Unidirectional Robust Header Compression for Reliable Low Latency Mesh Networks

Máté Tömösközi[1,3], Daniel Lucani[2], Frank H. P. Fitzek[1], Péter Ekler[3]
[1]Communication Networks Group, Technische Universität Dresden, Germany
[2]Department of Engineering, Communications Systems, Aarhus University, Denmark
[3]Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary
Email: mate.tomoskozi@tu-dresden.de, daniel.lucani@eng.au.dk, frank.fitzek@tu-dresden.de, ekler.peter@aut.bme.hu

***Abstract*—Next generation use-cases of mesh networks, such as connected vehicles and industrial devices, require low latency transmissions while fulfilling high reliability constraints. However, they also suffer from an increased protocol encapsulation overhead when handling a large number of messages with small payloads. A solution to this problem is to employ *header compression* algorithms in order to reduce the size of the individual protocol headers. Unfortunately, the current state-of-the-art header compression schemes cannot be readily applied to network topologies that contain a combination of multiple-hops and paths, as the compression only works favourably on a peer-to-peer, single-hop basis.**

With the unique combination of *network coding* and *header compression* one can always utilise unidirectional compression with maximum gain. In this paper we introduce and evaluate, for the first time, an integrated network coded header compression solution, which we call *unidirectional Robust Header Compression* (*uRoHC*). We show that one can – proportionally to the logical payload size – double the payload delivery efficiency compared to standard IPv4 and that we achieve results 10–15 % better than that of *RoHCv2* for streams containing 33 bytes of payload.

Keywords: Robust Header Compression, Network Coding, RoHCv2, uRoHC, Mesh Network, Latency, Reliability

I. Introduction

Mission critical applications of wireless mesh networks pose a serious concern when considering the combination of low delays while still providing high reliability. Generally, wireless mesh networks are known for their very dynamic nature where nodes can disappear from the network due to failure of hardware or a change in network accessibility while being mobile, etc. Nonetheless, fifth generation wireless use-cases demand that both of the above properties should be fulfilled as best as possible.

Since we live in a world where billions of devices and more are connected to the same network, one has to provide enough information for the network to be able to deliver to the right recipient at the right time, in the right order and has to make sure that the receiver knows which application to deliver to. This, in turn, requires an encapsulation overhead that can even exceed that of the original data. As seen on Figure 1, as an example, to deliver 25 bytes of real-time information, one needs to provide at least 48 bytes of various protocol headers. And at this point we didn't even touch on the need for recovery from losses prevalent in wireless networks.

When compressing applicable streams with *Robust Header Compression version 2* (*RoHCv2*, defined in *RFC 5225*) one reduces the size of the *Application*, *Transport* and *Internet layer* protocol headers down to a fraction of their original size, in an ideal case, to a mere 1–3 bytes or by $\sim$ 90 %

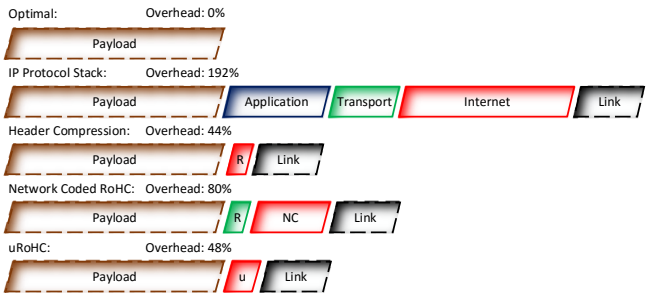

Fig. 1. Example of protocol overhead assuming *RTP/UDP/IPv4* packets of the *Internet Protocol Suit* with 25 byte long payload and 8 byte-link layer header, where *R* is the *RoHC*, *NC* the *RLNC* and *u* the *uRoHC* headers.

of the original header size [1]. The price one pays for this is an increase in computational requirement per sender/receiver pair, a loss of flexibility and an increase in the probability of successfully delivered packets being discarded because of decompressor desyncronisation. With the loss of flexibility we refer to certain assumptions a compressor makes about the structure of the headers and that it should receive uncompressed packets without any errors. Consequently, it has to assume either a direct peer-to-peer link between the compressor and the decompressor or a fast enough bidirectional channel to facilitate a feedback mechanism. A decrease in the amount of data requiring transport additionally yields indirect benefits in shorter network interface activity, which can reduce the energy footprints, as predicted by [2].

