DYNAMIC CONTEXT UPDATING IN SCHC

by

Anas Alhadi

Submitted in partial fulfillment of the requirements
for the degree of Bachelors of Computer Science, Honours

at

Dalhousie University
Halifax, Nova Scotia
April 2025

Table of Contents

Abstract

Static Context Header Compression (SCHC) is an adaptation layer capable of achieving large compression ratios on upper layer protocol headers. It does this by exploiting the persistent and predictable nature of IoT networks to make use of predefined static compression rules that act as blueprints for the expected network traffic. By synchronizing the rules stored at both ends of a link, the sender can avoid transmitting the entire packet and instead only sends a pointer to the rule describing the packet's content. However, the assumption of an unchanging network state that SCHC builds on makes its use in mobile networks infeasible given that the metadata that SCHC compresses is now variable. We look into the prospects of introducing dynamic updating of rules by evaluating the performance of weight assignment heuristics on network traffic in an emulation environment to predict rules that better describe the network flow, thus improving the chances of compression taking place. Our results show that in the presence of what we hope to be realistic assumptions, an overall decrease of approximately 60% on the average size of the metadata being transmitted is possible.

Acknowledgements

I would express my deepest gratitude to Dr. Samer Lahoud for his guidance and support throughout this project.

Chapter 1

Introduction

1.1 Context

The recent surge in IoT based networks facilitated the need for a shared networking layer to enable seamless communication between various types of devices regardless of their underlying communication medium and protocols. As such IPv6 has been widely adopted to play that role [11]. Yet, this in itself introduced a new set of challenges, specifically for low power wide area networks (LPWAN) given the physical restrictions imposed on the size of their maximum transmission unit (MTU). In which, the relatively large size of an IPv6 header exceeds the MTU necessitating the use of fragmentation. Moreover, the use of large packet headers results in longer time-on-air during message transmission thus increasing power consumption [1].

A solution to the aforementioned was introduced by the LPWAN working group in the form of the SCHC protocol with mechanisms for compression and fragmentation [6]. SCHC has received wide adoption, with the LoRa Alliance choosing it as the IPv6 adaptation layer for LoRa based communications, as well as the emergence of of studies examining its use in a wide range of IoT networks, such as that of the Internet of Vehicles via IPv6 over LoRa [11], and in Direct-to-satellite IoT networks (DtS-IoT) [9]

1.2 Motivation

A number of studies were produced with the aim of evaluating the use of SCHC in various mobile IoT applications [9, 11]. We notice however that both studies considered the process of configuring the compression parameters as a manual procedure done beforehand. While this is a reasonable assumption to make in certain IoT networks, it raises the question of the feasibility of using SCHC in applications that are inherently subject to some level of variability in the contents of the network traffic's

metadata. Given that presence of such variability violates a core assumption of the persistence of the network state that SCHC builds on. Moreover we argue that having a mechanism to dynamically configure SCHC can largely improve the protocol's adaptability and resilience to changing network conditions.

1.3 Objective

This thesis introduces the compression mechanisms used by the SCHC protocol and argues for the limitations of a purely static configuration process in the context of mobile network. We then introduce a novel mechanism for dynamic rule updating which makes use of existing scheduling heuristics to predict compression parameters. Finally, we evaluate the performance of the rule updating procedure given controlled levels of uncertainty on the contents of the IPv6 headers.

Chapter 2

SCHC Background

2.1 Network Architecture

In this section we cover the general network architecture that SCHC was built to operate on as well as the naming convention used.

The IETF RFC [6] states that SCHC was built with LPWAN networks in mind , thus there is an implicit assumption that all communications occur between a low-power low-compute IoT device (DEV) that is connected to a more powerful network gateway (NGW) in a star topology. The NGW is connected to an external server (APP) through the internet. Moreover SCHC explicitly distinguishes traffic based on the direction it flows at, with Uplink traffic moving from DEV to APP while downlink traffic moves from APP to DEV.

Figure 2.1 illustrates the aforementioned network structure, where Radio Gateways and the LPWAN-AAA server are the layer 2 switches and the LPWAN security server respectively, both of which are not directly involved in SCHC.

