Initial Concept of an Oracle-Structured Stream Compression Protocol for Arbitrary Network Flows

Máté Tömösközi[1,2], Mingchuan Luo[1], Frank H. P. Fitzek[1], Péter Ekler[2]
[1]Communication Networks Group, Technische Universität Dresden, Germany
[3]Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary
mate.tomoskozi@tu-dresden.de, Mingchuan.Luo@mailbox.tu-dresden.de, frank.fitzek@tu-dresden.de, ekler.peter@aut.bme.hu

***Abstract*—Next generation use-cases of wireless networks require a great deal of flexibility in order to adopt to the constantly changing state of innovation and to accommodate future application requirements. *Header compression* has been an ever present solution since the advent of wireless networks and the most current version of it, *Robust Header Compression* (*RoHC*), has seen a widespread adoption in *Long Term Evolution* (*LTE*) cellular networks.**

Recent research has mostly focused on the integration and enhancement of *RoHC*, instead of advancing the core concept of the compression. In this paper we present for the first time a novel design that can tackle the compression of arbitrary packet streams regardless of the employed protocols and the transmitted data. We present our initial findings for an error-free scenario with various simulated and real-life packet streams and show that even with the absence of design time knowledge about the packet structures, one can compress a significant part of the streams, yielding compression gains for *IP* packets up to 90 %, as an example.

Keywords: Header Compression, Network Flows, Stream Compression, Adaptability, Internet of Things, Machine Learning, Network Function Virtualisation

I. Introduction

As our Internet grows and becomes even more complex catalysed by the adoption of numerous next generation wireless network applications, such as the *Internet of Things* and *Vehicle-to-Infrastructure*, billions of devices would need to communicate in the same connected digital environment. To facilitate this, one has to provide sufficient metadata which tackle addressing, routing, synchronisation and error recovery concerns, among other things. However, this requires an encapsulation overhead that can even exceed the size of the logical payload.

The *Internet Protocol* (*IP*) has been widely adapted as the main *network layer* protocol. Typically, *IP* packets contain a protocol encapsulation overhead from higher layers of the network protocol stack as well, which are prepended to the logical payload (i.e., the binary information that needs to be transmitted). For mobile multimedia settings, a common protocol combination is *RTP/UDP/IP*, accounting for 40 bytes with *IPv4* and 60 bytes with *IPv6*.

Having stated this, network developers cannot assume that in our rapidly advancing age one has the perfectly fitting programming library (*function*) ready for every application which we did not even invent yet. One of the main limiting factors for the employment of *header compression* solutions for *fifth generation* networks is the constricting nature of these compression designs. For example, it has been recently shown that a customised compression based on *Robust Header Compression version 2* (*RoHCv2*) decreases the size of *Opportunistic Routing* routing messages in *mesh networks* by at least 50 % with the potential to be even better under certain conditions [1], not to mention that a decrease in message size also decreases the statistical probability of bit errors.

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]. The impact of *RoHC* on media performances have found that *header compression* cuts the required bandwidth in half for voice transmissions (*GSM*) and improves the overall voice quality (see [3] and [4]). Later the authors additionally discovered that video quality can be enhanced as well (see [5]).

However, since there are no standardised compression solutions for the aforementioned routing scenario, one would need to develop unique implementations for every use-case. Since the available expertise in the area of *header compression* is very limited, this could prove quite costly and could deter future (protocol) applications – for example *QUIC* [6] – from employing compression.

The first *IP* header compression scheme was the *Compressed Transport Control Protocol* (*CTCP* or *VJHC*). It was designed by Van Jacobson [7] and it exclusively focuses on the compression of the *TCP* protocol. *CTCP* combines *TCP* and *IP* headers together for better results and lower complexity, while the compression algorithm itself employs *delta coding*. The main benefit of *CTCP* – besides its novel concept – is the high compression ratio. Unfortunately, it is very susceptible to bit errors, as it was designed to operate on the wired networks of its time, which results in the loss of synchronism with the decompressor and the consequent discarding of successfully received compressed packets. Moreover, it lacks any internal error detection scheme and it relies on the protection mechanism of other protocols. *Robust Header Compression* (*RoHC*), specifically version 1 of *RoHC*, which was introduced in [8], was modelled on the concept of extensibility with various profiles that were added later and was incorporated into the *3GPP-UMTS* and *WiMAX* networks. However, the newest header compression standard, version 2 of *Robust Header Compression*, defined in [9], opts for simplicity in design over extensibility. It aims to be more robust under similar network conditions, while increasing compression gain. Consequently, it had gained widespread adoption in the *Long Term Evolution* (*LTE*) networks next to *RoHCv1*.

