Automated Header Compression in Constrained Networks

Soumya Banerjee, Dominique Barthel, Quentin Lampin, Marion Dumay, Stéphane Coutant, Cédric Adjih, Paul Muhlethaler, Thomas Watteyne

Automated Header Compression in Constrained Networks

Soumya Banerjee, Dominique Barthel, Quentin Lampin, Marion Dumay,
Stéphane Coutant, Cédric Adjih, Paul Muhlethaler, Thomas Watteyne

***Abstract*—In low-power wireless networks, every byte sent by an embedded device causes its radio to stay on a little longer, which eats into its limited energy reserve. And because the radio is often the most power-hungry circuit in the device, reducing the number of bytes to be sent and received automatically increases the battery lifetime of the device, resulting in a lower total cost of ownership for the end-user, hence better adoption. Low-power wireless devices tend to generate short data payload, typically in the order of 2-50 B. This means that protocol headers make up a large portion of the bytes inside a wireless frame, 30-70% is not uncommon. Compressing those headers, i.e. removing bytes that can be reconstructed anyways or that are not needed, makes perfect sense. This article serves as a primer on header compression in constrained networks. We start by describing exactly why it is needed, then survey the different standards doing header compression. We indicate how today's approach requires expert input for every deployment, severely hindering the roll-out of such approaches. Instead, we argue that an automated approach based on machine learning and artificial intelligence is the right way to go, and provide blueprints for such approaches.**

***Index Terms*— Automated Rule Generation, Compaction, Constrained networks, Header Compression, Internet of Things (IoT), Low-Power Wide Area Network (LPWAN) Device, Machine Learning, Protocols.**

I. Why Header Compression Is Needed

Most communication protocols, such as Internet Protocol version 6 (IPv6), were designed to run on technologies (Ethernet, WiFi, etc.) which do not have any real constraints in terms of frame length. For example, the Maximum Transmission Unit (MTU) of Ethernet is 1,500 B: a single Ethernet frame can contain up to 1,500 B of link-layer payload. This means that the 40 B of the IPv6 header only account for 2.7% of the Ethernet payload. In the networks we use day-to-day, the concept of "header compression" doesn't make much sense.

This changes when we talk about "constrained" networks. These are technologies, pretty much always wireless, where frames tend to be very short. IEEE802.15.4 is one such example, in which frames are 128 B long at most, leaving approximately 110 B of link-layer payload. Low-Power Wide Area Network (LPWAN) technologies such as SigFox feature a link-layer payload size that can be as low of 12 B, and "new space" satellite constellations such as a Kineis, which allow IoT devices on the ground to send their payload directly to the satellite, have a payload of 30 B.

S. Banerjee is with Trasna-Solutions Technologies Ltd., Ireland.
D. Barthel, Q. Lampin, M. Dumay, S. Coutant are with Orange Labs, Meylan, France.
C. Adjih, P. Muhlethaler, T. Watteyne are with Inria, France.

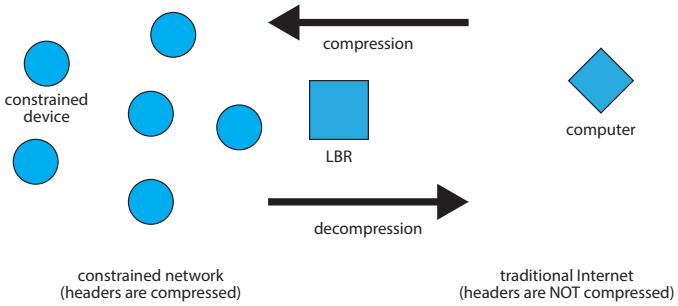

Fig. 1. The Low-power and Lossy Network Border Router (LBR) is the interface between the compressed-header IPv6 mesh and the legacy Internet.

