Constrained Application Protocol Profile for Robust Header Compression Framework

Mikko Majanen, Pekka Koskela and Mikko Valta

VTT Technical Research Centre of Finland Ltd
Email: firstname.lastname@vtt.fi

***Abstract*—Due to the extensive growth of Internet of Things (IoT), the number of wireless devices connected to the Internet is increasing and will continue to increase remarkably in the near future. In wireless networks, the available bandwidth is always restricted. Much of the bandwidth is consumed by protocol overheads, while the actual data payload may be only couple of bytes. In this paper, we propose to use header compression to minimize the protocol overhead, especially for IoT-based communication using the new lightweight Constrained Application Protocol (CoAP). We define a CoAP compression profile for Robust Header Compression (ROHC) framework and evaluate its performance in the name of compressed packet size, delay caused by the compression and decompression, behavior in (wireless) lossy links, and energy efficiency. In our tests, the packet size could be reduced by 91.2% at best by using the proposed header compression. The round-trip delay increased slightly due to the extra processing needed for compression and decompression; however in lossy links with bit error rates $\geq 10^{-3}$ the smaller packet size due to the header compression turned out to be extremely beneficial due to the smaller need for packet retransmissions. In single packet transaction in our test bed, the header compression increased the energy consumption by 2.5%. However, in lossy links energy may be saved due to the smaller need for packet retransmissions. Other possible scenarios for energy savings were also identified as future work items.**

Keywords–Constrained Application Protocol (CoAP); Robust Header Compression (ROHC); delay; energy efficiency.

I. INTRODUCTION

Due to the extensive growth of Internet of Things (IoT), the number of connected devices to the Internet is increasing and will continue to increase remarkably in the near future. For example, Gartner estimates that the IoT, which excludes PCs, tablets and smartphones, will grow to 26 billion units installed in 2020 representing an almost 30-fold increase from 0.9 billion in 2009 [1]. Most of the new IoT devices are wireless, communicating locally for example via Bluetooth Low Energy, IEEE 802.15.4, or IEEE 802.11ah. Usually, a gateway node is needed to connect this local wireless network into the Internet. The Internet connection is in many cases also wireless; it can be for example based on cellular networks (3G, 4G LTE) or satellite networks. In wireless networks, the available bandwidth is always restricted. Usually, the IoT devices are for example sensors that periodically report their data values to the cloud services in the Internet. Thus, the transmitted data may be only couple of bytes. The problem is the protocol overheads: at the network layer, IPv4 or IPv6 header takes 20 or 40 bytes, respectively, and TCP or UDP on the transport layer takes 20 or 8 bytes, respectively. On the application layer, Hypertext Transfer Protocol (HTTP), for example, can take easily over 40 bytes.

Constrained Application Protocol (CoAP) [2] is a new proposed standard for a lightweight application layer protocol. It can be thought as a lightweight HTTP that can connect, for example, IoT devices to the Internet. CoAP has a small header overhead and it works on top of UDP, so it has considerably smaller protocol overhead than, for example, HTTP running on top of TCP.

For the IoT devices using CoAP, for example, a sensor node, a very common behavior is to send periodically the sensor data to the server, or vice versa, the server (or user) periodically asks the data value from the sensor. The header part of the packet remains almost constant all the time while only the data part changes. Since there are possibly many wireless hops along the path from the sensor to the server or user, the header overhead consumes the wireless bandwidth much more than the actual transmitted sensor data.

In this paper, we propose to use header compression to further minimize the protocol overhead in wireless links. We define a CoAP compression profile for Robust Header Compression (ROHC) [3] and evaluate its performance in the name of compressed packet size and delay caused by the compression and decompression. We also study its behavior in (wireless) lossy links and its energy efficiency.

The rest of the paper is organized as follows: Section II explores the related work and Section III shortly presents the principles of ROHC. In Section IV we define the CoAP profile for ROHC and in Section V we evaluate its performance in a real test bed environment. Finally, Section VI concludes the paper with future work items.

II. RELATED WORK

Header compression is based on the redundancy between header field values within packets, and especially between consecutive packets belonging to the same flow. For example, many header fields remain constant between packets, or change according to a known pattern. Over a single link, not all that information is needed and part of it can be temporarily removed, i.e., the full IP packet has to be re-created on the receiving side of the link.

Header compression is not a new idea. The first header compression scheme was compressed TCP (CTCP) [4] developed in 1990. IP Header Compression [5] can compress IP, UDP and TCP headers. CRTP [6] adds RTP header compression ability to IP Header Compression.

6LoWPAN Header Compression [7] defines a scheme for header compression for IPv6 packets transmitted over IEEE 802.15.4 networks, i.e., packets transmitted in Low Power Wireless Personal Area Networks (6LoWPANs) [8]. The compression format relies on shared context to allow compression of arbitrary prefixes. The approach is able to compress IPv6 and UDP headers. Also, it is meant to be used only inside 6LoWPANs. [9] is an addition to 6LoWPAN

 ISBN: 978-1-61208-406-0

Header Compression that enables the compression of generic headers and header-like payloads next to IPv6 header, without a need to define a new header compression scheme for each new such header or header-like payload.