Although both versions of *RoHC* achieve around 80-90 % in gain (see [10]), a lack of knowledge about the structure of the packet data – including any headers – normally necessitates to omission of *header compression* altogether. Since

the various compression standards rely on fixed structures inside the headers, which purpose can only be determined at *design time* of the compression algorithm, one would need to construct separate compression schemes for each to accommodate compression of that particular stream. In the world of massive heterogeneity and standardisation of our Internet, this is, of course, a quite impregnable hurdle. However, with the advancement of the increasing data processing capabilities of modern hardware, one can utilise such emerging concepts as *Network Function Virtualisation*, *Artificial Intelligence*, as well as *Machine Learning*.

In this work we advocate the employment of an *oracle* in order to determine the structure of the packet flow and to enable their future compression. We call this idea and the accompanying compression scheme *Oracle-Structured Stream Compression* or *O2SC*. An *oracle*, or an *oracle machine* in computational theory terms, is an abstract entity which can make decisions of higher complexity. It acts as a *black box*, that given a certain input, produces a corresponding output in a single operation. In our case, we delegate the *design time* determination of the packet structure to an *oracle* entity and we construct a compression scheme that, given an arbitrary set of packets, can query the aforementioned *oracle* and interpret its solution to determine how to compress the given stream.

This in turn enables the compression of packet flows that would otherwise not be compressible at all (basically, any non-*UDP/IP*, *TCP/IP* stream), or only partially by current standards (various extensions, and/or header combinations do not have standardised compression profiles).

Note, that this treatment of our concept is initial and some of the applied considerations can be improved on. We base our design on years of first hand experience with *header compression*, especially with *RoHCv2* defined in [9]. As of yet, we do not consider multiple flows, nor losses, as well as packets of varying sizes. However, none of these provide serious design issues and can be solved with relative ease, which we plan to publish in the near future. Moreover, in this paper we have not yet considered the integration of some of our more forward looking concepts, such as *network code*d compressed packets [11], as well as *reliable base proposal*.

In the following Section we discuss the overall design of this *oracle-structured* solution. In Section III we evaluate the compression for specific header combinations in the *IP* protocol stack. After that, Section IV presents various measurements with real-life packet flows, such as the control messages of a *Franka Emika* robot arm. Finally we conclude in Section V.

II. Compression Design

In order to facilitate the compression of any set of consecutive network packets, we define two entities that sit on opposite sides of a (direct) channel: the *compressor* and the *decompressor*. The main goal of the compressor is to guarantee that the decompressor is in synchronism with it and is aware of every change in the compressed flow, while making sure that only the bare minimum of data is transmitted. The decompressor in turn has to verify each incoming packet and determine whether it can safely update its context without corrupting it for the future recovery and decompression of messages.

Our main metric of evaluation throughout this paper will be the *compression gain* or *savings*, which 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 *RoHC* employs exactly this formula and expresses the amount of compression that can be achieved for a given packet. For further details we refer to [10].

A. The Oracle

Since we utilise the *oracle machine* concept for the determination of the *design time* compression structure, we need to construct its input and output rigorously. First of all, we specify the available knowledge of the *oracle* as:

$$\mathbf{P}_n = \{p_1, \ldots, p_n\}, \tag{2}$$

where $p_i \in \mathbb{F}_{2^8}^m$, $n, i \in \mathbb{N}$, $1 \leq i \leq n$. $\mathbb{F}_{2^8}^m$ a *finite field* representing a packet of length m bytes. $\mathbf{P}_n$ in turn is a set of n available uncompressed messages (i.e., a history of previous packets).

