THE THESIS TITLE

by

Anas

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
Month Year

© Copyright by Anas, Year

optional, will fill later

Table of Contents

Abstract

Static Context Header Compression (SCHC) is an adaptation layer capable of achieving large compression ratios on upper layer protocol headers (ie: IPv6, CoAP) by exploiting the presistent and predictable nature of IoT networks to make use of predefined static compression rules that act as a blueprints for the expected network traffic in which a sender can avoid transmitting the entire packet when a matching blueprint is present and instead sends a pointer to the rule. The afformentioned properities that SCHC builds on however makes it unideal for use in mobile networks in which the metadata we want to compress is variable. We look into the prospects of introducing dynamic updating of rules by evaluating the performance of scoring/weight assignment heuristics on network traffic to predict rules with improved packet coverage. Our results show that in the presence of reasonable assumptions an overall improvement on the average header size is possible.[**?**]

Acknowledgements

Thanks to all the little people who make me look tall.

Chapter 1

Introduction

The recent boom of IoT based networks facilitated the need for a shared networking layer to enable seemless communication between various types of devices regardless of their underlying communication medium and protocols. As such IPv6 has been widley adopted to play that role cite IoV paper. Yet, this in itself introduced a new set of challenges, specifically for long-distance low-power communications (ie LPWAN networks) that have physical restrictions on the size of the maximum transmission unit (MTU) of which the size of an IPv6 header often exceeds. Moreover, the additional bloat introduced by the IP metadata result in longer time-on-air during message transmission thus increasing power consumption cite schc impact paper.

A solution to the aformentioned was introduced by the LPWAN working group at the IETF in the form of the SCHC protocol with mechanisms for compression and fragmentationcite rfc 8724. SCHC has recieved wide adoption, with the LoRa Allience choosing it as the IPv6 adaptation layer for LoRa based communications, as well as the emergence of a considerable range of studies examining it's use in mobility centered IoT networks, such as that of the Internet of Vehicles via IPv6 over LoRa cite IoV paper, and in Direct-to-satellite IoT networks (DtS-IoT)cite the dts-iot paper

We notice however that most of the testing done on SCHC in mobile networks required manual configuration, leaving an unexamined hole in the literature regarding the feasibility of implementing SCHC in mobile applications in which the most impactful metadata fields become uncompressable via SCHC's framework (ie: the source and destination IPv6 addresses).

This study aims to evaluate the use of weight assignment hueristics on network traffic to construct new SCHC rules based on the state of the end-to-end communication, thus providing a mechanism for dynamic rule updating.

This work is organized as follows. Chapter 2 is an overview of the SCHC framework. Chapter 3 describes the short comings of SCHC in mobile networks. Chapter

4 describes the packet scoring algorithm and the test bed used. Chapter 5 presents and discusses the results.

Chapter 2

Network Architecture - INCOMPLETE

Because SCHC is built with LPWAN in mind cite rfc 8873, there is an implicit assumption that the communication is between a low-power low-compute IoT device (DEV) and a more powerfull network gateway (Core).

Chapter 3

SCHC Compression Background

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 admin's 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 ≈ 60 *bytes* per packet, while a single SCHC context represented in YANG format and compressed via CBOR (the defacto-standard binary representation in IoT networks cite CBOR) results in a message of ≈ 400 *bytes*, this in itself makes flow based context synchronization (similar to that used in the ROHC protocol) unideal as we will need to compress 7 IPv6 packets per flow before seeing a net gain, which is unrealistic given that minimizing the time-on-air of communications is the main reason we perform compression in the first place.

3.1 Context Structure

Before compression, the sender must decide on a context that fits the current packet. To best understand how a context rule is choosen, it is benificial to cover the hierarchical structure of SCHC contexts.

3.1.1 Context

Each context is represented as a pair:

$$C \ = \ (CID, \ R)$$

- **CID** is a unique identifier for the DEV relative to all other devices connected to the Core.
- **R** is a set of Rules.

3.1.2 Rules

Similar to contexts, each rule $r \in R$ is represented as a pair:

$$r = (RID, \ F)$$

- **RID** is a unique identifier for the rule, relative to all rules in the Core's network.
- **F** is a set of Field Discriptors.