ROHC [3] is a general and extendable framework for header compression, on top of which profiles can be defined for compression of different protocol headers. Currently, there are ROHC compression profiles for RTP, UDP, IP, UDP-Lite, ESP and TCP protocols. Especially, RTP profile is widely used in multimedia transmissions. ROHC has been incorporated into 3G and WiMAX network specifications and standards. Its performance has been evaluated, for example, in [10] and [11]. As can be seen, ROHC does not support CoAP protocol yet. In the next section, we shortly summarize the principles of ROHC, and then we define a missing CoAP profile for ROHC in Section IV.

III. Robust Header Compression (ROHC) principles

ROHC consists of a compressor and a decompressor located on the ends of a link with limited capacity. Redundant information in packet headers is transmitted only in the first packets; in the next packets, only dynamic header parts are transmitted. Packets are classified into flows to take advantage of inter-packet redundancy. A compression profile defines the rules how the packet headers are compressed.

ROHC has three operational modes [12]:

- In **Unidirectional Mode** the packets are sent only in one direction, from compressor to decompressor. This mode is always used in the beginning.
- In **Bidirectional Optimistic Mode** the decompressor uses feedback channel to send error recovery requests and acknowledgments of significant context updates to the compressor.
- In **Bidirectional Reliable Mode** the feedback channel is used more intensively. It aims to maximize robustness against loss propagation and damage propagation, i.e., minimize the probability of context invalidation, even under heavy loss/error burst conditions.

The ROHC compressor has three states [12]:

- The compressor starts in the **Initialization and Refresh (IR)** state, in which it sends full header header information.
- When the compressor is fairly confident that the decompressor has received the static header information correctly, it may proceed to **First Order (FO)** state. In this state, the compressor sends all the irregularities in the packet flow.
- In **Second Order (SO)** state, the compression is optimal. The change patterns of all dynamic header fields are fully exploited.

The ROHC decompressor has three self-explanatory states: **No Context**, **Static Context**, and **Full Context**. The used ROHC packet type depends on the current state of the compressor and decompressor. An interested reader can take a look at [3] for more details on ROHC.

IV. Constrained Application Protocol (CoAP) profile for ROHC

The first step in the header compression consists of identifying and grouping packets together into different ”flows”, so that packet-to-packet redundancy is maximized in order to improve the compression ratio [3]. Grouping packets into flows is usually based on source and destination host (IP) addresses, transport protocol type (e.g., UDP or TCP), process (port) numbers, and potentially additional unique application identifiers, such as the synchronization source (SSRC) in RTP. The compressor and decompressor each establish a context for the packet flow and identify the context with a Context Identifier (CID) included in each compressed header.

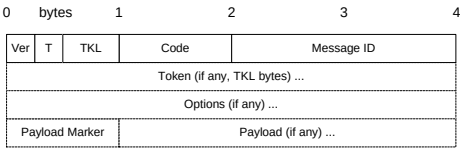

Figure 1. CoAP message format [2]

For the sensors using CoAP protocol, a very common behavior is to send periodically the sensor data to the server, or vice versa, the server (or user) periodically asks the data value from the sensor. CoAP PUT and GET request messages are used for these two, respectively. So, the PUT or GET request messages form one flow, and the corresponding responses to these requests (i.e., the ACK/RESPONSE messages) form another flow in the opposite direction.

The second step is to understand the change patterns of the various header fields. On a high level, header fields fall into one of the following classes [3]:

- INFERRED: These fields contain values that can be inferred from other fields or external sources, for example, the size of the frame carrying the packet can often be derived from the link layer protocol, and thus does not have to be transmitted by the compression scheme.
- STATIC: Fields classified as STATIC are assumed to be constant throughout the lifetime of the packet flow. The value of each field is thus only communicated initially.
- STATIC-DEF: Fields classified as STATIC-DEF are used to define a packet flow as discussed above. Packets for which respective values of these fields differ are treated as belonging to different flows. These fields are in general compressed as STATIC fields.
- STATIC-KNOWN: Fields classified as STATIC-KNOWN are expected to have well-known values, and therefore their values do not need to be communicated.
- CHANGING: These fields are expected to vary randomly, either within a limited value set or range, or in some other manner. CHANGING fields are usually handled in more sophisticated ways based on a more detailed classification of their expected change patterns.

The CoAP message format [2] is depicted in Figure 1. As can be seen, a CoAP message consists of a fixed length header information, followed by a (possibly zero-length) token, and optionally options and payload. The option format is depicted in Figure 2.

Version (Ver) is a 2-bit unsigned integer indicating the CoAP version number. Currently, it has to be set to 1, so it can be treated as STATIC-KNOWN in ROHC for now. In the

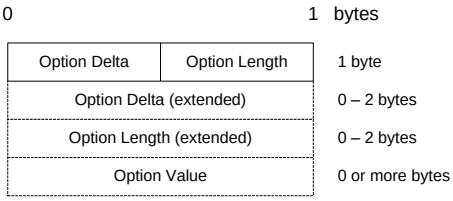

Figure 2. CoAP option format [2]

future, there might be other versions of CoAP, so the version number could then be STATIC.

