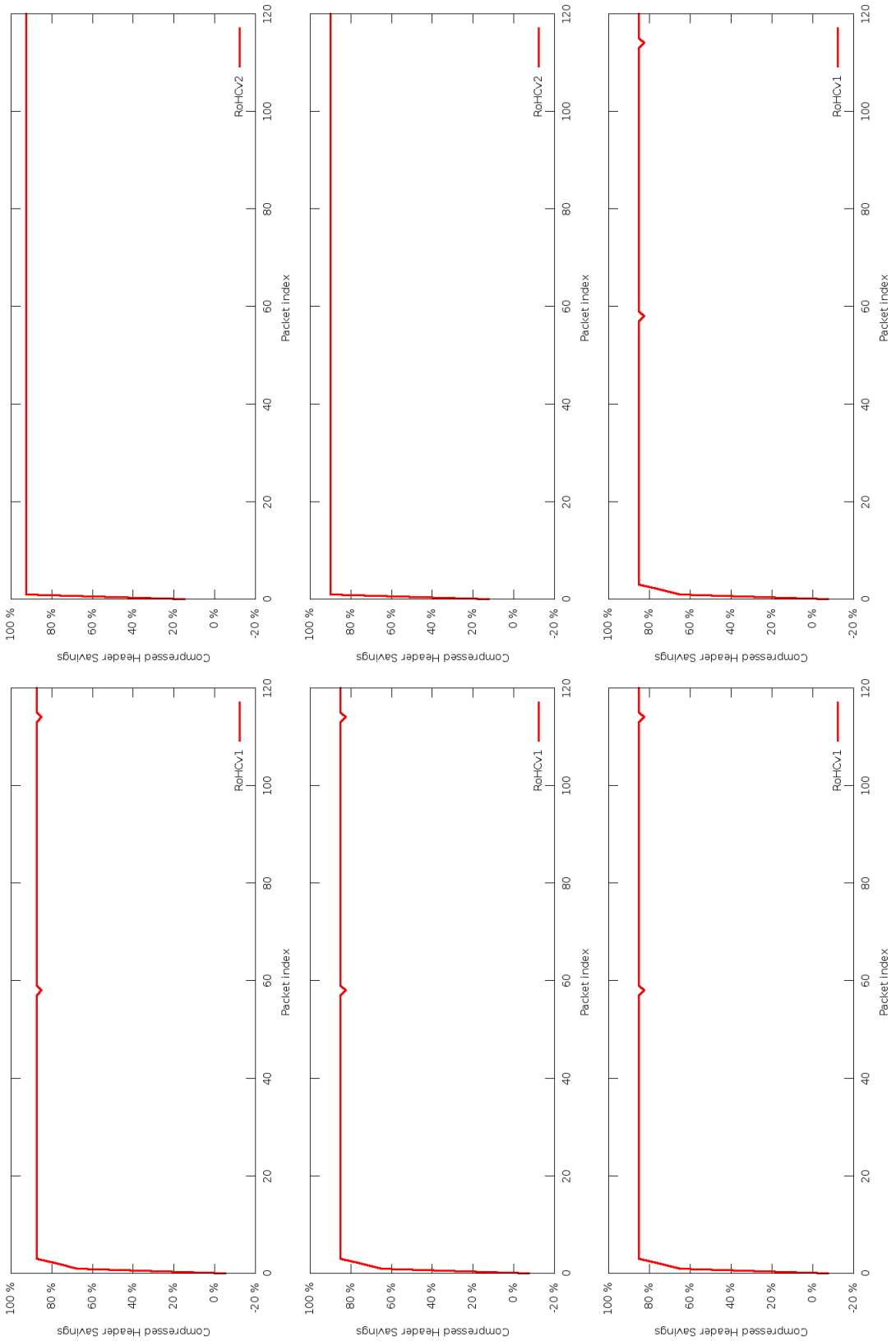


Figure 3: Asterisk call header savings for each context

Bibliography

[1] Way-Chuang Ang, Tat-Chee Wan, Kotaro Kataoka, and Chee-Hong Teh. Performance evaluation of robust header compression (rohc) over unidirectional links using dvb-s testbed. *KEIO SFC JOURNAL*, Vol. 8 No. 2:pp. 33–48, 2008.