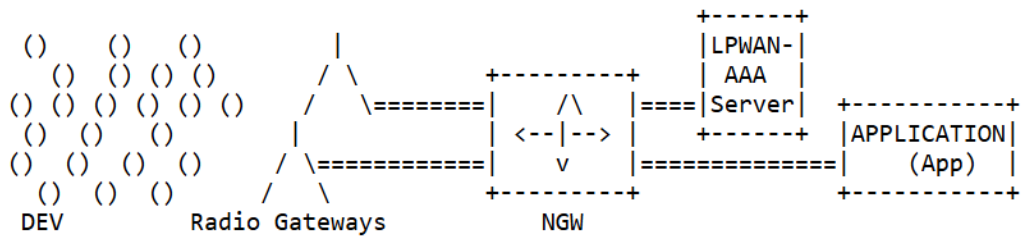


Figure 2.1: LPWAN Architecture. Reproduced from RFC8376 [2]

Given that the radio gateway marks the end of the constrained link all SCHC operations occur between the DEV and the NGW [4, 6]. Moreover, because of the network's star topology the NGW is expected to store the compression rules for all the DEVs connected to it, while each DEV only needs to store rules that are directly associated with it's link to the NGW. A consequence of this is that any operation

made to update the rules should occur at the NGW which then communicates the changes to the DEVs.

2.1.1 SCHC Workflow

To illustrate the general workflow of SCHC, consider the case of a CoAP/UDP/IPv6 communication over a LoRaWAN network between a sensor device (DEV) and a remote server (APP). Initially, the DEV would construct the CoAP/UDP/IPv6 packet with the appropriate application data and sends it down its network stack to the SCHC layer. SCHC would then parse the header fields of each of the above layers and compares them to available rules. Once a matching rule is found the compression process constructs a new SCHC packet which replaces the upper layer headers with a SCHC header containing the selected rule's identifier and some additional metadata. The SCHC packet (SCHC header and application data) is then sent to the LPWAN layer ultimately landing it at the NGW (given the network's star topology). Once the NGW receives the packet, it uses the rule identifier and the metadata in the header to reconstruct the CoAP/UDP/IPv6 packet which is then sent over the Internet to the APP.

2.2 Compression

The SCHC protocol uses a lossless, context based compression mechanism, that is, it makes use of previous knowledge about a given communication to omit repeated parts of a packet's metadata. Though it differs from other context based compression protocols in the derivation of said contexts, in which contexts are acquired in a purely offline and manual manner, relying on the network admins assumed pre-existing knowledge on the state of the communication between two endpoints to predefine the rules needed. This assumption is not only safe given the stable and simple nature of IoT applications with devices often running a single application, but also plays an important role in bypassing the need for synchronization messages. To put it into perspective, the minimum size of the metadata in a COAP/UDP/IPv6 network stack is approximately 60 bytes per packet, assuming the network stack is operating over a low data rate LoRaWAN connection means that the MTU would be 59 bytes with only 51 bytes available for upper layer data [5], thus requiring fragmentation.

The additional overhead incurred by the need to fragment uncompressed packets in itself makes flow based context synchronization, similar to that used in the ROHC protocol infeasible. [7].

Though even with predefined configurations, a compressor needs to first decide on which context to use based on the current network traffic before initiating the compression process. To best understand how a context rule is chosen, it is beneficial to cover the hierarchical structure of SCHC contexts.

Context

A context in SCHC (also referred to as profile) is a pair $C = (CID,\ R)$ in which the CID is a unique identifier for the devices connected to the NGW, used to decide on which context to use based on the DEV at the other end of the communication. The SCHC standard puts no restrictions on what can be used as a CID and provides the option to use the DEV's data-link address (i.e. MAC address). R on the other hand is a set of compression rules that can be used for the communications with the DEV that the CID corresponds to.

Rules

SCHC rules can be thought of as blueprints for packet headers with some predefined field values. Each rule is represented as a pair $r = (RID,\ F)$ in which RID is a unique rule ID with it's uniqueness being in reference to the NGW's network and by extension all the DEVs connected to the NGW. While F is a set of field descriptors that dictate the expected value that a field should contain, as well as the method in which fields are checked for the expected values and how to compress them if the check is successful.

Field Descriptors

Finally, A field descriptor is a 7-tuple $f = (FID,\ FL,\ FP,\ DI,\ TV,\ MO,\ ACT)$ in which the FID is an identifier for the field's type (i.e. IPv6 source address, UDP checksum, etc), FL refers to the size of the field in bits, FP is a position identifier used to distinguish between fields that repeat in the same header (i.e. CoAP option

fields), DI is the direction of the packet as referenced in section 2.1, TV is a set of target value(s) that the field is expected to have (can either be a single value or a vector), MO is a comparison operation performed on the field and the TV to decide if compression is allowed, and ACT is the compression and decompression action to perform on the field.