Type (T) is a 2-bit unsigned integer indicating if this message is of type Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3). As said in the first step, the type of the packet is used to define the flow, so the Type could be treated as STATIC-DEF.

Token Length (TKL) is a 4-bit unsigned integer indicating the length (0-8 bytes) of the **Token** value. Token value is used for correlating requests and responses. The server shall use the same token in the response as it was in the request. The tokens currently in use for a given source/destination endpoint pair shall be unique. Thus, when sending the requests sequentially and having piggy-packed responses, the token can be kept as constant (or even as zero-length). In that case, the Token and Token Length can be considered as STATIC-DEF. In other cases, it should be treated as CHANGING.

Code is a 8-bit unsigned integer indicating if the message carries a request (1-31) or a response (64-191), or is empty (0). In case of a request, the Code field indicates the Request Method; in case of a response a Response Code. Code can be used to define the flow, so it can be treated as STATIC-DEF.

Message ID is a 16-bit unsigned integer that is used for the detection of message duplication, and to match messages of type Acknowledgement/Reset and messages of type Confirmable/Non-confirmable. The same Message ID should not be re-used in the communication with the same endpoint within EXCHANGE_LIFETIME, so the Message ID should be treated as CHANGING. An Acknowledgement or Reset message related to a Confirmable message has to repeat the same Message ID as in the Confirmable message.

Option Delta is a 4-bit unsigned integer indicating the difference between the **Option Number** of this option and the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta fields of this and previous options before it. Values 13-14 together with the Option Delta extension field are used for deltas larger than 12, and the value of 15 is reserved for the payload start marker (see below).

Option Length indicates the length of the **Option Value**, in bytes. Normally Length is a 4-bit unsigned integer allowing value lengths of 0-12 bytes. For longer options, values 13-14 together with the extension field are used. Length of 15 is not allowed.

Value is a sequence of exactly Option Length bytes. The length and format of the Option Value depends on the respective option. Option Value may be an unsigned integer (uint), a string, opaque, or empty.

Options include for example the Uri-Host, Uri-Port, Uri-Path, Uri-Query, Content-Type, and some matching options (If-Match, If-None-Match) [2]. Uri-Path and Uri-Query specify the path to the resource requested, which can be treated as STATIC-DEF in the flow. Content-Type is usually constant in the flow (e.g., sensor sending periodically the same measurement data), so it can be treated as STATIC-DEF. Also other options can usually be treated as STATIC-DEF in the flow. The only exception is perhaps the Location-Path that is normally used in a response to POST method and that defines the location path for the newly created resource, and thus it is CHANGING. But remember that POST is only used when creating the resources in the beginning. After that, the resources are updated by using PUT method. So POST is used much more seldom than PUT, and it is perhaps not wise to create a flow for POST messages at all, only for PUT and GET messages and their responses. Thus, basically all the options in the CoAP PUT and GET flows can be treated as STATIC-DEFs. Moreover, all the options can be treated as one combined bundle; there is no need for handling every single option as separate in compression/decompression.

Payload is preceded by the **Payload Marker** (0xFF). This marker can be treated as STATIC-KNOWN. It can be handled together with the options. Payload naturally changes and it is not part of the packet header, so header compression is not used for the payload.

Our CoAP profile was implemented for the open source ROHC library [13]. A version 1.6.1 of the ROHC library was used as the basis for the implementation. Basically, the CoAP profile had to implement the CoAP context for the flow, profile and flow identification function, and rules for compressing and decompressing the CoAP header part of the packet. The CoAP context for the flow basically consists of one CoAP header belonging to the flow's packet, in addition to the CID. The profile and flow was identified by using IP addresses, UDP ports, and CoAP Type and Code fields, i.e., the fields defined as STATIC-DEFs earlier. The compression of CoAP header part was based on the static and dynamical header parts discussed earlier in this section. The UDP and IP header parts of the packet were compressed as in the existing UDP/IP profile.

V. Evaluation of the CoAP profile

The performance of the developed CoAP profile was tested in a real test bed environment. First, we tested the compression ability, i.e., how much the packet headers could be compressed by using the CoAP profile. Secondly, we measured the delay caused by the compression and decompression. Thirdly, we studied the performance of header compression in lossy (wireless) links. Finally, we studied the energy efficiency of the header compression.