The payload length is limited for two reasons. First, longer payloads take longer to be transmitted, and statistically, the longer in the air, the more prone to issues that would cause the receiver not to be able to receive. Given a constant Bit Error Rate (BER), the shorter the frame, the higher the probability it will be received correctly. Second, IoT devices are battery operated the vast majority of the time: every byte transmitted typically results into 10's to 1,000's of micro-Coulombs of charge leaving the battery. In finding the different trade-off points of these technologies, the designers rightfully chose short payload lengths.

If we wanted to transport standard protocols such as IPv6 and/or User Datagram Protocol (UDP) on top of these technologies, the headers of those protocols alone (40 B and 8 B in the case of IPv6 and UDP, respectively) would either take up an unreasonable amount of space, or simply not fit at all. The idea of header compression is a little trick to make it work nonetheless. The application running on these devices is typically very repetitive and simple, for example reporting a 2 B sensor reading to server with IPv6 address `2001:DB8::1` on UDP port 5683, every hour. The idea is hence to *not* send these long IPv6 address and UDP port fields, as they are always the same.

Of course, to allow a device to send the data to a regular server, we need some sort of computer to reconstruct the full packet. That computer, often called a Low-power and Lossy Network Border Router (LBR), sits between the compressed and uncompressed (regular) domains. The constrained devices start by sending packets with compressed headers; when these packets reach the border router, they get inflated into regular packets which can travel over the traditional Internet and be understood by the destination server. Similarly, when the servers send packets to the devices, the LBR compresses their

headers as the packets enter the constrained network.

To allow for some flexibility in the compression, the compressed header must be replaced in the packet by a (much shorter) set of flags that tell the LBR how to inflate. This can be stateless, i.e. the packet itself contains all the information for inflation, or stateful. In a stateful approach, a "context" is shared between the device and the LBR. In the example above, context item 3 might mean "packet for `2001:DB8::1` to UDP port 5683": the device hence only indicates "item 3" in the packet as it knows that the LBR will be able to inflate it. Of course, that context somehow needs to be shared between device and LBR beforehand.

This article is tailored for the advanced researcher or networking expert, and is tailored as both a primer on automated header compression and a reference for identifying research directions in the fields. It's contributions are threefold:

- it provides a comprehensive survey on standards-based header compression
- it discusses in detail the challenge of *automated* header compression, including early work and the intuition behing using modern pattern recognition techniques
- it defines a research roadmap for the field with two blueprints

The remainder of this article is organized as follows. Section II surveys standard-based header compression approaches Section III lists the main challenges of automated header compression. Section IV focuses on early work that used dictionary-based compression. Section V discusses the general intuive approch to automated header compression. Section VI introduces modern tools for patter recognition, then details research avenues for automated header compression, presented as blueprints. Finally, Section VII concludes this article.

II. Standard-Based Header Compression Approaches

Several standard-based header compression approaches have been developed for constrained low-power wireless systems.

IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [1] is an early example, which mainly targets IPv6 and UDP. The 6LoWPAN header replaces the IPv6 and UDP headers and indicates how each field is to be reconstructed. The node and the LBR agree on a prefix/address dictionary which can contain up to 16 entries. When doing so, the 6LoWPAN implementation on the node replaces a 16 B IPv6 address by the 4 b index. 6LoWPAN features an elegant way to compress the IPv6 addresses, as often-used address prefixes such as `fe80::` are replaced by single-bit flags. Some fields, such as the IPv6 length field, can be recomputed by looking at fields in other headers; Finally, some field, such as the Time-to-Live (TTL), cannot be completely removed, and appear as-is in the packet.

6LoWPAN-GHC [2] (GHC stands for "Generalized Header Compression") is an extension to 6LoWPAN. It is generic in that it works on arbitrary payloads and next headers, without ever needing to be redefined. 6LoWPAN-GHC is a dictionary-based compression scheme: the compressor goes through the payload to be compressed byte-by-byte. If there are segments of that payload that appear in a previously defined dictionary (a sequence of 48 B containing patterns which often appear in packets), the compressor replaces those segments by their start/end indices in the dictionary. All what it cannot compress is carried inside the packet. 6LoWPAN-GHC defines only a single static dictionary, in the hope that the segments it contains appear often.