2.2.1 Packet Classification

Packet Classification refers to the process of determining the subset of rules that a packet can be compressed by. The compressor iterates over all the rules in the chosen context, and for each rule it first performs a set of general checks that do not require looking into the content of the packet's fields, once the rule passes all the general checks we can then perform the matching operation, which requires a deeper level of inspection on the packet's header.

General Checks

The compressor begins by ensuring that for every field in the packet's header there exists a corresponding field descriptor with a matching FID. It then checks that the matched field descriptors have a DI value that matches the traffic direction, that is, it is either set to "UP" for up-link traffic, "DW" for down-link traffic or "BI" for all traffic directions. Finally, the compressor checks for the existence of repeated fields in the packet's header, this check involves looking into the FP value of the matched field descriptors, if FP is set to 0 then the existence of repeated fields is not considered, thus all repeated fields are matched to the same field descriptor. On the other hand if there exists a field descriptor $f \mid f[FP] = j > 0$ then there must exist a j^{th} repeated instance of the field in the packet's header.

The SCHC standard requires that a rule passes all the general checks, thus at any point when a check fails that rule is immediately disregarded and the compressor moves to checking the next rule in the context.

Matching Operations

Once a rule passes all the general checks, we will have it that every field in the packet's header will be matched to an appropriate field descriptor in that rule. Now

for every field and its matched field descriptor, the compressor will apply the matching function defined by the descriptor's MO value on both the field and the descriptor's target value(s). SCHC defines 4 of matching functions that the field descriptor can use:

- **Equal:** A direct equality comparison between the field's value and the descriptor's TV; requires the TV to be a single value.
- **Ignore:** No comparison is performed, always returns True.
- **MSB(x):** The compressor compares the x most significant bits in the field against the TV; requires the TV to be a single value.
- **match-mapping:** Checks if the field is **Equal** to any of the values stored at the TV; requires the TV to be a vector.

Note that the SCHC standard makes no further restrictions on rule selection beyond the aforementioned checks. Further, applying both the general checks and the matching operations does not guarantee that only a single rule is valid. Thus different implementations might want to test all the rules in the context first then choose one with the highest compression ratio such as those with highly constrained links, while other implementations might prefer to short circuit the checking process and choose the first matching rule when.

2.2.2 Compression/Decompression Modes

Before discussing SCHC's compress process, we must first cover the different compression functions (often referred to as actions) that the standard defines. Compression functions in SCHC can either completely omit a field, in which the compressor avoids transmitting the field, knowing that the decompressor knows what that field's value is based on the compression rule used, or the compression function may replace the field with a different (often smaller) value called the field residue which will be sent to and used by the decompressor to derive the original field. The 7 actions defined by the standard are:

- **not-sent:** At compression the field is completely omitted with no residue produced. The decompressor uses the TV stored in the rule as the field's value.

- **value-sent:** The field is not compressed, instead it is fully sent as a residue and used by the decompressor as is.

- **mapping-sent:** This action occurs in conjunction with the match mapping MO, in which the compressor sends the index of the field's value as found in the TV vector. The decompressor then uses the index to retrieve the value from the TV vector.

- **LSB:** Again, this action is only used in conjuncture with the MSB(x) MO, in which the compressor send the field's $f[FL] - x$ least significant bits as residue, which the decompressor prepends with the MSB stored in the descriptor's TV.

- **compute:** This action is kept for fields used in integrity checks (i.e. UDP checksum), in which the output of the integrity check (often a hash value) is used as the field residue.

- **DevIID and AppIID:** Both these actions serve the same purpose in which the compressor omits the 64 least significant bits in either the DEV's or the NGW's IPv6 address resp. for DevIID and AppIID. The decompressor uses the data-link layer address as the IID (interface identifier)

Field Residue

We mentioned that certain compression functions produce field residues that are required by the decompressor to derive the original field value; a caveat of using a system of residues is the need for additional metadata in cases when the residue size is variable, this occurs when using the value-sent or compute actions. In this case the compressor must prepend the residue with it's size which is encoded as either a 4 or 12 or 28 bit 2's complement integer, with the encoding being dictated by the descriptor's $f[FL]$ value.