A. Test bed and testing scenario

Our test bed (see Figure 3) consisted of an old laptop connected to the VTT's CNL laboratory's Willab network via WiFi and a Raspberry Pi (RPi) computer connected to the Willab using Ethernet. An UDP-tunnel was established between the laptop and the RPi by using the tunnel application provided by the ROHC library for easy testing of the library. It is to be noted that the tunneling was used only for practical testing reasons since we do not have a real ROHC implementation integrated in the protocol stack in the both ends of the link. The laptop served as a CoAP server running the example CoAP server software provided by the C-based libcoap-4.0.3 library [14]. The RPi acted as a CoAP client requesting periodically the 'time' resource from the server

 ISBN: 978-1-61208-406-0

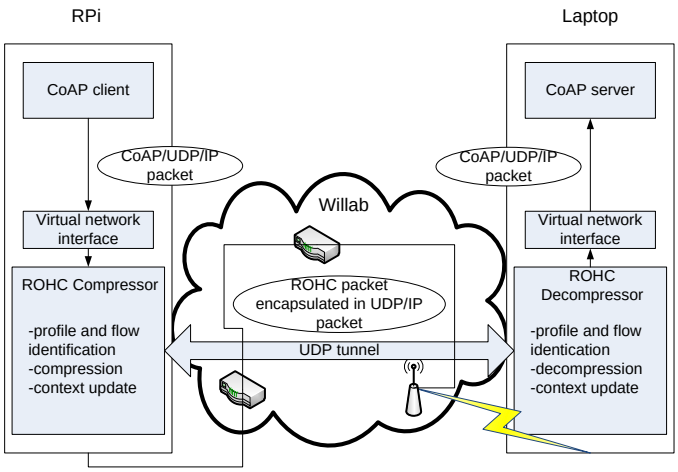

Figure 3. The test bed

using libcoap's example CoAP client software. Thus, it created a GET request flow to the server. The server responded to the requests by sending acknowledgements piggy-packing the CoAP RESPONSEs. The laptop had a 1.73 GHz Intel Pentium M processor and 1 GB of RAM. RPi was a Model B having a 700 MHz ARM CPU processor and 512 MB of RAM.

In the energy-efficiency studies, we used 2 RPis, one as a server and another one as a client. RPis were communicating via an IEEE 802.11g WLAN access point using 54 Mbit/s bit rate. We used D-Link DIR-300 as a wireless access point. The client RPi was connected to the access point with D-Link DWA-121 USB-WLAN dongle and the server RPi with Ethernet connection. The current as a function of time was measured from the client RPi with a current probe (AM503B amplifier) connected to the oscilloscope (Tektronix TDS3032B). The client RPi was Model B+, which is an improved version of prior Model B, having a lower power consumption, for example.

B. Compressed packet size

The created CoAP GET request message consisted of the following CoAP header fields:

- version = 1
- type = 0 (confirmable)
- tkl = 0 (no token)
- code = 1 (GET request)
- message ID = 58468 (changing number)
- option Uri-Path = 'time' in TLV format = 0xb474696d65; Type=11, Length=4, Value='time'
- no payload

In the beginning, the compressor starts in the ROHC Initialization and Refresh (IR) state and transmits full headers. The uncompressed packet size was 37 bytes consisting of IPv4 header of 20 bytes, UDP header of 8 bytes, and CoAP header of 9 bytes. One extra byte was needed for the CID in ROHC, so the compressed packet size was in the beginning 38 bytes. However, after couple of packets when the context was created and the compressor changed into First Order (FO) state, only the dynamical (changing) header parts were transmitted and the compressed packet size decreased to 15 bytes with the CoAP profile. The only dynamical part of the CoAP header is the Message ID that takes 2 bytes. So the 9-byte CoAP header was compressed to only 2 bytes, i.e., the savings in the CoAP header part were 77.8%. The rest of the compressed packet was used by the UDP and IPv4 headers. The savings for the whole packet were 59.5%. When the compressor changes into Second Order (SO) state, also the change patterns of the dynamical header parts are taken into account. Since the CoAP Message ID may vary totally randomly, there were no further saving possibilities in the CoAP header part. However, further compression was still available in the other (UDP and IPv4) headers, and the packet could be compressed to only 5 bytes at best. Thus, the savings for the whole packet were 86.5%. The achieved gains are equivalent to the gains of about 85% reported, e.g., in [11] and [15] for RTP profile.

The same packet flow was also compressed with the UDP/IP and IPv4-only profiles. With these profiles, the compressed packet sizes decreased to 22 and 28 bytes, respectively, in the FO state. The savings were 40.5% and 24.3%, respectively. UDP/IP profile compresses only the UDP and IP header parts, while the IP-only profile compresses only the IP header part. These profiles do nothing for the CoAP header part. In SO state, the IPv4 compression profile compressed the packet to 18 bytes at best, so the savings were only 51.4%.

The server's acknowledgement message piggy-packed also the response, and its length was 51 bytes uncompressed. IPv4 header took 20 bytes, UDP header 8 bytes, CoAP header 8 bytes, and CoAP payload 15 bytes. The CoAP header part consisted of the following header fields:

- version = 1
- type = 2 (ACK)
- tkl = 0 (no token)
- code = 69 (2.05 Content)
- message ID = 58468 (changing number)
- options Content-Format (Length = 0) and Max-Age (length 1 and value 1) in TLV format = 0xc02101;
- Payload Start Marker = 0xFF
- Payload = 'Oct 14 19:45:32'

For the ack/response packet, the header parts could be compressed as much as in the request messages, but the overall savings were smaller because the payload part could not be compressed. Some of the messages also contained ROHC feedback data. With the CoAP profile, the compressed packet size (including the payload of 15 bytes) was 53 or 58 bytes in IR state (depending on the amount of feedback data), 31 or 22 bytes in FO state, and 21 bytes in SO state. So the savings in the total packet size were 58.8% at best in SO state. With IPv4-only profile, the packet size could be compressed to 33 bytes at best, meaning 35.3% savings in total packet size.