TABLE I
Example SCHC rule, which compresses IPv6 (reproduced and adapted from RFC8724 [3], Fig. 27).

FID	FL	TV	MO	CDA
IPv6 Version	4	6	ignore	not-sent
IPv6 Diffserv	8	0	equal	not-sent
IPv6 Flow Label	20	0	equal	not-sent
IPv6 Length	16		ignore	compute-*
IPv6 Next Header	8	17	equal	not-sent
IPv6 Hop Limit	8	255	ignore	not-sent
IPv6 DevPrefix	64	[alpha/64, fe80::/64]	match-mapping	mapping-sent
IPv6 DevIID	64		ignore	DevIID
IPv6 AppPrefix	64	[beta/64, alpha/64, fe80::64]	match-mapping	mapping-sent
IPv6 AppIID	64	::1000	equal	not-sent

The development of the Generic Framework for Static Context Header Compression and Fragmentation (SCHC) [3] was triggered by the development of LPWAN technologies. The classes of LPWAN networks considered have payloads too small to carry 6LoWPAN-style flags, and are often upstream-only (there is no way to negotiate the context between LBR and devices). SCHC relies entirely on static contexts, i.e. a set of rules. Table I is an example rule which compresses the IPv6 and UDP headers of a set of packets. Each rule matches the considered packet headers entirely and contains one row per field in that packet. The compressor starts by looking whether the rule applies to the packet under consideration by ensuring the values of the different fields match the "Target Value" (TV) of the corresponding row in the rule, using the specified "Matching Operator" (MO). If yes, the compressor compresses the packet by applying the action specified in the Compression/Decompression Action (CDA) column. Compression might mean not sending the field and restoring it at the receive end from the TV (CDA `not-sent`), not sending the field and recomputing it by a well-known mechanism such as based one or several fields in the same or in different headers (CDA `compute-*`), or sending the index of the value from an array (CDA `mapping-sent`). The use of a set of rules allows SCHC to compress each flow of packets in a specific manner, as opposed to 6LoWPAN compressing all UPD/IPv6 flows the same way. This yields a higher compression ratio, at the expense of having to install and store the set of rules at both ends of the constrained link. On the constrained device, these rules necessarily occupy space in the limited memory.

In summary, a number of standard-based compression approaches have been proposed. Specifications such as 6LoWPAN are closely tied to the protocol they compress. The more generalized 6LoWPAN-GHC allows for compressing "any" protocol, at the expense of compression efficiency. Finally,

the recent SCHC stands out by clearly differentiating policy (the list of rules that build up a compression context) from mechanism (how to compress a packet given a context). For an in-depth survey about header compression, the interested reader is referred to [4].

III. Challenge

The ideal header compression approach would be efficient (i.e. the compressed packet is much smaller than the uncompressed one) and generic (i.e. it can work on any protocol). By separating policy from mechanism, SCHC achieves both efficiency and generality. That being said, the big downside of SCHC is that the set of rules that defines the context is so far crafted by protocol experts. The rules are typically specific to a particular deployment, even to a specific flow within a particular deployment. Before running SCHC on a deployment, the end user must either themselves be a SCHC protocol expert, or work with some type of SCHC consultant. They will go through the different types of traffic involved in that deployment, for example listing the IPv6 addresses of the different servers, then come up with a set of rules. These will then be installed in the devices and the LBR, and SCHC can be enabled.

This approach has at least two significant drawbacks. First, performance. When an expert crafts a set of rules, we haven't seen a rigorous performance evaluation of that set of rules. We do not at all doubt their expertise, but haven't seen any performance evaluation which shows that the particular set of rules identified is the better compared to other approaches. Second, scalability. The obvious downside is that involving experts for each SCHC deployment doesn't scale. That is, it is impossible to expect experts to participate in each SCHC deployment. This is a major shortcoming which severely impacts the adoption of SCHC as a compression protocol.