3.1.3 Field Discriptor

Finally, each Field Discriptor $f \in F$ is represented as a 7-tuple:

$$f = (\ FID, \ FL, \ FP, \ DI, \ TV, \ MO, \ ACT\ )$$

- **FID** is a Field identifier, used to identify the specific field that f representes (ie: UDP source port, IPv6 destination address)
- **FL** is the Field's length representing the size, in bits, of the header field that f represents (ie: 128 for an IPv6 source address)

- **FP:** The Field's position, used to distinguish between header fields that can have multiple instances in a single header. (ie: Uri-Path field in CoAP headers).

- **DI:** A Direction Indicator, can take one of 3 values {Up, Dw, Bi}. Indicating whether f can be applied to a packet based on the direction of traffic in relation to the DEV and the Core. In which "Up" refers to traffic from the device to the Core, "Dw" is downlink traffic moving from the Core to the device, while "Bi" applies the field to packets traveling in both directions.

- **TV:** The Target Value that the header field is compared against. Can be a scalar value or a vector depending on the MO used. With the types that the TV can take being dictated by the SCHC implementation.

- **MO:** Matching Operator, this dictates the comparision function to apply on the field against the target value. SCHC defines 4 functions:
  - **Equal:** a simple equality comparision, requires target value to be a scalar.
  - **Ignore:** always returns true.
  - **MSB(x):** compares the most significat x bits of the header's field against a scalar target value.
  - **Match-mapping:** returns true if the field matches any of the Target Values, requires TV to be a vector.

- **ACT:** Defines the compression/decompression action to be taken when the matching operation passes. SCHC defines 7 such actions:
  - **not-sent:** Compression omits the field value, with the stored target value being used at decompression. Used with the equal MO
  - **value-sent:** Compression keeps the field as is, with the recieved value being used at decompression.
  - **mapping-sent:** Compression outputs the index of the Target Value in the TV vector. Decompression uses the element at said index. Used with the match-mapping MO

- **LSB:** Compression omits the x most significant bits, outputting the remaining $FL-x$ least significant bits. At decompression the x MSB stored at the target value are concatenated with the recieved LSB. Requires the MO to be set to MSB(x)
- **App IID:** omits the 64 least significant bits from the Core's IPv6 address at compression, and use the Core's MAC address when decompressing.
- **Dev IID:** omits the 64 least significant bits from the DEV's IPv6 address at compression, and use the DEV's MAC address when decompressing.

3.2 Packet Classification

Packet classification refers to the process of determining the rule with which a packet header is to be compressed by, and is the first step in the compression process. The sender begins by determining the flow direction of the packet, in the context of SCHC this can either be:

- **Uplink:** DEV to Core.
- **Downlink:** Core to DEV.

Uplink traffic is the trivial case since DEV only stores a single context. On the otherhand, to compress Downlink traffic, the Core must search for the context with the CID corresponding to the desired DEV. SCHC puts no restriction on what determines the CID value, though it provides the option to use the DEV's MAC address (or any Layer 2 address) as a CID.

After deciding on a context $C = (CID, R)$, the compressor then performs the following checks on every rule $r \in R$:

1. Ensure that for every field in the current header, there exists a field descriptor $f \in r$ such that $f[FID]$ matches the field type.
2. Ensure that every matched descriptor f has a $f[DIR]$ value that matches the packet's direction.

3. Ensure the the j^{th} instance of the header field has a corresponding descriptor f where $f[FP] = j$. The descriptor $f \mid f[FP] = 0$ matches all values of j.

4. Once every field in the current header has a matching descriptor f that has the correct FID, DIR and FP. We then apply the matching operation $f[MO]$ on the field's value and the target value(s) $f[TV]$.

Applying the mentioned checks on every rule results in a valid set of rules $(r_1..r_n)$ that pass all the checks. The compressor can the choose any of the valid rules to use. In the case where non of the rules pass, then the packet is sent uncompressed using the default rule r_0.

3.3 Compression

The compression process follows a simple procedure once a rule is chosen. The compressor iterates over all the field descriptors in the order that they appear in the rule, and applies the Compression/Decompression Action defined in the descriptor's $f[ACT]$ to the field value.

Chapter 4

Conclusion