We also studied the packet sizes of the same CoAP request and ack/response messages when using IPv6 protocol. IPv6 protocol header takes 40 bytes, i.e., 20 bytes more than IPv4. Thus, the uncompressed CoAP request packet size was now 57 bytes and the corresponding ack/response packet 71 bytes. In the IR state, the compressed packet sizes (with ROHC header and feedback information) were 61 or 68 bytes for the CoAP request packet and 75 bytes for the ack/response packet. In the next state, when only dynamical header parts were transmitted, the compressed packet sizes decreased to 12 and 27 bytes, respectively. Finally, in the SO state, the compressed packet sizes were only 5 and 20 bytes, respectively, with the developed CoAP profile. As percentages, these mean total savings

 ISBN: 978-1-61208-406-0

of 91.2% and 71.8% in the request and ack/response packet sizes, respectively. With the UDP profile, the SO state packet sizes were 12 and 26 bytes, respectively, meaning 78.9% and 63.3% savings. With IPv6-only profile, the SO state packet sizes were 18 and 32 bytes, respectively, meaning 68.4% and 54.9% savings. Compared to the IPv4 packet sizes, IPv6 results in larger packets in the beginning of the compression, but in the end, the packets can be compressed to about the same sizes (or even smaller) as with IPv4. Thus, the percentile savings in the packet sizes are actually bigger with IPv6 than with IPv4.

So, even if the CoAP header is designed to have only a small overhead, header compression can still make it even smaller. The used CoAP GET message was practically almost as small as a CoAP packet can be and the savings were 77.8% in the header part. In theory, if the CoAP packet does not have any options or token, the minimum size would be 4 bytes. This could be compressed to 2 bytes, so the savings are always at least 50% in the CoAP header part. If the CoAP Message ID field had a certain known change pattern, the compression could be enhanced still by 2 bytes.

C. Delay measurements

First, we measured the compression and decompression processing delay, using the same traffic as in packet size measurements, but now sent locally over loopback interface at the laptop and the RPi. So the laptop and RPi both ran the CoAP server and the CoAP client, and the communication was only inside the device. Tables I and II represent the processing delays due to the compression and decompression for different ROHC packet types at the laptop and RPi, respectively. As can be seen, the processing delay is bigger in the beginning when the context is created for the flow. After reaching the SO state and the smallest ROHC packet sizes, the delay is much smaller. It is to be noted that the IR state with longer delay is needed only in the beginning of the flow, or if the context needs to be updated for some reason (e.g., because of corrupted context due to the dropped packets). All the first 6 IR packets are reported individually since the packet size varies a little bit due to the feedback information, and also because the very first GET and RESP packets need more processing time due to the initialization of the flow context. There are two UOR-2 packets in each flow before switching to the SO state; the processing time does not differ between them. Also, in the SO state, the delay remains in the same level all the time. Each reported measurement in the tables is an average of at least 5 packets of that type. As can be seen, the total processing delay in the SO state is less than 0.3 ms at the laptop and 0.9 ms at RPi.

Next, we measured the round-trip delay from CoAP request sent from the RPi to the CoAP response from the laptop received at the RPi. As the ROHC library needs the tunnel application between the communicating devices, there is some extra delay caused by the tunnel itself. For that reason, we also measured the round-trip delay without the tunnel and ROHC by using only CoAP client and server to see what is the overhead caused by the tunneling. The measurements were done for both IPv4 and IPv6. The results are based on the average (Avg) of 20 packets and are depicted in Tables III and IV. Minimum (Min) and maximum (Max) values are also reported in the tables. As can be seen, if ROHC is used, the delay is the shortest with the uncompressed profile, i.e., when the packet is not compressed at all. This is because of the processing

TABLE I. ROHC COMPRESSION AND DECOMPRESSION PROCESSING DELAY AT LAPTOP.

ROHC state	*packet type ROHC/CoAP*	*packet size (bytes)*	*comp (ms)*	*decomp (ms)*	*total (ms)*
IR	IR/GET	38	0.322	0.365	0.687
IR	IR/RESP	58	0.215	0.283	0.498
IR	IR/GET	44	0.248	0.265	0.512
IR	IR/RESP	53	0.143	0.217	0.360
IR	IR/GET	38	0.175	0.235	0.409
IR	IR/RESP	53	0.143	0.228	0.371
FO	IR-DYN/GET	15	0.148	0.191	0.339
FO	IR-DYN/RESP	31	0.130	0.167	0.297
FO	UOR-2/GET	6	0.172	0.211	0.384
FO	UOR-2/RESP	22	0.112	0.164	0.276
SO	UO-0/GET	5	0.139	0.145	0.254
SO	UO-0/RESP	21	0.101	0.138	0.216

TABLE II. ROHC COMPRESSION AND DECOMPRESSION PROCESSING DELAY AT RASPBERRY PI.