The idea is hence to automate the rule generation. This can be done as follows. In a SCHC deployment, the operator gathers characteristics of the data flows in their network. Typically, this is a packet trace (e.g., `tcpdump`) of the data already flowing through the network before SCHC is activated, and which is representative of the data that will be flowing in the future. The operator then hands that trace to a computer program which analyzes that data and generates a set of rules. Together with the rules, the computer program gives some indication about the performance of these rules, possibly both in terms of compression rate of the packets and memory footprint on the devices. As described above, this automated approach is both perfectly scalable, and is driven by the performance of the resulting rules.

IV. Early Attempts

The idea of being able to compressing packets in an automated way has been explored in different forms. Massey *et al.* proposed a protocol-agnostic compression approach as early as 2010 in [5], [6]. The idea, at the time, was to consider the compression routine as "outside" of the protocol itself. That is, the protocol stack running on a device generates a frame to be transmitted, which gets compressed right before it is sent to the radio. This approach is very similar to how file compression algorithms (zip, etc.) work, as the compressor doesn't need to know anything about the contents of the file. This point is very different from the 6LoWPAN or SCHC header compression approaches listed in Section II, which know how to parse a packet header.

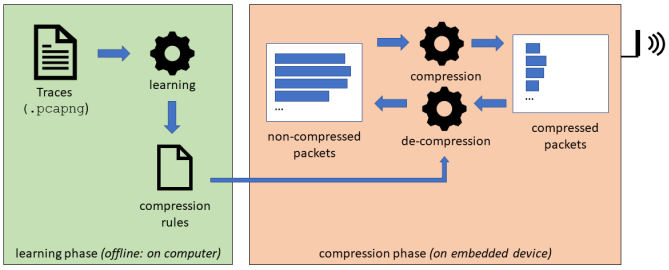

Fig. 2. Overview of the automated header compression approach.

Massey's compression technique can be seen as a dictionary-based approach. The compressor operates on a stream of packets. It receives the packets one after another, and does a byte-wise comparison between a packet and each of previous ones in a recent packet buffer. If there is a sequence of consecutive bytes that is the same, it calls that a "pattern". It then puts the pattern in a pattern list (which serves as the dictionary) and replaces that pattern by its index in the list. Compression stems from the index being much shorter than the pattern. The algorithm has three parameters: the minimal length of a pattern, the length of the pattern list and the length of the pattern buffer. The algorithm can be tuned by changing those three parameters to result in the highest possible compression, while maintaining an acceptable Random-Access Memory (RAM) memory footprint.

One of the strengths of the approach is that the pattern list (the dictionnay) is never explicitly transmitted from the compressing device to the decompressing device. Rather, because they run the same algorithm on the same packets, they make the same decisions at the same time, and therefore end up building the same dictionary. This is also the important pitfalls of this approach: it assumes the protocol is perfectly reliable. And while it could operate above the link-layer, i.e. where link-layer acknowledgments can offer some level of reliability, it certainly should include a recovery mechanism in case the dictionaries get out of sync.

This last point highlights the two subtle differences between file/stream based compression, such as the well-known Lempel-Ziv algorithm LZ77, and protocol compression. First, protocol compression has to deal with packet loss, the necessity to detect it and recover from it; file/stream decompression is impossible if parts of the compressed file is missing. Second, it is much more costly for a protocol compression algorithm to share the dictionary with the decompression algorithm; file/stream compressors simply write the dictionary at the end of the compressed file.

V. Intuitive approach

Fig. 2 illustrates the approach of automated header compression. It operates in two steps: a learning phase done on a

computer, a compression phase done on the embedded devices. The goal of the learning phase is to generate the SCHC rules to be used later on the embedded device. It consist in running the algorithm for automated rule generation, detailed in Section VI, on series of traces. These traces, typically known as "`PCAP` files" generated by the popular Wireshark sniffing uility, must be representative of the traffic to be compressed. These traces are fed to the learning algorithm that is in charge of analyzing the traces and produce compression rules. The learning is done automatically, and consists in identifying patterns which repeat in different packets. Depending on the type of compression, this analysis can be done on a field-by-field basis (which assumes the compressor is capable of parsing the headers), or on a byte-by-byte base as used in the early attempts surveyed in Section IV. The resulting rules are installed on the device, which can now enter the compression phase. During the compression phase, when the embedded device is about to transmit a packet, it first applies the compression routine (which uses the rules) to generate smaller compressed packets, which are then sent over the antenna to nearby devices. Similarly, when the device receives a packet, it knows it is compressed, and applies the decompression routine (again using the rules) to obtain a non-compressed version of the packets, and analyze those further.