In turn, we define the *oracle* as a function

$$f: \quad \mathbb{F}_{2^8}^m \times \mathbf{P}_n \rightarrow \mathbb{F}_{2^2}^m. \tag{3}$$

The corresponding output of the function is of $\mathbb{F}_{2^2}^m$, which is the same length as the packet and represents a certain compression pattern for a given packet from $\mathbb{F}_{2^8}^m$

The output pattern consists of a series of 2-bit flags signalling whether a given field is *static*, *sequential* or *random*. These in turn are defined as follows:

- *Static*: These fields have constant, very rarely changing value between packets. Their transmission should generally be done once per the lifetime of the flow. Examples of them include: *IP source* and *destination address*es, *UDP source* and *destination port numbers*, *RTP payload type*s.
- *Sequential*: Fields of this type change in a well defined manner, meaning that the delta between consecutive packets stays relatively constant throughout the transmission. Fields of this class can be: *IPv4 identification*, *RTP timestamp*, *RTP sequence number*, etc.
- *Random*: Lastly, these fields do not exhibit any specific pattern, and change in an erratic way. They include various *checksum*s, or an already compressed logical payload. Note, that some of these fields could be deduced in the knowledge of the underlying algorithms, e.g., *CRC*s and *checksums*, however, the current *oracle* design does not consider this.

For this article we employed a model trained with a basic *logistic regression* as the *oracle*, and it provided a generally accurate estimation of the packet structure. An obvious drawback of this approach is that the compression cannot commence until a model is created. This either requires the buffering of numerous packets before compression or the offline generation of the model. Our current research efforts target the determination of the right *classification*, one that finds sufficient balance between accuracy and prediction complexity. An in-depth evaluation of these techniques is, however, out-of-the-scope of this contribution.

B. Compressed Packet Pool

We define the following compressed packet pool, which currently contains 4 packets, one *non-sequential* (*Non_seq*) and three *sequential* ones (*Seq_0*, *Seq_1*, *Seq_2*).

The *non-sequential* packet type is used to initialise the decompressor context, as well as to transmit new compression patterns. Table I shows the field allocation of this particular packet. In general, all packet types contain one byte of *flags* as a bit field. In this version of *O2SC*, we reserve two bits to distinguish the four packet types and one bit to signal if the *pattern length* field is two bytes long instead of one. The rest of the bits are currently unused and set to zero. In the future we will need these to add *context identifier*s for the separate *compression context*s, as well as to accommodate more packet types.

TABLE I
THE FIELD STRUCTURE OF THE NON-SEQUENTIAL PACKET TYPE.

1	1-2	$\mathcal{L}_{pattern}$	$\|\|st\|\|$	$\|\|sq\|\|$	$\|\|r\|\|$
Flags	Pattern Length	Pattern	Static	Sequential	Random
Bit field	Uncomp.	Comp.	Uncomp.	Uncomp.	Uncomp.

The second field of the *non_seq* packet is the actual pattern length. Since at the maximum one can use two bytes to signal the pattern length, at the moment the compression only supports up to 2^{18} octets. This allows us to cover only a small portion of the *IPv6 jumbogram*s, however, the *compression gain* for packets on this scale would possibly be negligible due to the enormous payload size. The 2^{18} length is achieved via *pattern compression*, which packs every 2-bit value sequentially. We use $x \in [0, 1, 2]$ to signal each of the 3 field types. In case a given pattern characteristic repeats more than 4 times (e.g., the *static* flag $0x00$ is repeated at least 4 more times), we use the fourth flag value ($0x03$) to signal the number of repeats up till 32. If the number of repeats is still larger than this, new instances of the same field is created until all values are referenced. The compressed size of the pattern in bytes with a single sequence of repetition can be obtained by the following formula:

$$\mathcal{L}_{pattern}(m,k) = \left\lceil \frac{2 \cdot (m-k) + 10 \cdot \lceil k \cdot 33^{-1} \rceil}{8} \right\rceil, \quad (4)$$

where m is the total number of bytes in the packet, k is the number of times a certain value repeats. In essence, Equation 4 counts number of "non-compressed" pattern bits and the number of "compressed" ones – accounting for repeats longer than 33 – in bytes. For example, a patter with length 100 which contains the repetition of one of the values 72 times would yield with $k = 2$, $m = 100$, 11 bytes. Without compression this would cost $m \cdot 4^{-1} = 25$ bytes.