2.2.3 Compression

Given a compression rule and a packet header, the compressor begins by creating a new **compressed** packet and prepending the selected rule's RID to it . It then

iterates over all of the rule's field descriptors based on their order in the rule definition, for every field descriptor we apply the compression function (dictated by the descriptors $f[ACT]$ value) on both the field's value and the descriptors TV. In the case where the compression action returns a residue, we append it to the **compressed** packet, otherwise move on to the next field descriptor. Figure 2.2 represents the final compressed packet.

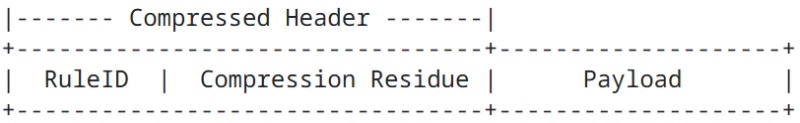


Figure 2.2: SCHC compressed packet. Reproduced from RFC 8724 [6]

2.2.4 Decompression

The decompression process is a simple inversion of the compression step. The decompressor begins by reading the RID at the start of the compressed packet, which points it to the rule used, it then iterates over the rule's field descriptors in the order that they are defined in (just as in compression). For every descriptor with an $f[ACT]$ that produces a residue the decompressor reads in the next residue block from the compressed packet and uses it in the decompression action defined by the descriptors $f[ACT]$ value.

2.2.5 Compression Ratios

SCHC's compression framework has the capability of producing large compression ratios, with studies reporting ratios of around 45% - 54% decrease in the size of CoAP/UDP/IPv6 headers as well as a 76% - 95% decrease on CoAP/UDP headers [12]. We observe however that these compression ratios are unilaterally dependent on the quality of the SCHC rules at predicting the application's packets. Take for example the case of the sensor DEV (see section 2.1.1) given knowledge of the DEV and NGW's IPv6 addresses as well as the application running on APP, a user is able to define SCHC rules where the IPv6 address, UDP ports and CoAP methods are predefined. Thus the output of compression need only contain the RID, integrity checks and payload. This means that a header with minimum size of 60 bytes can now

be represented in as little as 3 bytes. On the other hand, any uncertainty regarding the IPv6 addresses can result in the transmission of those fields uncompressed (as a residue), resulting in the compressed header with a minimum size of 35 bytes.

2.3 Mobility Limitations of SCHC

While SCHC has demonstrated the capacity to provide impressive compression ratio's when provided with a comprehensive rule set, its dependence on the persistence of the network's state comes with major drawbacks - mainly it's inability to adapt to certain changes in network configurations that may occur between the DEV and the NGW.

Take for example the use of SCHC in a communication in which the IoT device is mobile and moves between different networks. While existing standards such as MIPv6 (Mobile IPv6) may be used to ensure that IoT devices keep their IP addresses even when moving between networks, and thus maintaining their context. The use of such protocols in low power IoT devices is currently infeasible given the long hangover latency, intense signaling and packet loss attributed to them [3].

That is to say, we have proof of the advantages of applying SCHC to mobile based IoT applications such as that demonstrated by [11] however the actual deployment of SCHC into applications with changing network configurations comes at the cost of losing the ability to compress large parts of the packet, such as the source and destination IPv6 addresses which are by definition variable in these use cases.

Chapter 3

Dynamic Rule Updating

3.1 Previous Work

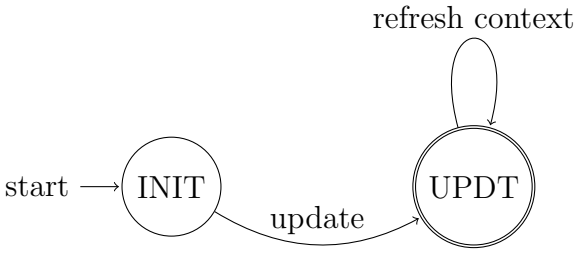
The idea behind using dynamic context synchronization is not novel, and was mention by [12] in their survey of header compression protocols with reference to learned or negotiated context exchanges. Moreover [8] examined the possibility of using context registration in which a DEV would sends requests to the NGW to create new rules, this however requires additional infrastructure with added complexity at the DEV. Finally, the groundwork for exchanging contexts have already been developed with existing SCHC implementations such as openSCHC including mechanisms to convert SCHC contexts into the YANG data model which can then in turn be compressed via CBOR, and transmitted via the SCHC channel uncompressed (though fragmentation is likely required).