The main Key Performance Indicator (KPI) of this approach is the compression ratio. That is, what set of compression rules will result in the shortest compressed packets. Of course, this KPI has to be computed given the bounds of the system. The main bound is on the memory footprint on the embedded device. The compression rules typically indicate the pattern (i.e. the field and/or position in the packet, its target content, and possibly what operation to use to match it) and the way to compress it. This takes up space in memory (either RAM or flash depending on the implementation); the footprint of those rules cannot exceed the memory capabilities of the device. Given the computational power of embedded devices, where 16- and 32-bit Central Processing Units (CPUs) running at 10's or 100's MHz is commonplace, consuming only around 1 nJ/instruction, we don't see the computational overhead on the embedded device as a bottleneck in the general case. Similarly, even if the learning algorithm may involve some advanced machine learning or artificial intelligence routines, the learning is done once offline on a powerful computer. So here again, we don't see the computational power during the learning as a bottleneck.

VI. Blueprints of Possible Approaches

This section proposes gives an overview of modern tools for "pattern recognition" (Section VI-A), then develop two blueprints to automate the rule generation process for SCHC (Section VI-B and VI-C). For the latter, we use the approach from Section V: identify groups of similar packets out of the whole set of IoT traces, then generate a single compression rule for all the packets belonging to a same group. From a machine-learning perspective, we want to identify groups of packets that can be compressed by a specific rule. This is analogous to the classical problem of *clustering* in unsupervised learning: automated rule generation becomes a clustering problem. That is (1) perform clustering of the packets and (2) generate one SCHC rule for each cluster.

A. Modern Tools for "Pattern Recognition"

Classical compression approaches based on dictionaries or entropy coding have been adapted to IoT. Matsuda *et al.* [7] for example create a specific dictionary for different kinds of data and focus on the compression of application-related data (not the headers). Machine learning concepts can also be used.

Tomoskozi *et al.* [8], [9] use linear regression to create a model that derives and predicts the compression utility scores for RoHC [10]. They use an offline characterization of the given compressor implementation, header dynamics, packet loss probabilities, etc.

Nivasch *et al.* [11] use reinforcement learning in the context of classical compression method (Lempel-Ziv-Welch, LZW [12]). They dynamically build a dictionary for stream compression. The novelty is in the way the dictionary is built: when the LZW algorithm decides to add a string to its dictionary, an extra confirmation decision has now to be given by a machine learning algorithm.

Classical lossless compression techniques exploit redundancies in the data: they replace a repeated set of bytes with a more compact reference to a previous occurrence or to a dictionary entry. They build those data structures dynamically. SCHC has been deliberately designed around the use of *static* contexts. This means the redundancies need to be exploited in an offline manner, also in a different way using their properties. That is, online machine learning techniques are not always applicable.

B. Grouping Packets through Clustering

We want to group packets into clusters by similarity. This can be done through *distance-based* clustering (e.g. the popular k-means clustering): clusters are groups of packets for which the internal distance between each pair of packets is minimized. For SCHC, the "distance" between two packets can be defined in several ways. Two packets can be compared field-by-field: the more dissimilar, the higher the distance. Of course, many fields are not values that can be numerically compared (e.g. IPv6 addresses); on those, metrics such as Gower distance make more sense.