The rest of the packet is made up of the uncompressed value of each field packed consecutively in groups of *static*, *sequential* and *random* fields. The complete *non_seq* packet size is therefore obtained as:

$$\mathcal{L}_{non\_seq}(m,k) = 1 + \mathcal{L}_{pattern}(m,k) + m. \quad (5)$$

Using the above example, the complete packet size would be 112 bytes, which is 12 bytes more than the original packet. This is however not an issue, since the majority of the compressed traffic will use *sequential* packets.

The *seq_2* (*delta*) packet type only transmits the *random* fields as uncompressed values, see Table II. This packet yields the maximum *compression gain* as the rest of the fields are inferred from the *decompressor context* based on the last two received packet's difference for the given *sequential* fields (i.e., the current field *delta*). This in turn necessitates that the decompressor receives at least two consecutive compressed packet before a *seq_2* packet and that there is no fluctuation in the field deltas at all. In case one field's delta changes, but the rest does not, this packet type cannot be used. In turn, a prefect *oracle* would flag such fields as *random* if the trade-off of the extra *random* bytes is justified. The compressed packet length would be calculated as follows:

$$\mathcal{L}_{seq\_2} = 1 + ||r||, \quad (6)$$

where $||r||$ is the number of *random* fields in the flow. Continuing the previous example, given 10 *random* fields, $\mathcal{L}_{seq\_2}$ would be 11 bytes, which is an 89 % *gain*.

TABLE II
THE FIELD STRUCTURE OF THE SEQUENTIAL_2 (DELTA) PACKET TYPE.

1	$\|\|r\|\|$
Flags	Random
Bit field	Uncompressed

In case at least one of the above *delta*s change, the compressor can fall back to the *seq_1* (*lsb*) packet (Table III). This one transmits each of the *sequential* fields with *LSB* compression applied to them. The *LSB*s (which stands for *Least Significant Bit*) can transmit a small amount of change in the fields, which is bounded by the employed number of bits. This technique is used throughout the two versions of *RoHC* and defined in e.g., [9]. For further details we refer to the *RFC*s. We employ *LSB*s with parametrisation $k = 4$, $p = 1$, meaning that we use 4 bits per field, and can represent a change in delta between $[-1, 14]$. We are considering making the choice of k and p flexible, however, for this version we stay with these values. Consequently, the length of this packet type can be obtained as:

$$\mathcal{L}_{seq\_1} = 1 + \lceil ||sq|| \cdot 2^{-2} \rceil + ||r||, \quad (7)$$

where $||sq||$ is the number of *sequential* fields. Since each *sequential* field is represented by 4 bits we apply the 2^{-2} multiplier to their original size. In case our experimental packet contains 18 bytes of *sequential* fields, the length of this packet would be 16 bytes, which yields a *gain* of 84 %.

TABLE III
THE FIELD STRUCTURE OF THE SEQUENTIAL_1 (LSB) PACKET TYPE.

1	$\|\|sq\|\| \cdot 2^{-2}$	$\|\|r\|\|$
Flags	Sequential	Random
Bit field	LSB4	Uncompressed

If none of the above conditions can be fulfilled for the *sequential* fields, the compressor can still use the *seq_0* (*random*) packet type from Table IV before reverting back to *non_seq* packets.

TABLE IV
THE FIELD STRUCTURE OF A SEQUENTIAL_0 (RANDOM) PACKET TYPE.

1	$\|\|sq\|\|$	$\|\|r\|\|$
Flags	Sequential	Random
Bit field	Uncompressed	Uncompressed

In this case, all of the *sequential* fields are transmitted uncompressed, resulting in:

$$\mathcal{L}_{seq\_0} = 1 + ||sq|| + ||r||. \tag{8}$$

For the above example, the packet size would be 29 bytes, yielding a 71 % *gain*.