3.2 Algorithm

With the excessive overhead of flow based rule updating of ROHCv1 [7] (see section 2.2) and the relatively high computational complexity of ROHCv2 via it's context state machine, relative to LPWAN networks [12]. We aim for a rule updating mechanism that only makes use of the existing SCHC infrastructure with an emphasis on limiting protocol operations at the DEV. It follows then that with the availability of the a context exchange mechanisms (see section 3.1), what remains is a simple algorithm that relies purely on the NGW to decide on which rules to use as well as when to update said rules.

On a high level the procedure can be thought of as a continuous transition between two states that is occurring at the NGW, an initiating state where no dynamic updating is yet to take place with either static rules being used or no compression, and an update state which is transitioned to after the first update and repeats indefinitely

to continue updating the rules as per an update frequency parameter.



3.2.1 Weight assignment

Given our aim of limiting the involvement of DEVs in rule updating and relying on a central controller of rules. We notice that these restrictions draw some parallels to the interaction between CPU schedulers, CPU time and processes. In which the NGW can be thought of as a rule scheduler, with SCHC rules being the CPU time and network traffic being the processes that we want to assign rules to. As such build on this framework by using priority based scheduling and introduce mechanisms for assigning and ordering priorities to different traffic flows to decide on which header fields occurs frequently enough to warrant the creation of a new rule that best fits them.

First, the NGW maintains fixed sized priority queues for every field we wish to dynamically update, in the case of mobile networks the source and destination IPv6 addresses are sufficient since the remaining fields are not affected by mobility. The entries in each queue corresponds to a 3-tuple (value, [ID], P) where value is the IP address, ID is a list of identification values we assign to packet flows, and P is the priority value.

The NGW also defines two variables, an update and aging frequencies, which dictates the number of packets that are forwarded by the NGW between the DEV and APP before a rule update should occur, and before the priority values in the queue are aged (decreased) allowing for more adaptive responses to network changes.

For every packet that the NGW receives it checks the source and destination address, packets with the same source and destination addresses are assigned the same ID. Once a packet is parsed, the priority queues are updated by increasing the priority of the entry with the respected address as value (or adding it if it is not in the queue) and updating the list of IDs for the specified address.

Finally, the frequency parameters are checked to either trigger queue aging or rule updates.

3.2.2 Choosing Rules

Once the update frequency parameter triggers a rule update the following steps take place:

First, the x packet flows (by ID) with the highest cumulative priority value across all the queues are selected , these will be the top level compression rules in which there will be a new rule for each flow, with the **not-sent** compression action being used for the specified fields, allowing for maximum compression of both the source and destination addresses.

Next, find the y top queue entries by priority that do not share an ID with any of the x top level rules. Once y values are selected from each queue, a set number of second level rules are created in which subsets of the chosen y values are grouped together in their corresponding field descriptor's TV with the compression action being set to **match-mapping** thus partial compression occurs with the index of the address relative to the TV being sent as field residues.

3.2.3 Rationale

Assumption

An explicit assumption made while defining the rule update procedure is that of a reasonable levels in variability. So while the expected traffic that SCHC operates on is expected to be mostly persistent, the update procedure still assumes some level of stability in the type of traffic but loosens the requirement to allow for some variability.

Explanation

By assigning the top x flows their own rules with the maximum compression ratio possible we make it so that the most frequently sent packets have the smallest size. On the other hand by assigning the y second level fields rules with index matching,

we make it so that less frequent but still used packet fields are also compressed, be is at a slightly lower compression ratio.

The main reason for the distinction between the top x and y values is aimed to limit the total number of rules used. Each rule represented in YANG format and compressed by CBOR is approximately 400 bytes which is multiple orders of magnitude larger than an uncompressed packet, as such we want the updates to be small and less frequent. In addition, more rules means that a larger number of bits are required to uniquely assign RID values increasing the minimum possible size of all SCHC packets.

Chapter 4

Methodology

4.1 Testbed and Workflow

All tests are performed in an emulation environment provided by the OpenSCHC Virtual Machine in [10]. We begin by provisioning three devices, a DEV representing the low-power IoT device, an NGW representing an LPWAN network gateway, and CoAP server. The CoAP server has access to a set of 10000 CoAP/UDP/IPv6 packets (see section 4.2) which it transmits to the DEV via the NGW. The NGW runs both the SCHC compression mechanism and the rule updating, it does this in the same program by first performing the compression then using the output of the parsing performed during compression as input to the rule updating procedure. Finally, the DEV runs a standard SCHC de-compressor. We circumvent the actual transmission of the context updating YANG messages by having the DEV and NGW share the same rule database and instead incur a penalty whenever a rule update occurs (see section 4.3.2) since the actual implementation of rule exchange is out of the scope of the study.