There are subtleties, however. First, the control choice of the number of clusters is not obvious: when it is an input parameter of a clustering algorithm, there is a question of the optimal number clusters to generate efficient rules. Second, given a cluster of *similar* packets, it is not assured that an efficient rule can be generated to compress all of them. A favorable scenario is, for instance, when all fields are identical for all packets, except for one fixed field. An associated SCHC rule can been generated where all fields, except for that one field, has a SCHC compression action that removes them. An unfavorable situation is a cluster where each pair of packets have mostly similar fields, but the fields that are similar vary substantially depending on which pair of packets is selected. In that case, we can create efficient rules that match and compress any pair of packets of the cluster, but probably not all of them

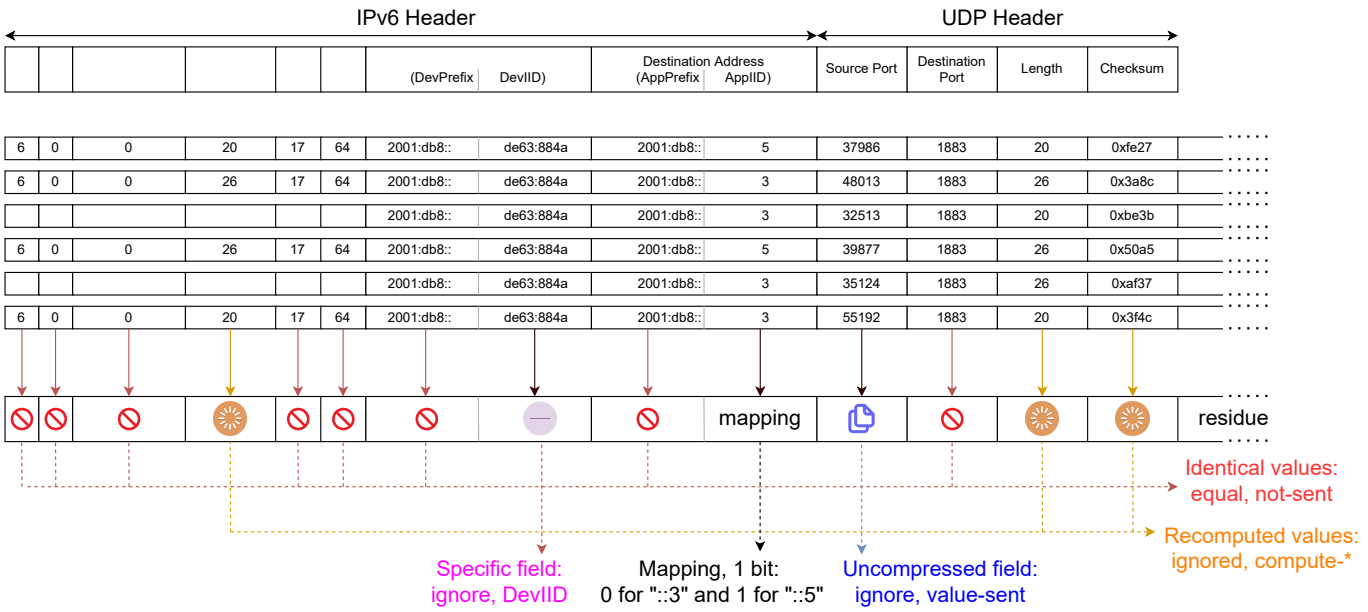

Fig. 3. Example rule generation from a cluster with 6 packets.

at the same time. So rule generation and clustering are not exactly the same: we may need to modify existing clustering algorithms.

There are practical domain-specific issues that make the problem different from clustering:

1) Not all packets have identical structures: some can have different fields that are present, e.g. Constrained Application Protocol (CoAP) packets versus plain UDP packets without CoAP.
2) SCHC rules are always applied within a context of a device transmitting to a core server: thus a SCHC rule does not need to be applied to all packets in general.
3) Some fields have special semantics; some specific matching operators and compression/decompression actions are included in SCHC for special fields such as length, checksum or IPv6 suffixes.