One can circumvent desyncronisations and add support for multi-path/multi-hop transmissions by encoding header compressed packets via various *FEC* codes. *Random Linear Network Coding* (*RLNC*, or simply *network coding*) [3] is a coding technique that has been used in many applications including, but not limited to, distributed storage, bandwidth optimisation, encryption, etc. It is a unique rateless code which provides the benefit of recoding. In multi-hop topologies, when a packet goes through a relay, the node normally has to first decode than encode again. This, of course, adds a lot of delay, which is unacceptable in delay critical systems, such as in our case. *Network coding*, instead, can simply recode an already coded symbol, thereby limiting the delay to a single encoding operation, which is significantly faster than decoding and encoding again, as seen in [4]. It also supports *online* and *systematic* codes which avoid buffering and unnecessary coding, both of them being critical for the fulfilment of latency

requirements [5].

Our primary concern with *RLNC* is the added overhead of *network coding* information, which is relative to the employed *Galois-field* and *generation size*s. We refer to the naive combination of *network coding* and *header compression* as *coded Robust Header Compression* or *cRoHC*, as seen in Figure 1. However, the main contribution of this paper is the introduction of an integrated *network coded header compression* technique in a way that minimises the *network coding* overhead while still keeping the complete functionality of *RLNC*, which are beneficial for low latency and high reliability.

Wireless mesh applications usually exhibit a self-organising and autonomous nature as well. Both *Random Linear Network Coding* and *Opportunistic Routing* (*OpR*) have been examined thoroughly in this context [6]. It has been recently shown that a customised compression solution based on *RoHCv2* decreases the size of *OpR* routing messages by at least 50 % with the potential to be even better under certain conditions [7], not to mention that a decrease in message size also decreases the statistical probability of bit errors. *Network coding* for the mesh is also known not only in the context of *OpR*, but also for enabling smaller delays compared to other *Forward Error Correcting* codes (*FEC*), as discussed in [5].

When the integration of *RoHCv2* into a low latency wireless mesh network is considered, one must face either the challenge of long round trip times or the added computational requirements of the compression and decompression procedures at each relay node. In the former case, one increases the overall delay because of an added decompressor feedback loop, which opted mesh enabled solutions to only consider a subset of the header fields for compression (*static* and *implied* fields, see, e.g., [8]) or to provide redundant context information in each packet (so called *Additional Information Container*s or *AIC*s, see, for example, [9]). Albeit both of these cases increase the potential efficiency of the transmissions, they also decrease the maximum achievable compression gain due to the above limiting factors.

Concerning the latter problem, when hop-by-hop compression is advocated, one can – in the worst case scenario – completely disrupt the compression, as the procedure assumes that the input traffic is from a single source, complete and in its original ordering. The compression of a stream with missing packets – which would surely occur after multiple consecutive hops in a wireless mesh – would force the compressor to completely reinitialise the decompressor on a more frequent basis, as it cannot exploit the inherent progression characteristics of the header fields. This, in turn, decreases the compression gain.

In this paper we introduce and evaluate a unique integrated approach for network coded robust header compression called *unidirectional Robust Header Compression* or simply *uRoHC*. This novel take on header compression rends itself naturally to the employment in low latency high reliability wireless mesh networks by exploiting the benefits of rateless codes for reliability and recoding for latency, both of which are inherent properties of *Random Linear Network Coding*. This work also presents for the first time a method for enabling, specifically, version 2 of *Robust Header Compression* for deployment in the mesh. Nonetheless, the introduced concepts can be utilised for a wide range of header compression standards.

In the following Section we describe the employed metrics and our measurement testbed. Section III explores the pros and cons of applicable *header compression* setups while Section IV introduces the solution's design. Following that, Section V evaluates and shows actual measurement results before we conclude in Section VI.

II. Metrics and Evaluation Setup

In order to evaluate the benefit of the individual compression methods, various ways of quantification can be defined, the most straightforward being the amount of bytes saved when compressing. In this case, one simply calculates the ratio of the compressed packet size compared to the original uncompressed packet. This we call *compression gain* or *savings* and can be expressed as:

$$S = 1.0 - \frac{\|T_{co}\|}{\|T_{uc}\|}, \tag{1}$$

where $\|T_{co}\|$ and $\|T_{uc}\|$ are the total transmitted bytes during compression and without compression. Our initial evaluations of compression employs exactly this formula. For further details we refer to [1].