ROHC state	*packet type ROHC/CoAP*	*packet size (bytes)*	*comp (ms)*	*decomp (ms)*	*total (ms)*
IR	IR/GET	38	1.506	1.618	2.976
IR	IR/RESP	58	1.837	1.009	2.846
IR	IR/GET	44	0.703	0.803	1.627
IR	IR/RESP	53	0.603	0.723	1.326
IR	IR/GET	38	0.578	0.688	1.266
IR	IR/RESP	53	0.511	0.811	1.321
FO	IR-DYN/GET	15	0.507	0.616	1.124
FO	IR-DYN/RESP	31	0.476	0.611	1.087
FO	UOR-2/GET	6	0.545	0.617	1.162
FO	UOR-2/RESP	22	0.428	0.584	1.012
SO	UO-0/GET	5	0.397	0.427	0.824
SO	UO-0/RESP	21	0.370	0.469	0.838

delay needed for compression and decompression. If any compression is used, we can see that it is then worth to compress all the packet headers, since the CoAP profile seem to have a smaller delay than UDP/IP or IP-only profiles. There is no difference whether we use IPv4 or IPv6, the order of the profiles remains the same. Without the tunneling between the laptop and the RPi, and without any ROHC profile, the delay is about 8 ms shorter than with uncompressed profile with IPv4, and about 3 ms with IPv6. So, that is the overhead caused by the tunneling and ROHC without any compression. It is to be noted that tunneling would not be needed if ROHC was implemented directly on both sides of the link; the tunnel is meant only for easy testing of the ROHC library.

TABLE III. ROUND-TRIP DELAY WITH DIFFERENT ROHC PROFILES (IN MS), IPV4.

Profile:	*CoAP*	*UDP*	*IPv4*	*uncompressed*	*no tunnel*
Avg:	20.194	22.368	21.511	18.824	10.627
Min:	6.769	7.480	7.261	5.962	2.209
Max:	37.577	34.496	43.166	34.596	25.325

TABLE IV. ROUND-TRIP DELAY WITH DIFFERENT ROHC PROFILES (IN MS), IPV6.

Profile:	*CoAP*	*UDP*	*IPv6*	*uncompressed*	*no tunnel*
Avg:	20.085	22.366	21.089	16.289	13.365
Min:	8.428	8.070	8.493	5.851	2.389
Max:	46.282	34.950	37.598	39.564	59.472

D. Performance over lossy links

We also studied the effect of smaller packet size to the delay and throughput in lossy (wireless) links. Every real

 ISBN: 978-1-61208-406-0

wireless link has bit errors. For example, for 3G links bit error rates 10^{-3} or even higher are possible [10]. The smaller packet has a smaller probability to have bit errors, so it has a bigger probability to go through the lossy link without errors, meaning less retransmissions, shorter delay, and greater throughput.

The tunnel application provided by the ROHC library supports setting a given bit error rate (BER) to the tunnel. We used it to generate bit errors with a given rate according to a uniform distribution. The tunnel application generates the bit errors on the sender side of the tunnel, and packets with bit errors are not actually transmitted through the tunnel. Retransmissions are then triggered at the CoAP client application because of a missing ACK to the sent message.

With the bit error rate of 10^{-4} (and less than that), the uncompressed profile was still slightly better than compressed profiles, since there were not so many errors. The round-trip delays were almost the same as in Table III, being 20.961 ms for the CoAP profile and 18.503 ms for the uncompressed profile. Tables V - VI represent the round-trip delays with the bit error rate of 10^{-3}. The results for each profile are based on the average (Avg) of 20 sent packets. Also minimum (Min) and maximum (Max) values are reported in the tables. With the uncompressed profile, 5 packets out of 20 were dropped totally due to the exceeding the number of maximum CoAP retransmissions (= 4 times); with other profiles, this did not happen due to the smaller packet size and hence less errors and retransmissions. As can be seen, with high bit error rates such as 10^{-3}, it is useful to use header compression, since it results in shorter delay, less dropped packets, and hence better throughput. Note that the delay increases from milliseconds to seconds! The difference between different profiles is also clear: it is beneficial to compress all the headers to get the packet size as small as possible. There was also a small difference between unidirectional and bidirectional ROHC modes: bidirectional mode (i.e, with feedback) seems to recover faster from corrupted flow context caused by dropped packets and hence it seem to have a slightly shorter delay.