4.2 Dataset

To test the performance of dynamic updating based on the different levels of network variability, we need to have direct control over the distribution of packets that are being transmitted. To do this we define a sample space S as a set of 80 randomly generated CoAP/UDP/IPv6 packet headers each sized between $57 - 64$ bytes. We then sample 10000 packets from S with replacement using two sampling distributions:

- **Dataset A:** A heavily skewed and narrow beta distribution with $\alpha = 2, \beta = 40$ and a $\sigma \approx 0.05$.
- **Dataset B:** A less skewed beta distribution with $\alpha = 10, \beta = 10$ and a $\sigma \approx 0.48$.

Dataset A simulates a stable network with little to no variability in the header fields of the packets being transmitted. Dataset B simulates a slightly more variable network as we increase the standard deviation.

4.3 Evaluation

4.3.1 Control Group

For every dataset we begin by filtering out the $x + y$ most repeated packet headers and create static rules for them with the compression action set to **not-sent** allowing for maximum compression. We then transmit the packets from the APP to the NGW and log the original vs compressed size of every packet.

4.3.2 Dynamic Updating

To evaluate the performance of the rule updating procedure, we begin by setting the NGW to the initiating state with no static compression rules defined, we then transmit the dataset packets from the APP to the NGW, the NGW performs the rule updating procedure then attempts to compress the packets, logging the size of the original and compressed headers. Finally the compressed packet is transmitted to the DEV.

Whenever a rule update is triggered the NGW also logs in the size of the context's CBOR representation which needs to be transmitted for the update to occur, and thus counts as a penalty cost for performing an update.

Chapter 5

Results and Analysis

In this section we focus on comparing the size, in bytes, of the compressor's output traffic, flowing from the NGW to DEV for each of the datasets on both the Control (static) and dynamic implementations of SCHC rules.

5.1 Direct Comparison

By analyzing the cumulative average of the size of packets flowing from the NGW to the DEV (the compressor's output) in figures 5.1 and 5.2. We first observe that the average size of compressed headers for the control (static) implementation on datasets A and B differ by approximately 10 bytes this confirms our initial analysis on the static rule's performance being heavily dependent on the stability of the network traffic, that is, the static rules performed considerably better with lower levels of variability in the packets sent.

On the other hand, the dynamic rule updating showed a positive correlation between improved compression ratios and the number of packets being compressed, indicating that the updating heuristics perform very well for both distributions.

Moreover, the use of dynamic updating showed a considerable decrease of approximately 67% in the less variable Dataset A and 72% in Dataset B.