While the above metric is perfect for ascertaining the benefit of compression if losses on the network are of no or little concern, however, it fails to capture the effort it takes to retransmit (recover) missing packets and/or *decompressor context*s. Therefore we define the *payload delivery efficiency* in order to ascertain the effort it takes to deliver a given payload successfully to the recipient under losses as:

$$E = \frac{Rx}{Tx}, \tag{2}$$

which is the ratio between the total received bytes at the sink node that can be delivered to the application layer (e.g., the *logical payload*) without errors (Rx) and the total transmitted bytes including any redundant transmissions (Tx).

Specifically for our testbed, we use an equivalent form of Equation 2 as:

$$E = 1.0 - \frac{\|L\| + \|F\|}{\|T\|}, \tag{3}$$

where E is the efficiency as before, $\|L\|$ is the amount of dropped packets in bytes (excluding redundant packets), $\|F\|$ is the amount of packets which failed decompression in bytes and $\|T\|$ the total transmitted bytes of the given scenario (as sent by the source).

The downside of the above formulae is, however, that the measured values are always relative to the amount of *logical payload* (i.e., smaller *payloads* yield higher *gain* and *efficiency*), therefore, one always have to keep in mind the type of data that is transmitted.

In order to indicate the robustness of the compression with respect to the discarding of compressed packets, we define the *decompression error ratio* as the ratio between the bytes of compressed packets discarded by the decompressor – due the desyncronisation – and the total transmitted compressed packets in bytes:

$$D = \frac{\|F\|}{\|T_{co}\|}. \tag{4}$$

We also evaluate our solution based on the difficulty and reality to compress *IP* traffic. Therefore, we consider the following streams for compression:

- *Easy*: An artificial well-behaved *RTP/UDP/IPv4* stream where fields either stay static or increase by the same delta during the lifetime of the transmission. The payload is fixed to a length of 33 bytes to reflect real *VoIP*.
- *Hard*: Another artificial *RTP/UDP/IPv4* stream where the field deltas change randomly resulting in a hard to compress transmission. The payload length is 33 bytes.
- *GSM*: To enable measurements under common real-world scenarios, we utilised the *Asterisk VoIP* server[1], which connected a fixed desktop client and an *Android* smartphone, both using the *ZoIPer VoIP* client software[2].
- *CAIDA*: We used real-life traces obtained from the *Centre for Applied Internet Data Analysis* (*CAIDA*)[3]. The trace contains more than $12 \cdot 10^3$ *UDP* packets from one flow between a source and a destination.

The streams are replayed locally and compressed (coded) on a computer in the *pcap* format with the commercial implementation of *RoHCv2* from *acticom GmbH.*[4], as well as, the *kodo network coding* library provided by *Steinwurf ApS*[5]. The latter was modified in order to accommodate efficient *header compression*, as described in Section IV.

During our assessment, we additionally measured the *network code*d compression output for each of the above streams under simulated loss rates based on correlated loss probabilities $P_l \in [0.0; 1.0)$ from the *Gilbert-Elliot model* [10]. Losses are presumed to cover both missing and corrupted packets (e.g., an *erasure channel*). The model, illustrated in Figure 2,

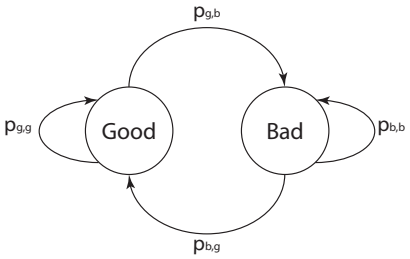

Fig. 2. Gilbert-Elliot channel model with two different loss probability states *g(ood)* and *b(ad)*.

is a two state *Markov-chain*, where (i) no errors occur in the *Good* state and (ii) no packet is conveyed successfully in the *Bad* state. In our measurement setup we streamline in such a way that only one parameter is used, i.e., $p_{g,g} := 1.0 - p_{g,b}$, $p_{b;b} := 1.0 - p_{b,g}$ and $p_{b;b} := 1.0 - p_{g,g}$. We initially assume an error-free channel with $p_{g,b} := 0.0$, which we continuously increase with a delta of 0.01. This ensures that the loss pattern generally contains equally long bursts of packet losses and error-free transmissions.