TABLE V. ROUND-TRIP DELAY (S) AND PACKET LOSS WITH DIFFERENT ROHC PROFILES IN LOSSY LINKS, BER=10^{-3}, BIDIRECTIONAL MODE

Profile:	*CoAP*	*UDP*	*IPv4*	*uncompressed*
Avg:	1.040	1.715	2.873	22.342
Min:	0.008	0.007	0.007	0.019
Max:	8.048	8.275	14.324	44.452
#Dropped:	0	0	0	5

TABLE VI. ROUND-TRIP DELAY (S) AND PACKET LOSS WITH DIFFERENT ROHC PROFILES IN LOSSY LINKS, BER=10^{-3}, UNIDIRECTIONAL MODE

Profile:	*CoAP*	*UDP*	*IPv4*	*uncompressed*
Avg:	1.373	2.399	3.250	21.512
Min:	0.007	0.013	0.008	0.015
Max:	8.562	17.549	20.738	44.395
#Dropped:	0	0	0	5

The effect of BER to ROHC has also been studied, e.g., in [10] and [16]. They both show significant decrease in packet loss when the BER is high and ROHC is used. In both of these, RTP profile was used with video and voice applications, respectively.

E. Energy efficiency

In energy efficiency point of view, compression and decompression processing will consume extra energy. On the other hand, as a result of compression, the packet size is reduced, which will save energy during packet transmission.

Figure 4 depicts the current consumption on the client RPi during a single CoAP GET message sending and receiving a response from the server (as in earlier studies). The current was measured for both the CoAP and the uncompressed profiles. With the CoAP profile the packet size was 5 bytes, while with the uncompressed profile it was 38 bytes. The total energy consumption for the whole operation was 156.4 mJ for the CoAP profile and 153.0 mJ for the uncompressed profile. For this calculation, we used a time interval that was started when the current rises above 260 mA at about 14 ms and stopped when the current drops below 260 mA at about 115 ms (uncompressed) or 117 ms (CoAP profile). Thus, the CoAP profile seems to increase the energy consumption about 2.2% compared to the uncompressed profile. However, everything interesting (compression, decompression, transmission, reception) happens during the highest peak in the figure. Thus, if we only observe that part of the figure, we can say that it takes 15.9 ms from the uncompressed profile and 18.2 ms from the CoAP profile. The difference is 2.3 ms and it is because of the compression and decompression processing needed at the client and the server side. This is in line with our delay measurements, where it took little bit less than 1 ms for RPi to compress and decompress a packet. The difference in energy consumption is then roughly $2.3\ \mathrm{ms}{*}340\ \mathrm{mA}{*}5\ \mathrm{V} = 3.9\ \mathrm{mJ}$, which means a 2.5% increase. Note that the extra peak at 77 ms is due to the 802.11 beacon reception. That is why we used 340 mA in the calculation.

The energy savings due to shorter transmission time are insignificant because of the high speed of transmission. In case of uncompressed packet of 38 bytes and compressed one of 5 bytes, the theoretical transmission times with the speed of 54 Mbit/s are $5.6\ \mu$s and $0.7\ \mu$s, respectively, so the saving in transmission time is less than $5\ \mu$s. However, in lower speed networks, the packet size and transmission time may play a bigger role. Studying this remains as one of our future work items.

So, in these tests, the energy needed for compression and decompression processing was bigger than the possible energy saving due to the shorter transmission time. However, the increase in the energy was only about 2.5%. In lossy links with bit errors, this amount of energy can be easily consumed due to the retransmissions because of bit errors. So, let E be the energy needed for one transmission. For X transmissions, the needed energy is $X * E$. If the E increases 2.5% due to the header compression, then by solving X from $X * 1.025E \leq (X + 1) * E$, we get $X \leq 40$, which means that if there is a need for one retransmission at CoAP level in every 40 packets or less, then it turns out to be energy efficient to use the CoAP profile. In the future, we will test this in a real test bed.

In these tests, we observed only one sender and receiver. The smaller packet size due to the header compression saves the bandwidth in the network. This may also turn out to be energy efficient when there are more than one sender and more traffic, since smaller packets may result in less queuing and less packet collisions. Studying this will be one of our future

 ISBN: 978-1-61208-406-0

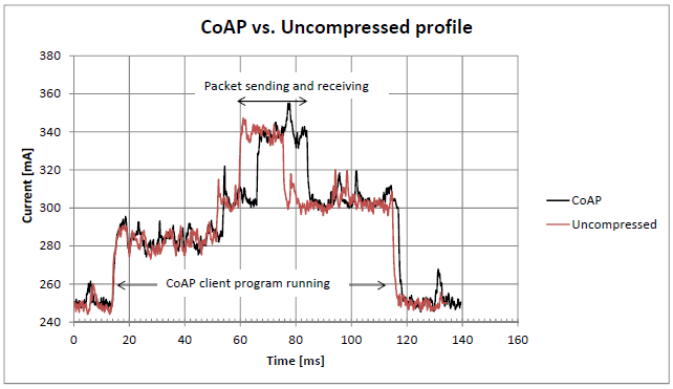

Figure 4. Current as a function of time at the client

work items, too.