	Dataset A	Dataset B
Control	43 bytes	54 bytes
Dynamic	14 bytes	15 bytes

Table 5.1: Total Average Header Size

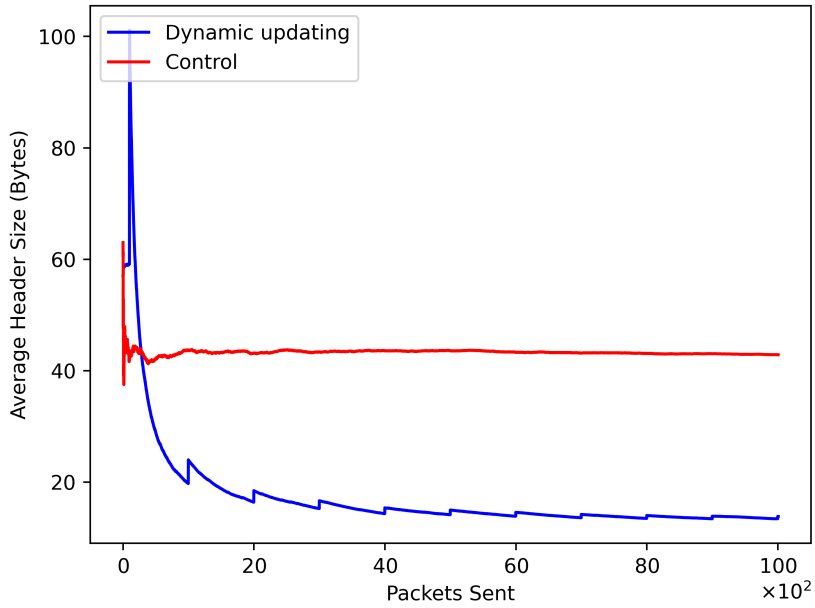


Figure 5.1: Cumulative average of compressed header size on Dataset A

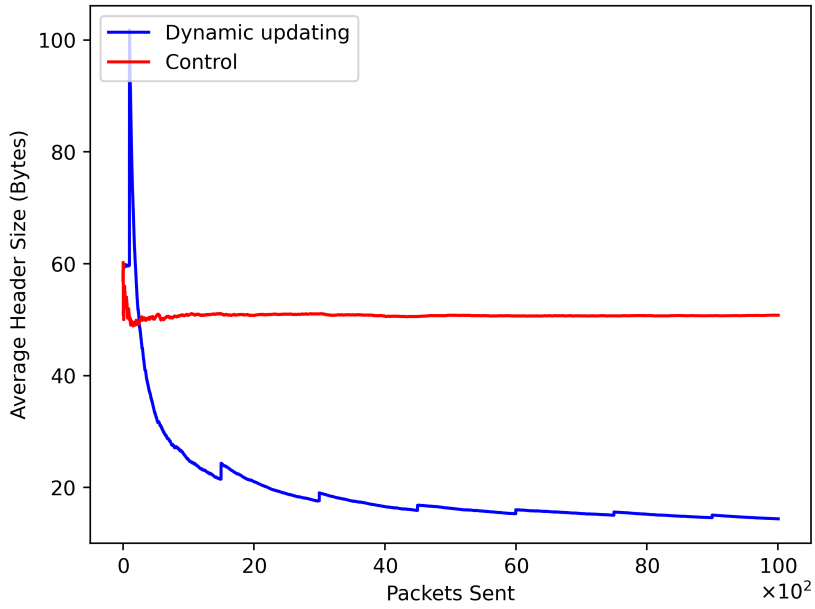


Figure 5.2: Cumulative average of compressed header size on Dataset B

5.2 Limitation

While the use of the rule updating procedure showed significant decrease in the average packet size. This came at the cost of large occasional spike as seen in the initial surge in Figures 5.1 and 5.2.

Moreover, the surges in header size become more apparent when instead of a cumulative average we group packets into windows of size 10 and calculate the average size of the headers in each window, as seen in Figure 5.3.

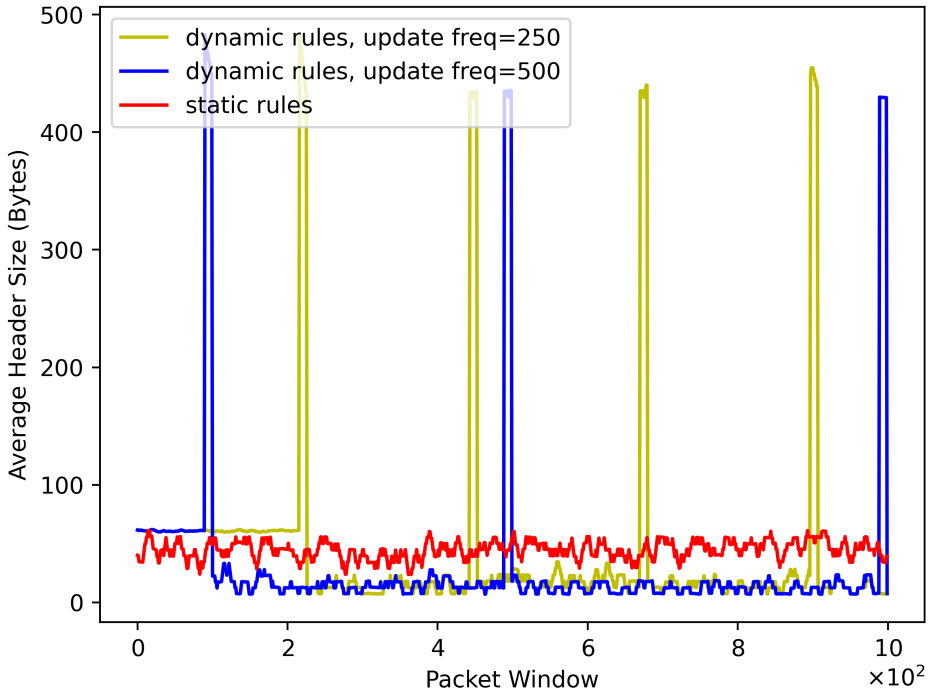


Figure 5.3: Windowed average of compressed header size on Dataset A

We note here that the extreme spikes seen in Figure 5.3 represent failed rule updating, in which a rule synchronization packet is sent from the NGW to the DEV to update the rules, however the next window of packets breaks from the trend that dictated the new rules. The probability of these spike and the frequency of updates naturally has a positive correlation.