This model – even if not resembling any specific wireless system – is well suited for the evaluation of the compression efficiency under a lossy *erasure channel*, as it can produce finely tunable bursty loss sequences, which is exactly the condition that would make the compressor lose synchronism with the decompressor, thereby rendering the decompression of any following packet impossible until a context refresh. We employed 100 loss sequences generated for each integer loss probability, resulting in 10000 experiments for each transmission scenario with at least 1000 packets per experiment.

[1]https://www.asterisk.org/
[2]http://www.zoiper.com/.
[3]http://www.caida.org/data/passive/passive_2018_dataset.xml
[4]http://acticom.de/header-compression/robust-header-compression-2/
[5]http://steinwurf.com/products/kodo.html

III. Compression Gain and Robustness

Before we can combine *RoHCv2* with *network coding* we first need to examine the compression configuration space and find the setting that suits our purposes the most. Since *Robust Header Compression* is capable of recovery from losses on the channel, we can evaluate it under specific policies that balance robustness and *compression efficiency*. Because *network coding* is the recovery mechanism that we want to rely on the most, we need to find the appropriate operational mode of *RoHCv2* that can coexist with it. Specifically, we consider the following compressor and decompressor settings:

- *Unreliable*: The compression utilises settings that favour high *compression gain*. Periodic context reinitialisations are set to occur every 1000 packets, *optimistic repeats* are set to 0 and *context confidence* is always ensured. This operational mode by itself alone would require a channel with no or very little losses ($\varepsilon < 1\%$) to function.
- *Static*: The compression settings do not depend on the observed loss rate of the channel. There is one fixed configuration used throughout the transmission session which is supposed to compensate for losses even if they do not occur. This is the most common method employed by the integrators of *RoHC*. In this case the *timeout* is set to 33 packets and the *optimistic repetition count* is 3.
- *Adaptive*: Compression overhead and decompression *context confidence* is subject to the expected loss rate of the channel. The *optimistic repetition count* is increased by 1 after every 10 % increase in loss rate until a maximum of 5 (which is the highest setting for the implementation). Context timeout is also decreased from 1000 to 100, 50, 33, 25, 20 and 16 at the same time. Note that the adaptive configuration can be more fine-tuned as long as one knows the characteristics of the compressible stream, see [11] for further details.

First, we evaluate *RoHCv2* for the *compression gain* in relation to robustness for the above *easy* and *hard* streams. Figure 3 shows the calculated average *gain* under correlated losses. We observe that the *gain* depends very much on the compressor and decompressor settings. We take the *IP*-only *compression gain* (Equation 1) as the baseline, which in this case is 0 since no compression is performed. We see that the *static* performs at exactly 0.4 and the *adaptive* setting decreases the savings after each 0.1 drop in the expected loss probability until it reaches its minimal value at about 0.25.

When we compare them to the respective *decompression error ratio*s from Equation 4 (i.e., robustness to losses) in Figure 4, we see that both of them perform very reliably and, as expected, the *adaptive* setting achieves better robustness in general. When we look at the *unreliable* setting's savings in

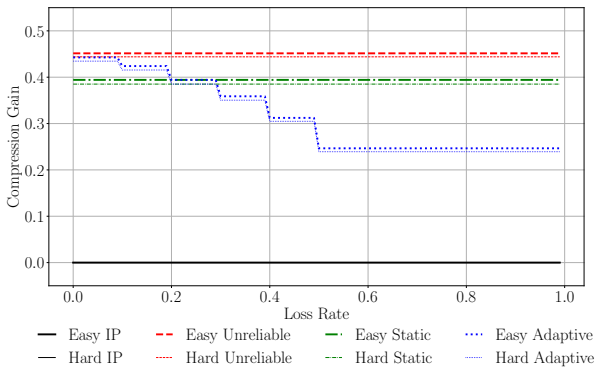

Fig. 3. *RoHCv2 compression gain* for *unreliable*, *static* and *adaptive* settings of the *easy* and *hard RTP* streams over correlated losses.