We obtain the compressed size of the complete transmission with a single pattern simply as the number of each sent packet type multiplied by their length:

$$\begin{aligned}\mathcal{L}(m,k) = \quad & a \cdot \mathcal{L}_{non\_seq}(m,k) + b \cdot \mathcal{L}_{seq\_0} \\ & + c \cdot \mathcal{L}_{seq\_1} + d \cdot \mathcal{L}_{seq\_2},\end{aligned} \tag{9}$$

where a, b, c, d are the number of times each corresponding packet type occurs during the transmission. Note, that we do not include a thorough discussion about the theoretical analysis of the achievable compression gain as the main focus of this paper is to prove the feasibility of our concept via practical evaluations. Nonetheless, an in-depth study would be quite intriguing – which we will publish as soon as we finalise the compression design – since the achievable gains significantly depend on various parameters, such as the ratio of static fields to sequential and random ones, as well as the probability of incorrect pattern predictions by the oracle, etc.

C. Compressor States

At the moment the compressor has three available states as depicted in Figure 1. These act like a *finite state machine* and enforce certain decisions during compression, as discussed previously. A new instance of the compressor always starts in the *None* state, which signifies that there is no connection established with the decompressor as of yet and no uncompressed packet was seen.

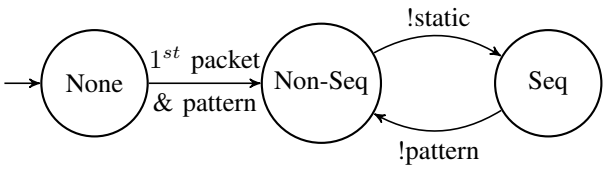

Fig. 1. Compressor state machine

The compressor will always transit from the *None* state to the *Non-Seq* (i.e., *Non-sequential*) state when it sees the first uncompressed packet *and* has access to a *byte-pattern* from the *oracle*. In the *Non-Seq* state the compressor is always forced to send *non-sequential* packets in order to ensure decompressor initialisation or the synchronisation of *static* fields.

From the *Non-Seq* state the instance can advance to the *Seq* (*sequential*) state when no change occurs in any of the *static* fields. The packets sent in this state can be any of the *sequential* pool. The state machine does not enforce the choice of *seq* packet types, as the compressor always chooses the one that would yield the smallest packet, while still being decompressible.

A transition back to the *Non-Seq* state is also possible if either a new pattern is received from the *oracle* or the field characteristics violate the last used pattern.

III. HEADER COMPRESSION

In this section we discuss our measured results for *O2SC* applied to the standard *IP* protocol stack using the following stream as an example:

- *IP*: In the case of *IPv4*, the header contains every mandatory field, resulting in 20 bytes of header information and the *identification* number always advances by exactly 1 for each consecutive packet. In case of version 6 of the *Internet Protocol*, only the base header is considered (i.e., no *extension header*s are present, like *routing*, *hop-by-hop options*, etc.)
- *UDP*: The *UDP* packet header adds an extra 8 bytes of overhead to the above *IPv4* (*IPv6*) header, however, it only contains *static* and *random* fields (i.e., *UDP checksum*).
- *Easy RTP*: With the addition of the *RTP* header, the structure of the packet becomes more complicated. For this evaluation, we only consider a constant size *RTP* header, meaning that optional and variable sized fields are omitted (the *CSRC identifiers* and the *extension header*). This way, the *RTP* adds an extra 12 bytes of overhead.
- *Hard RTP*: This stream is the same as the *easy RTP*, except that the *marker-bit* sets and unsets after a time randomly determined from the $[5, 25]$ interval. The same is true for the *packet type*, which randomly changes after $[100, 200]$ packets and the *timestamp*, which varies with a delta between 150 and 300 every $[150, 300]$ packets.

In all cases we omit the payload in order to measure the pure header compression.