[2] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura, and H. Zheng. *RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed*, July 2001. RFC3095.

[3] A. Calveras, M. Arnau, and J. Paradells. An improvement of tcp/ip header compression algorithm for wireless links. *Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the Fifth International Conference on Information Systems Analysis and Synthesis (ISAS'99), IEEE, Orlando, USA*, vol. 4:pp. 39–46, July/August 1999.

[4] A. Calveras and J. Paradells. Tcp/ip over wireless links: Performance evaluation. *48th An-nual Vehicular Technology Conference VTC '98 IEEE, Ottawa (Ontario), Canada*, vol. 3:pp. 1755–1759, May 1998.

[5] W.-T. Chen, D.-W. Chuang, and H.-C.Hsiao. Enhancing crtp by retransmission for wireless networks. *Proceedings of the Tenth International Conference on Computer Communications and Networks*, pages pp. 426–431, 2001.

[6] M. Degermark, H. Hannu, L. Jonsson, and K. Svanbro. Evaluation of crtp performance over cellular radio links. *IEEE Personal Communications*, vol. 7(no. 4):pp. 20–25, 2000.

[7] Frank H. P. Fitzek, Stephan Rein, Patrick Seeling, and Martin Reisslein. Robust header compression (rohc) performance for multimedia transmission over 3g/4g wireless networks, 2005.

[8] V. Jacobson. *Compressing TCP/IP Headers for Low-Speed Serial Links*, February 1990. RFC1144.

[9] L-E. Jonsson and G. Pelletier. *RObust Header Compression (ROHC): A Compression Profile for IP*, June 2004. RFC3843.

[10] G. Pelletier. *RObust Header Compression (ROHC): Profiles for User Datagram Protocol (UDP) Lite*, April 2005. RFC4019.

[11] G. Pelletier and K. Sandlund. *RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite*, April 2008. RFC5225.

[12] G. Pelletier, K. Sandlund, L-E. Jonsson, and M. West. *RObust Header Compression (ROHC): Profiles for User Datagram Protocol (UDP) Lite*, July 2007. RFC4996.

[13] S. J. Perkins and M. W. Mutka. Dependency removal for transport protocol header compression over noisy channels. *International Conference on Communications (ICC), Montreal, Canada*, vol. 2:pp. 1025–1029, June 1977.

[14] Stephan Rein, Frank H. P. Fitzek, and Martin Reisslein. Voice quality evaluation in wireless packet communication systems: A tutorial and performance results for rohc. *IEEE Wireless Communications*, 2004.

[15] Stephan Rein, Martin Reisslein, and Frank H. P. Fitzek. Voice quality evaluation for wireless transmission with rohc, 2003.

[16] M. Rossi, A. Giovanardi, M. Zorzi, , and G. Mazzini. Tcp/ip header compression: Proposal and performance investigation on a wcdma air interface. *12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, IEEE*, vol. 1:pp. A–78 – A–82, September 2001.

[17] M. Rossi, A. Giovanardi, M. Zorzi, and G. Mazzini. Improved header compression for tcp/ip over wireless links. *Electronics Letters*, vol. 36(no. 23):pp. 1958–1960, November 2000.

[18] P. Seeling, M. Reisslein, F.H.P. Fitzek, and S. Hendrata. Video quality evaluation for wireless transmission with robust header compression. In *Information, Communications and Signal Processing, 2003 and Fourth Pacific Rim Conference on Multimedia. Proceedings*

of the 2003 Joint Conference of the Fourth International Conference on, volume 3, pages 1346–1350 vol.3, 2003.

[19] K. Svanbro, H. Hannu, L.-E. Jonsson, and M. Degermark. Wireless real–time ip services enabled by header compression. *Proceedings of the IEEE Vehicular Technology Conference (VTC), Tokyo, Japan*, vol. 2:pp. 1150–1154, 2000.