Figure 3, we see that it is albeit the best at 0.45, however the decompression failures contrast it quite much, where it discards at most 40 % of the successfully delivered packets for the *easy* stream and even 60 % for the *hard* stream in Figure 4. Therefore this setting is not feasible for most scenarios where losses may occur. We confirm that the robustness to compressed packet losses on the channel is inversely proportional to the *compression gain*, e.g., the *adaptive* setting compared to the *unreliable*. We will show later that with *uRoHC* this won't be a concern any more.

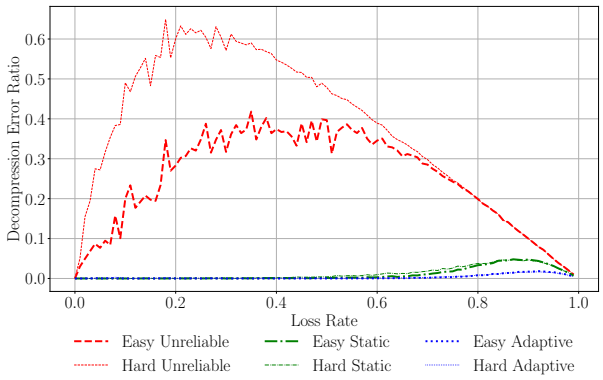

Fig. 4. Failed decompressions with *RoHCv2* for *unreliable*, *static* and *adaptive* settings of the *easy* and *hard RTP* streams over correlated losses.

IV. Solution Design

When considering the combination of *network coding* and *RoHCv2* one can proceed in two ways. The straightforward way involves a hierarchical design where first the compression on the original stream is resolved and the resulting output is fed into the *network coding* algorithm. In this paper we refer to this method as *coded Robust Header Compression*. The benefit of this approach is that the compression and encoding phases are handled independently of each other, which gives a greater flexibility when choosing coding policies (e.g., *full vector network coding*, *online* or *sliding-window network coding*, *progressive shortening* [12]) and coding parameters (*systematic approach*, *generation size*, *field size*, etc.). The downside of this is that the two stages do not share any knowledge between each other which results in some redundant information being transmitted in some packets (for example, original length of packets, position inside the *generation*). This, of course, counteracts any gains from compression.

In order to limit the (de)coding overhead both in relation to *compression gain* and coding complexity, we will always assume the usage of the so called *systematic approach*. When using this method, each original packet is coded together with the other packets having 0 coefficients. Which means that the packet passes through the encoding process unmodified and can be decoded directly without any need for *Gaussian-elimination* or other coded packets. The actual random linear network encoding only takes place when redundant packets are sent over the network to make up for losses.

We also assume that *RLNC* recovers all losses by *network coded* redundancy transmissions on the *packet erasure channel*. Therefore we can be sure that the decompressor receives all compressed packets without fault. With this consideration we can utilise the *unreliable* setting from the previous section, which produces the highest *compression gain* without any concerns for robustness.

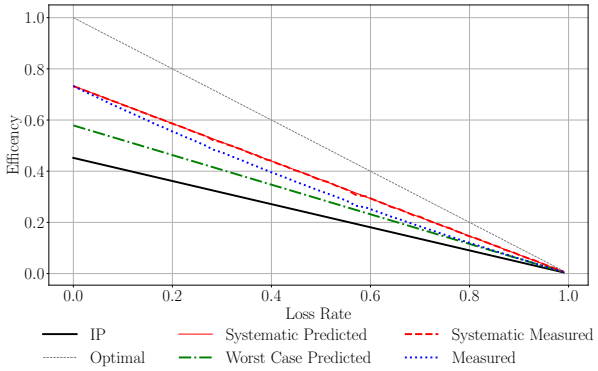

Fig. 5. Predicted *efficeincy* depicting the best (*systematic*) and worst case scenarios using *generation size* 8 for the *easy* stream.

Figure 5 shows the theoretical bounds of this combined header compression and *network coding* assuming the *easy* stream from before. The *efficiency* of the *IP* transmission is represented in black. Both the predicted and measured values show the expected linear fall-off when factoring in losses on the channel assuming 40 bytes of header overhead (20 bytes *IPv4*, 8 bytes *UDP*, 12 bytes *RTP* with 33 bytes logical payload). The *optimal efficiency* presumes that there is a 0 byte overhead, which is the theoretical maximum gain. The *systematic approach* of *network coding*, illustrated in red, passes every packet onto the channel without coding and is therefore ideal for achieving high *compression gain*s. If there are no retransmissions on the channel, this method would coincide with a header compression only scenario assuming a specific static configuration producing 3 byte long compressed headers with some extra overhead introduced by purely *network coding* (in this case 5 bytes). The complete *coded Robust Header Compression* scenario – show in green – depicts the theoretical worst case *efficiency* and the blue one teases the actual measured values. The *network coding* overhead is a combination of the *systematic phase* and the

transmission size of the *redundant* packets (which have a *network coding* header of *generation size* $+1$, given an 8 symbol *generation* and 2^8 *finite field size*).