In Figure 2 we show the achieved *compression gain*s when employing the experimental implementation of the *O2SC* compression. In all cases the compression performs with at least 80 % gain, in the case of *IP* and *UDP/IP* even around 90 %. For the *RTP*, *compression gain* is somewhat lower due to the *oracle* flagging more fields as *random*.

In Figure 3 the mean compressed header is presented with 95 % confidence intervals. In the case of the non-*RTP* streams, the average size falls closely around 1 byte, while for *RTP* it is mostly between 5 and 6 bytes.

IV. COMPRESSION OF CAPTURED STREAMS

In this section we evaluate the compression for various real-life streams that were captured on the network interface. Consequently, most of them contain 14 bytes of *Ethernet* header. For the creation of the *logistic regression* model we took the first 100 packets of each as the training datasets and

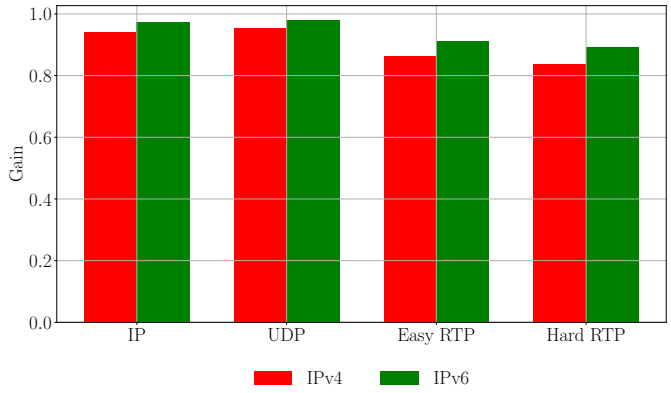

Fig. 2. Various protocol headers and their corresponding average *compression gains* with *O2SC*.

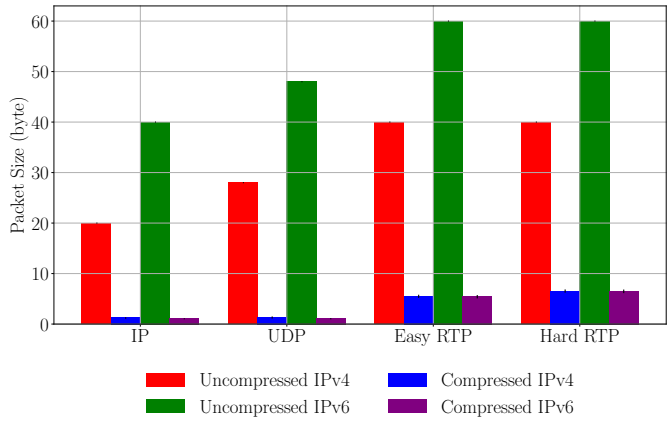

Fig. 3. Various protocol header lengths before and after compression with *O2SC*, as well as 95 % confidence intervals.

evaluated the compression with the following independent 900 packets.

- *Franka*: This stream was captured on a direct link between a *Franka Emika* robotic arm[1] and its controlling computer. During the recording the arm performed repeated circle drawing motions. The stream contains a single *IPv4* header and 235 bytes of undefined payload.
- *Asterisk*: We connected the *Asterisk* VoIP server[2] to a fixed desktop client and an *Android* smartphone, both using the *ZoIPer VoIP* client software[3]. This configuration utilised the *GSM 06.10* codec, which is the full-rate audio codec version that results in 33 bytes of payload. The *RTP timestamp* and *sequence number*s have a constant increase (delta) of 160 and 1, respectively. The *RTP marker bit* is always 0 except in the first two packets.
- *Ekiga*: This scenario represents a video conferencing session, which was originally generated using the *Ekiga* open source softphone software[4]. The packets contain *RTP* headers and a relatively large payload of 160 bytes.
- *Radio*: A *TCP acknowledgement* stream of a digital radio station, which is in general efficient to compress as no logical payload is present. This stream is the only one which doesn't contain an *Ethernet* header.
- *VLC*: This scenario represents a high fidelity audio transmission and was generated using the *RTP* streaming features of the popular *Video Lan Client* (*VLC*) open source software[5]. This is again an *RTP* session, however, in this case the underlying *IP* version is the 6. The logical payload is 320 bytes.