For managing (1), a practical step is to pre-process the packet traces so that packets in each group have the exact same list of fields, before doing the clustering. For managing (2), filter packets belonging to the same context (corresponding to the same device), then try to find rules. For managing (3), identify the fields with specific compression/decompression action, and build the rule accordingly.

C. Rule Generation from a Set of Packets

Once the clusters are built, the final step is to generate the rules for each of them. Fig. 3 shows an example where a cluster consists of six packets from one device. The part at the bottom represents the structure of the unique rule that is being generated:

- The fields marked with a red crossed circle are fields for which all packets have the same value. Use matching operator that verifies the field is exactly equal to the common value. Ignore the field when transmitting the packet.
- The field marked with a magenta disk is ignored and ellided because it is known, as it is the DevIID part of the IPv6 address of the device, based on its layer-2 address.
- The fields marked with an orange disk can be recomputed on the receive side. They are ignored and are not sent.
- The field corresponding to the IPv6 destination address (AppIID) has one of two values in the example (`::3` and `::5`. Use compression action `mapping-sent`, with 1 bit specifying which of the values is present in the packet.
- The field in blue, the UDP source port, is not compressed. It is sent as is, occupying 2 bytes.

The generated rule compresses the IPv6 and UDP headers from 48 bytes to 17 bits.

An important challenge of rule generation can also be understood from the example when considering the field UDP source port. Another possible encoding of the field is to consider the fact that six values have been observed. The field could be compressed as a mapping: sending an index to the list of the six possible values, and encoding it in 3 bits. This yields more efficient header compression (4 bits for the whole packet instead of 17 bits), but at the risk of encountering a different value for this field in future packets.

Philosophically, it boils down to the question of whether the packets that will be exchanged in the network in the future would be *exact copies* of the previously observed ones, or just be *similar*. Because SCHC's `mapping-sent` compression action doesn't have an "escape" value reserved for signaling a field value outside the predefined list of values, a heuristic choice has to be made. We need to choose between not compressing a field or using a mapping to a list of pre-defined values. In the example, six different values of the UDP source port have been observed for just six different packets, this

might be interpreted as a good indicator that the fields might take more values in the future.

In general, the rule for a cluster will follow the same principles for each field: do not send any field that has the same value in all packets of the cluster; do not send fields that can be re-generated at receiver side. For the other fields, either use a mapping to a list of predefined values, send the field as is, or use another optimized action, which is not presented in the example: that is, send the least significant bits of the field if some of the most significant bits are identical.

D. Optimizing Rule Generation

In the proposed approach, the complex problem of rule generation was separated into two different problems: clustering and rule generation. Of course, to some extent, this split is artificial and was introduced for tractability. As indicated, the problem faced is not exclusively a clustering problem: rule generation has its subtleties, and both are linked. Because of the complexity of the problem, it is not unreasonable to envision that the proposed blueprint could be enriched or even replaced by future methods based on deep learning.

E. Standardizing the Resulting Solutions

Protocols such as SCHC are already standardized [3], yet the big difference with earlier protocols such as 6LoWPAN [1] or 6LoWPAN-GHC [2] is that SCHC clear differiates rule generation from rule execution. This is typical in networking standards which differentiate "mechanisms" (typically the protocol, including packet formats, for devices to communicate) from "policies" (algorithms or rules that determine different aspects of the protocol). While the mechanisms must be defined and agreed upon for interoperability, policies are typically left to the implementor (i.e. not defined or standardized) to allow different implementors to differentiate. The same holds for SCHC, which defines how a rule is executed (the "mechanisms" of SCHC), but not how the rules themselves are defined (its "policies").

That being said, standardization working groups typically define a base or minimal algorithm which can serve as a kick-starter for more advanced one. One example is the Minimal Scheduling Function (MSF) [13] of the 6TiSCH protocol. The same can be done in the SCHC working group at the IETF [14]: submit the work resulting from these blueprints as standards-track Internet-Drafts in the working group.