VI. Conclusion and Future Work

This paper defined a CoAP profile for ROHC. Its feasibility was studied by measuring the compressed packet size and delay caused by the compression/decompression related processing, by studying its performance in lossy links, and by studying its energy efficiency. With CoAP profile and ROHC, the packet size could be reduced by 91.2% at best. It was found that the delay increased slightly due to the extra processing required for the compression and decompression. The total delay was less than 1 ms in SO state and less than 3 ms in IR state for a single packet compression and decompression at RPi. Due to the smaller packet size, the performance (delay, throughput) in lossy (wireless) links with bit error rates $\geq 10^{-3}$ was much better with the header compression than without. The energy efficiency was also studied. Due to the extra processing needed for compression and decompression, the total energy consumption increased 2.5% in a single CoAP message transaction. So, the savings in the shorter transmission time were not enough to beat the consumption of extra processing. However, in lossy links with bit errors, energy will be saved due to the smaller need for retransmissions. In our test bed, the threshold for energy efficiency was estimated to be around one CoAP level retransmission in every 40 packets.

Our future work items include energy efficiency measurements with different hardware and radio technologies. Especially interesting is to study the energy efficiency with radio technologies with much slower bit rates, such as IEEE 802.15.4-based XBee or ZigBee radios. We are also planning to study the energy efficiency and the performance of header compression in real lossy wireless links instead of using generated bit errors by the tunnel application. We are also thinking about possibilities to integrate ROHC directly to the both ends of the wireless link for avoiding the use of the tunnel application and its overhead. We will also study the effects of smaller packet size to the energy efficiency in a larger test bed with more nodes and traffic. The other ROHC profiles have been standardized, so standardization of the developed CoAP profile may be included in our future work, too.

Acknowledgment

This work was supported by TEKES as part of the Internet of Things program of DIGILE. The comments from the anonymous reviewers are also greatly acknowledged.

References

[1] "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020," 2013, URL: http://www.gartner.com/newsroom/id/2636073 [accessed: 2015-04-01].

[2] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," in RFC 7252. IETF, Jun. 2014, URL: https://datatracker.ietf.org/doc/rfc7252/?include_text=1 [accessed: 2015-04-01].

[3] K. Sandlund, G. Pelletier, and L.-E. Jonsson, "The RObust Header Compression (ROHC) Framework," in RFC 5795. IETF, Mar. 2010, URL: https://datatracker.ietf.org/doc/rfc5795/?include_text=1 [accessed: 2015-04-01].

[4] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," in RFC 1144. IETF, Feb. 1990, URL: http://tools.ietf.org/html/rfc1144 [accessed: 2015-04-01].

[5] M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression," in RFC 2507. IETF, Feb. 1999, URL: https://tools.ietf.org/html/rfc2507 [accessed: 2015-04-01].

[6] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," in RFC 2508. IETF, Feb. 1999, URL: http://tools.ietf.org/html/rfc2508 [accessed: 2015-04-01].

[7] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," in RFC 6282. IETF, Sep. 2011, URL: https://tools.ietf.org/html/rfc6282 [accessed: 2015-04-01].

[8] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," in RFC 4944. IETF, Sep. 2007, URL: https://tools.ietf.org/html/rfc4944 [accessed: 2015-04-01].

[9] C. Bormann, "6LoWPAN Generic Compression of Headers and Header-like Payloads (GHC)," in draft-ietf-6lo-ghc-05. IETF, Sep. 2014, URL: https://tools.ietf.org/html/draft-ietf-6lo-ghc-05 [accessed: 2015-04-01].

[10] A. Couvreur, L.-M. Le Ny, A. Minaburo, G. Rubino, B. Sericola, and L. Toutain, "Performance Analysis of a Header Compression Protocol: The ROHC Unidirectional Mode," Telecommunication Systems, vol. 31, no. 1, Jan 2006, pp. 85–98.

[11] 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, 2005, pp. 23–41. [Online]. Available: http://dx.doi.org/10.1007/s11277-005-7733-2

[12] C. Bormann (editor) and et.al., "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed," in RFC 3095. IETF, Jul. 2001, URL: http://tools.ietf.org/html/rfc3095 [accessed: 2015-04-01].

[13] "ROHC library," 2015, URL: http://rohc-lib.org/ [accessed: 2015-04-01].

[14] "libcoap: C-Implementation of CoAP," 2015, URL: http://sourceforge.net/projects/libcoap/ [accessed: 2015-04-01].

[15] M. Tömösközi, P. Seeling, and F. Fitzek, "Performance Evaluation and Comparison of RObust Header Compression (ROHC) ROHCv1 and ROHCv2 for Multimedia Delivery," in Globecom 2013 Workshop - Control Techniques for Efficient Multimedia Delivery, Dec. 2013.

[16] Effnet and Intel, "Effnet ROHC (Robust Header Compression) Performance on Intel Core Microarchitecture-Based Processors," in Technology Evaluation White Paper, 2007, URL: http://www.effnet.com/pdf/uk/19350_EFFNET_Final.pdf [accessed: 2015-04-01].

 ISBN: 978-1-61208-406-0