In Figure 4 we show the various *compression gains* for the above streams including any payload that is present. We observe that the highest *gain* is achieved for the *Radio* stream at about 83 % and the lowest is with the *VLC* stream, which is below 20 %. However, contrary to the normal *header compression* scenarios, *O2SC* does not have any knowledge about the separation of network headers and payload and it also considers the latter part for compression.

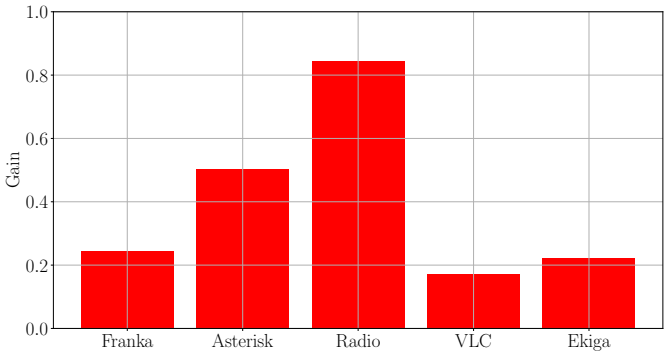

Fig. 4. *Compression gain* using *O2SC* for various real-life network streams.

In order to put these *gains* into more context, we present the uncompressed and compressed packet sizes in Figure 5. We observe that in every case there are savings of at least 50 bytes which, considering the limited amount of known headers (40-60 bytes), is a significant gain not unlike the established compression standards.

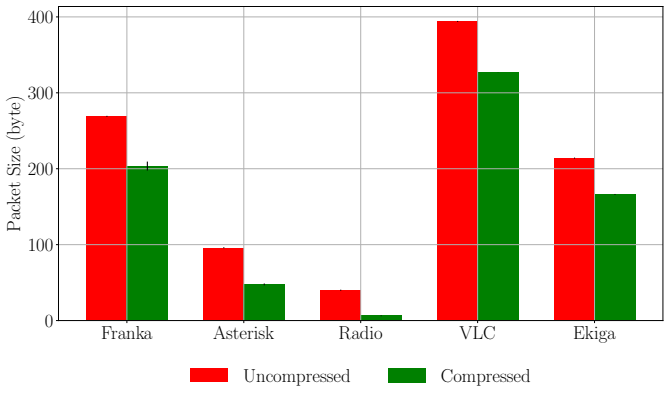

Fig. 5. Average packet sizes (and 95 % confidence intervals) without and with *O2SC* for various real-life network streams .

[1]See https://www.franka.de/panda/ for details.
[2]See https://www.asterisk.org/ for details.
[3]See http://www.zoiper.com/ for details.
[4]See http://www.ekiga.org/ for details.
[5]See http://www.videolan.org/ for details.

In Figure 6 we present the achieved compressed packet sizes. We note that most of the degradation of the compression gain hails from the repeated transmission of the oversized *non-sequential* packet.

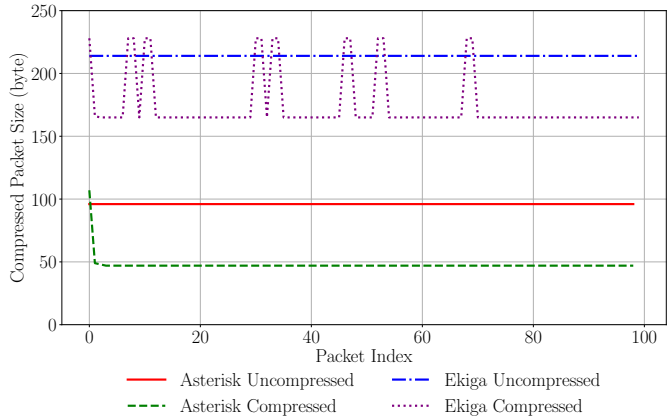

Fig. 6. Uncompressed and compressed packet sizes during voice call sessions with the *Asterisk* server and the *Ekiga* platform.