VII. Conclusion

The protocols we use on the Internet today are used to carry such a large amount of data (think of streaming a movie) that the handful of bytes in these protocol headers has little impact on the overall volume of data to be shipped around. And because of this volume, the underlying networking protocols and equipment can forward large physical-layer frames.

Things are very different in constrained low-power wireless networks, in which small battery-powered embedded devices are deployed in hard-to-reach places to periodically report a couple of bytes of data. Since we do want those devices to be part of the Internet in a native manner, the frames they generate need to be formatted "the Internet way", i.e. include the same protocol headers as generated by traditional computers. This leads to an odd situation where protocol headers can take up as much or more space in the frame as the data itself.

Header compression is a great answer in which the gateway of the constrained network, called an LBR, works with the constrained devices: the devices remove the bytes in the headers that can be reconstructed, the LBR reconstructs the full packet before sending it into the Internet. Several standard protocols such as 6LoWPAN [1], 6LoWPAN-GHC [2] and SCHC [3] do just that. The challenge is that they are often crafted for a specific protocol only, or are too generic to be highly efficient, or need tailoring the compression rules to each deployment case. The last point requires real protocol expertise, which severely hinders the adoption of these approaches.

This article argues for an automated approach, in which machine learning and artificial intelligence is used to analyze a set of traces offline. This routine results in a set of rules, which can the be used by the production system without expert intervention.

References

[1] J. Hui and P. Thubert, *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*, Internet Engineering Task Force (IETF) Std. RFC6282, 2011.

[2] C. Bormann, *6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, Internet Engineering Task Force (IETF) Std. RFC7400, 2014.

[3] A. Minaburo, L. Toutain, C. Gomez, D. Barthel, and J. Zuniga, *SCHC: Generic Framework for Static Context Header Compression and Fragmentation*, Internet Engineering Task Force (IETF) Std. RFC8724, 2020.

[4] M. Tömösközi, M. Reisslein, and F. H. P. Fitzek, "Packet Header Compression: A Principle-Based Survey of Standards and Recent Research Studies," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 698–740, 2022.

[5] T. Massey, A. Mehta, T. Watteyne, and K. Pister, "Protocol-Agnostic Compression for Resource-Constrained Wireless Networks," in *Global Telecommunications Conference (GLOBECOM)*. IEEE, 2010.

[6] ——, "Packet Compression for Time-Synchronized Wireless Networks," in *Conference on Sensor, Mesh and Ad Hoc Communcations and Networks (SECON)*. IEEE, 2010.

[7] K. Matsuda and M. Kubota, "Compound Compression Method for Gathering Traffic of IoT/CPS Data," in *World Forum on Internet of Things (WF-IoT)*. IEEE, 2019.

[8] M. Tömösközi, P. Seeling, P. Ekler, and F. H. P. Fitzek, "Performance Prediction of Robust Header Compression version 2 for RTP Audio Streaming Using Linear Regression," in *European Wireless Conference*. IEEE, 2016.

[9] ——, "Regression Model Building and Efficiency Prediction of RoHCv2 Compressor Implementations for VoIP," in *Global Communications Conference (GLOBECOM)*. IEEE, 2016.

[10] G. Pelletier and K. Sandlund, *RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite*, Internet Engineering Task Force (IETF) Std. RFC5225, 2008.

[11] K. Nivasch, D. Shapira, and A. Azaria, "Deep Reinforcement Learning for a Dictionary Based Compression Schema," in *Conference on Artificial Intelligence*. AAAI, 2021.

[12] W. Kinsner and R. H. Greenfield, "The Lempel-Ziv-Welch (LZW) Data Compression Algorithm for Packet Radio," in *Communications, Computers and Power in the Modern Environment Conference (WESCANEX))*. IEEE, 1991.

[13] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. R. Dujovne, *6TiSCH Minimal Scheduling Function (MSF)*, Internet Engineering Task Force (IETF) Std. RFC9033, 2021.

[14] A. Pelov and P. Thubert, *Static Context Header Compression (SCHC)*, Internet Engineering Task Force (IETF) Std. Charter 1, 2023.