Our second solution, the *unidirectional Robust Header Compression*, takes a more integrated approach. In this case we opt for a specific *network coding* mechanism, called *seed codes*. When utilising the *seed code*s of the *kodo* library, instead of sending each random coding coefficient in every packet, the decoder derives them by referring back to a common and shared seed used during the initialisation of the uniform random number generating algorithm by the sender. The transmitted seed is normally 4 bytes long in *kodo*, but we instead limit it to 7 non-zero bits. This is applicable since the seed of a random generation algorithm can be any random number and this seed is generated independently as well. The *systematic approach* header is reduced to 7 bits too. The limitation here is that one cannot use larger generations than 128. This is of little concern though, since it was shown in [5] that large *generation size*s have a negative effect on delay. With these considerations we limit the coding header to 1 byte, containing either 7 bits of packet index for the *systematic*, or 7 bits of seed for the *non-systematic phase*. The last bit is reserved for a flag signalling each of the phases (*0x80* for *systematic*, *0x00* otherwise). The transmission of the packet length is also omitted. Since our new header is always non-zero, we can exchange it with the last byte of the coded packet, allowing us to derive the packet size trivially upon reception.

The assumption of *erasure channels* also equips us for the omission of the *UDP checksum* by setting it to zero. This is normally not feasible, because *RoHCv2* indirectly uses it to check for bit errors. Since all *RoHCv2* packet formats contain their own *CRC* values, this is an optimisation step which we are willing to make. In case additional *error-detection code*s are required, one is advised to place them over the coded packets, as they are resolved first. It is also possible to determine errors based on the structure of the coded symbols, which is an ongoing research effort of one of the authors.

V. Evaluation

We first evaluate the naive *coded header compression* and than compare it to the more integrated *uRoHC* approach. Figure 6 shows the *cRoHC efficiency* using the *full vector network coding* of *kodo* with respect to various *generation size*s (G). We observe that we gain the highest *efficiency* with the smallest *generation size* ($G = 8$), maximising at a value ~ 0.75, which decreases toward zero as the loss rate increases. Since we employ the *systematic approach*, the larger *generation size*s also start at the same value, which produces uncoded packets with a small *network coding* overhead. However, as more lost packets have to be recovered, the ratio between *systematic* and redundant coded packets slowly shifts toward the latter. Since *non-systematic* packets contain a larger overhead than *systematic* ones, the first slowly starts to outweigh the second and the *efficiency* starts to drop more sharply. Also, because the overhead of *non-systematic* packets is mostly determined by the size of the *encoding vector*, a higher *generation size* means more coefficients need to be transmitted, which decreases the *payload delivery efficiency* even further.

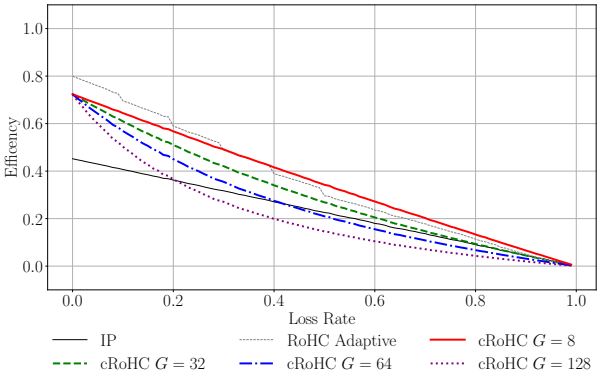

Fig. 6. *Efficeincy* of *cRoHC* with actual measurements using various *generation size*s with the *hard* stream.