In some cases this occurs quite frequently, like with the *Franka* stream, shown in Figure 7. A solution for this would be to further decrease the transmitted *pattern length* or to employ references for repeating patterns, such as the case with *table-based compression* [9]. Even though the achievable gain is degraded by the repeated transmission of the *non-sequential* packets, the enormous gain from the compression of some of the packets by more than 150 bytes adds up. This also leads us to believe that our compression concept can step past the constraints of only tackling the compression of standardised protocol headers and can provide benefits even inside the *application layer* and the logical payload.

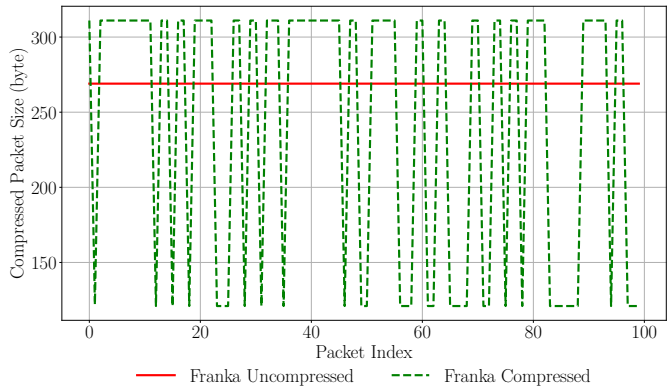

Fig. 7. Uncompressed and compressed packet sizes of the control stream for the *Franka Emika* robotic arm.

V. Conclusion

Compression of *IP* packet flows has seen a wide adoption in *fourth generation* wireless networks. However, solutions for flexibly compressing a wider range of (future) packet flows has not emerged as of yet.

In this paper we presented for the first time a compression design that can tackle the compression of various network flows without any prior knowledge during the design of the compression solution. We proposed the employment of a *machine learning*-based *oracle* entity that can determine the structure of compressible packets and which can be queried and interpreted by the appropriate compression platform. Consequently, we have introduced a completely new scheme based on the current state-of-the-art *Robust Header Compression* that can compress significant parts of the packets, yielding compression gains up to 90 %.

Nonetheless, our current design and evaluation is limited, as we do not yet consider losses in the compressed packet flow and multiple streams. Moreover, the handling of changing packet lengths and variable sized headers is still open. Implementing a solution to these issues is part of our ongoing work, as well as the thorough analysis of the underlying theoretical principles.

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.

References

[1] 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.

[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] S. Rein, M. Reisslein, and F. H. P. Fitzek, "Voice quality evaluation for wireless transmission with ROHC," tech. rep., in International Conference on Internet and Multimedia Systems and Applications, 2003.

[4] F. Fitzek, S. Rein, P. Seeling, and M. Reisslein, "Robust header compression (ROHC) performance for multimedia transmission over 3g/4g wireless networks," *Wireless Personal Communications*, vol. 32, no. 1, pp. 23–41, 2005.

[5] P. Seeling, M. Reisslein, F. 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*, vol. 3, pp. 1346–1350 vol.3, Dec 2003.

[6] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet-Draft draft-ietf-quic-transport-19, Internet Engineering Task Force, Mar. 2019. Work in Progress.

[7] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links." RFC 1144, Feb. 1990.

[8] A. Martensson, T. Wiebke, C. Burmeister, R. Hakenberg, H. Fukushima, T. Yoshimura, M. Degermark, K. Le, H. Zheng, H. Hannu, Z. Liu, K. Svanbro, L.-E. Jonsson, A. Miyazaki, T. Koren, and C. Bormann, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed." RFC 3095, July 2001.

[9] K. Sandlund and G. Pelletier, "RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite." RFC 5225, Apr. 2008.

[10] 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.

[11] M. Tömösközi, D. E. Lucani, F. H. Fitzek, and P. Ekler, "Unidirectional robust header compression for reliable low latency mesh networks," in *2019 IEEE International Conference on Communications (ICC): Mobile and Wireless Networks Symposium (IEEE ICC'19 - MWN Symposium)*, (Shanghai, P.R. China), May 2019.