While only 3 such spikes occurred over a 10000 packet exchange, the extremely constrained nature of LPWAN networks may not be able to efficiently deal with them,

as such careful assessment on the nature of the constrained link should be performed to ensure that spikes in header size do not become a choking point for the connection.

Chapter 6

Conclusion

6.1 Summary of Contribution

This thesis introduces the SCHC protocol's compression framework, which is considered to be at the cutting edge of header compression in the context of LPWAN networks, and highlighted the rational and assumptions behind certain aspects of its design with regard to context sharing. We then raise some of the concerns regarding the deployment of SCHC in mobile networks and introduced a simple mechanism as a proof of concept for the possibility of integrating dynamic context updating which we argue would improve SCHC's performance when used in mobile IoT applications.

6.2 Possible Improvements

While the current implementation of the rule updating presented in this thesis was able to show the possible benefits of dynamic updating. It also proved to be too simplistic in nature that it repeated some of the issues and limitations facing the previously mention RoHC protocol (see section 2.2).

Moreover, the deployment of rule updating outside of an emulation environment would require additional mechanisms that deal with malformed and missing rule synchronization messages, thus the development and real-world testing of such mechanisms is necessary to accurately gauge the performance of rule updating.

6.3 Future Work

The utilization of simple reinforcement algorithms based on the network traffic as opposed to weight and scheduling heuristics may be an interesting pivot when considering the process of creating new rules. Moreover, given that SCHC is both a compression and fragmentation protocol, it would be beneficial to test the impact of

rule updating in selecting fragmentation rules, especially in relation to the different acknowledgment modes that SCHC fragmentation uses.

Bibliography

[1] Hussein Al Haj Hassan, Ali Krayem, Ivan Marino Martinez Bolivar, Laurent Toutain, and Alexander Pelov. SCHC over LoRaWAN, a Framework for Interoperable, Energy Efficient and Scalable Networks. In *WF-IoT 2023: IEEE 9th World Forum on Internet of Things*, Aveiro, Portugal, October 2023. IEEE.

[2] Stephen Farrell. Low-Power Wide Area Network (LPWAN) Overview. RFC 8376, May 2018.

[3] Safwan M. Ghaleb, Shamala Subramaniam, Zuriati Ahmed Zukarnain, and Abdullah Muhammed. Mobility management for iot: a survey. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):165, Jul 2016.

[4] Olivier Gimenez and Ivaylo Petrov. Static Context Header Compression and Fragmentation (SCHC) over LoRaWAN. RFC 9011, April 2021.

[5] LoRa Alliance Technical commitee. LoRaWAN Regional Parameters. LoRaWAN1.0.2 Specification, July 2016. Version 1.0.

[6] Ana Minaburo, Laurent Toutain, Carles Gomez, Dominique Barthel, and Juan-Carlos Zúñiga. SCHC: Generic Framework for Static Context Header Compression and Fragmentation. RFC 8724, April 2020.

[7] C. Gómez Montenegro, A. Minaburo, L. Toutain, D. Barthel, and J. C. Zúñiga. Ipv6 over lpwans: connecting low power wide area networks to the internet (of things). *IEEE Wireless Communications*, 27(1):206–213, February 2020.

[8] Bart Moons, Eli De Poorter, and Jeroen Hoebeke. Device discovery and context registration in static context header compression networks. *Information*, 12, 02 2021.

[9] Rodrigo Munoz-Lara, Sandra Céspedes, and Marcos Diaz. Schc over dts-iot: Performance of schc confirmation modes in satellite iot. In *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*, pages 678–683, 2024.

[10] Paul Oultry. *The Book of SCHC*. IMT Atlantique, 2024. Available at https://mooc.openschc.net/downloads/The_book_of_SCHC.pdf.

[11] Ramon Sanchez-Iborra, Jesus Sánchez-Gómez, José Santa, Pedro J. Fernández, and Antonio F. Skarmeta. Ipv6 communications over lora for future iov services. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 92–97, 2018.

[12] Máté Tömösközi, Martin Reisslein, and Frank H. P. Fitzek. Packet header compression: A principle-based survey of standards and recent research studies. *IEEE Communications Surveys & Tutorials*, 24(1):698–740, First Quarter 2022.