In Figure 7 we present the distribution of the average coded packet sizes for the above scenario for various *generation size*s. We see that the smallest *network code*d packets are produced when the *generation size* is very small ($G \in \{8, 16\}$), in which case about half of the transmission contains smaller packets than that of the original *IP* transmission and the other half is at maximum the size of the original. For higher *generation size*s the *compression gain* deteriorates even further and at $G = 128$ it even produces packets with sizes 200 % or larger than that of the original ones.

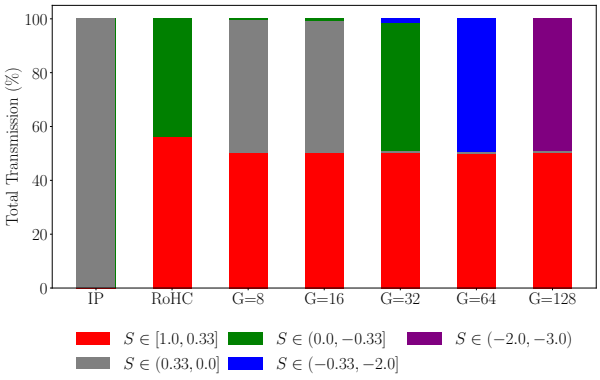

Fig. 7. Average packet size distribution and *compression gain* of *cRoHC* using various *generation size*s with the *hard* stream.

In case of the modified *seed code* based header compression, *compression gain*s and robustness to errors stays relatively constant, as attested by Figure 8. *uRoHC* performs better than *cRoHC* in all cases thanks to the higher integration of both schemes. It also exhibits a more stable fall-off since the length of the *network coding* overhead is no longer defined by the number of the coefficients. Figure 8 also shows a gain over traditional *RoHCv2*, which is due to our consideration from Section IV for the omission of the *UDP checksum*.

Figure 9 represents the packet size distribution of *uRoHC*. As expected, the size of the produced coded packets does not depend on the *generation size* because the transmission of the coefficients is replaced by a single constant seed.

Real life traffic evaluation shows similar results with the same steady linear-like fall-off observed in Figure 8. For the

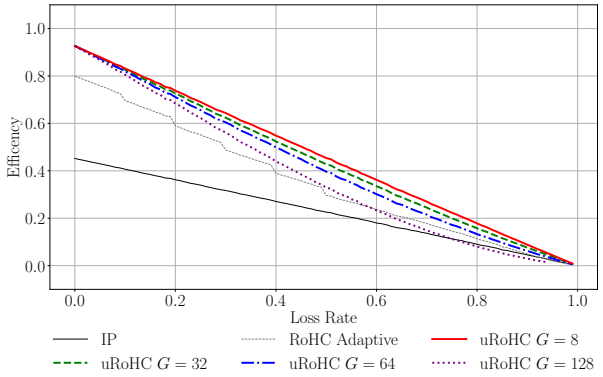

Fig. 8. *Efficeincy* of *uRoHC* with actual measurements using various *generation size*s with the *hard* stream.

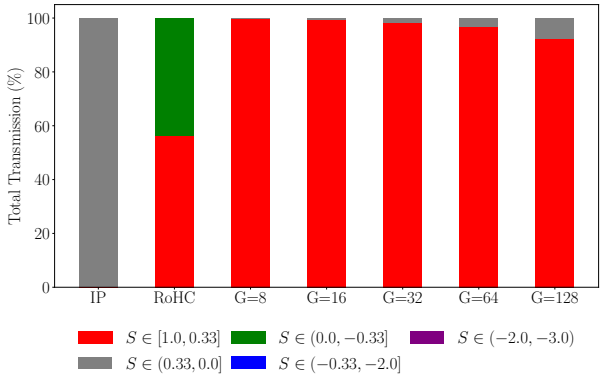

Fig. 9. Average packet size distribution and *compression gain* of *uRoHC* using various *generation size*s with the *hard* stream.

GSM stream there is an *efficiency* gain of ~ 15 % over standard *RoHCv2* compression and for the *CAIDA* there is only a minor gain of ~ 2 %, which is due to the *UDP/IP* compression being more robust to compressed packet losses as the headers contain mostly *static* fields and that the compression already provides peak gains at ~ 91 %.

VI. Conclusions

5G mesh network use-cases that require low latency transmissions and high reliability could suffer from a protocol encapsulation overhead that can outweigh the amount of data one is supposed to send. A solution would be to employ header compression algorithms in order to reduce the size of the individual protocol headers. *Robust Header Compression* is an already established standard which tackles *IP*-based compression quite well, however, the current scheme does not allow deployment in wireless mesh networks as the compression works on a peer-to-peer, single-hop basis.

In this paper we introduced and evaluated, for the first time, an integrated network coded header compression solution called *unidirectional Robust Header Compression*. With this combination one only needs to provide compression functionality on both endpoints of a transmission. We show that one can cut the payload delivery overhead in half for *RTP* transmissions even in cases when standard *RoHCv2* would struggle and potentially fail, and that we can even improve on *RoHCv2* under the assumption of *erasure channel*s.

Our ongoing research focuses on decreasing the coding overhead even further by applying *progressive shortening* [13] to header compressed packets and evaluating it in a real mesh environment, as well as, generalising the header compression scheme itself for the application to arbitrary protocol headers.

Acknowledgments

This work was supported by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC), the János Bolyai Research Fellowship of the Hungarian Academy of Sciences and the SCALE-IoT Project (Grant No. 7026-00042B) granted by the Danish Council for Independent Research, the Aarhus Universitets Forskningsfond Starting Grant Project AUFF-2017-FLS-7-1, and Aarhus University's DIGIT Centre.

References

[1] M. Tömösközi, P. Seeling, P. Ekler, and F. H. P. Fitzek, "Performance evaluation of network header compression schemes for UDP, RTP and TCP," in *Periodica Polytechnica Electrical Engineering and Computer Science, Online First (2016) paper 8958*, 2016.

[2] M. Tömösközi, P. Seeling, P. Ekler, and F. H. P. Fitzek, "Robust header compression version 2 power consumption on android devices via tunnelling," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 418–423, May 2017.

[3] T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, pp. 4413–4430, Oct 2006.

[4] M. Tömösközi, F. H. P. Fitzek, D. E. Lucani, M. V. Pedersen, P. Seeling, and P. Ekler, "On the packet delay characteristics for serially-connected links using random linear network coding with and without recoding," in *Proceedings of European Wireless 2015; 21th European Wireless Conference*, pp. 1–6, May 2015.

[5] M. Tömösközi, F. H. P. Fitzek, D. E. Lucani, M. V. Pedersen, and P. Seeling, "On the delay characteristics for point-to-point links using random linear network coding with on-the-fly coding capabilities," in *European Wireless 2014; 20th European Wireless Conference*, pp. 1–6, May 2014.

[6] S. Pandi, S. Wunderlich, and F. H. P. Fitzek, "Reliable low latency wireless mesh networks from myth to reality," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–2, Jan 2018.

[7] M. Tömösközi, I. Tsokalo, S. Pandi, F. H. P. Fitzek, and P. Ekler, "Header compression in opportunistic routing," in *European Wireless 2018; 24th European Wireless Conference*, pp. 1–6, May 2018.

[8] D. Kidston, W. Chae, and H. Rutagemwa, "Multihop multicast header compression in manets," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–7, May 2018.

[9] F. H. P. Fitzek, T. K. Madsen, P. Popovski, R. Prasad, and M. Katz, "Cooperative ip header compression for parallel channels in wireless meshed networks," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 2, pp. 1331–1335 Vol. 2, May 2005.

[10] P. Sadeghi, R. A. Kennedy, P. B. Rapajic, and R. Shams, "Finite-state markov modeling of fading channels - a survey of principles and applications," *IEEE Signal Processing Magazine*, vol. 25, pp. 57–80, Sep. 2008.

[11] M. Tömösközi, P. Seeling, P. Ekler, and F. H. P. Fitzek, "Regression model building and efficiency prediction of rohcv2 compressor implementations for voip," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec 2016.

[12] M. Taghouti, D. E. Lucani, M. V. Pedersen, and A. Bouallegue, "Random linear network coding for streams with unequally sized packets: Overhead reduction without zero-padded schemes," in *2016 23rd International Conference on Telecommunications (ICT)*, pp. 1–6, May 2016.

[13] M. Taghouti, M. Tömösközi, M. Höweler, D. E. Lucani, F. H. Fitzek, A. Bouallegue, and P. Ekler, "Implementation of network coding with recoding for unequal-sized and header compressed traffic," in *2019 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE WCNC 2019)*, (Marrakech, Morocco), Apr. 2019.