Using SCHC for an optimized protocol stack in multimodal LPWAN solutions

Bart Moons[1], Abdulkadir Karaagac [2], Jetmir Haxhibeqiri[3], Eli De Poorter [4] Jeroen Hoebeke[5]

University of Ghent - imec, IDLab, Department of Applied Engineering
[1]bamoons.moons@ugent.be, [2]abdulkadir.karaagac@ugent.be, [3]jetmir.haxhibeqiri@ugent.be,
[4]eli.depoorter@ugent.be, [5]jeroen.hoebeke@ugent.be

Ghent, Belgium

***Abstract*—Low Power Wide Area Networks (LPWANs) are formed out of cheap, small, interconnected devices which operate in the sub-GHz domain. The last couple of years, many communication technologies arose in this domain, each with its own characteristics. In order to satisfy more diverse requirements, devices are now equipped with multiple LPWAN radio technologies, which requires the use of a unified protocol stack independent of the underlying LPWAN technology. With its 2^{128} addresses available and its ability to operate over different link layer technologies, the IPv6 protocol stack would be the ideal candidate. However, many LPWAN configurations do not allow standardized IP/UDP communication, sometimes acquiring more header overhead than there is room for the actual payload. Recently, a new initiative to directly connect constrained devices over IP was initiated by the LPWAN working group of the Internet Engineering Task Force (IETF). This work resulted in the Static Context Header Compression or SCHC mechanism. This header compression mechanism is able to compress the overhead of these internet protocols up to 95%. In order to comply with the IPv6 Maximum Transfer Unit (MTU) of 1280 bytes, a fragmentation mechanism is also included. In this work, we validate the benefits of using SCHC for multimodal LPWAN solutions and show its implementation feasibility on such constrained devices.**

I. Introduction

Several problems arise when introducing the many embedded devices the IoT is expected to bring. To move away from complex gateways that translate between proprietary protocols and standardized internet languages, a shift to internet standards is needed to address the huge amount of devices. From an addressing point of view, the IPv6 protocol seems to suit this need. Combined with other standardized protocols like the Constrained Application Protocol (CoAP) and User Datagram Protocol (UDP), it may become one of the preferred ways to communicate with constrained devices. However, considering the severe energy, memory, processing and communication constraints of LPWA devices and networks, this stack is not a viable solution. The overhead brought by its headers is not in line with the few tens to hundreds of bytes which can be transmitted over these low power communication technologies. LPWANs are also limited by duty cycle regulations, as they operate in unlicensed spectrum, which limits the time on air and, consequently, the size of packet transmissions when operating at low bit rates.

Apart from that, more demanding and diverse IoT applications could require the devices to be equipped with multiple technologies. These devices could benefit from long range outdoor communication combined with higher data rate indoor communication. Also, the fact that LPWAN equipment may be shared across different organizations makes the concept of multimodal LPWAN solutions an interesting research topic.

When using a single application across different technologies, a single common stack is desirable. In order to pick the right stack, we evaluated different protocol stacks for use in multimodal LPWAN solutions, considering LoRa, DASH-7 and SigFox. We show that SCHC provides an efficient, adaptive fragmentation and compression mechanism, which can be used to connect small, multimodal LPWAN devices to the internet. As a next step, we also implemented the IETF SCHC protocol on top of a multimodal LPWAN device, providing an adaptation layer in between the standard IPv6 - UDP - CoAP protocols and the multiple LPWAN technologies. [1] [2]

II. Related Work

Fragmentation in LPWANs has received some attention over the past few years. In [4], the authors study the impact of using smaller fragments, since the probability of collisions are very high for larger packets.

Other than that, several solutions were proposed to benefit from using IPv6 over constrained networks, originating from the idea that even the smallest devices should be connected to the internet.

A. 6LoWPAN and 6lo

In late 2004, the 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) working group was formed to enable IPv6 over IEEE 802.15.4 networks. A few years later, the 6lo working group was added to bring IPv6 connectivity over several other technologies, such as Bluetooth Low Energy (BLE). [3] The IEEE 802.15.4 standard defines the maximum layer 2 Protocol Data Unit (PDU) to be 127 bytes. 25 bytes MAC header, 40 bytes IPv6 header and 8 bytes UDP header would only leave 54 bytes for the payload, without security headers. Therefore the 6LoWPAN working

group defined LOWPAN_HC1 and LOWPAN_HC2 in order to compress the internet protocol header and the transport layer respectively. However, these compression schemes can only be applied to link local addresses. As a consequence, the working group published an amendment, developed under the name LOWPAN_IPHC and LOWPAN_NHC. Communication using UDP and global IPv6 addresses now impacts the link layer frame with a minimum of 10 bytes overhead [11]. In order to support the IPv6 MTU requirement of 1280 bytes, the 6LoWPAN adaptation layer also defines a simple fragmentation mechanism, which does not provide any reliability.

Furthermore, new IETF initiatives arose in order to compress a multiple of protocols using Pages and a Paging Dispatch to switch contexts. [7] In addition to this, the 6LoRH compression technique has been developed to compress IPv6 routing information. [8]

B. CoAP block-wise transfer

CoAP block-wise transfer can be used on top of 6LoWPAN in order to avoid 6LoWPAN fragmentation and to provide reliability. As the transfer of one fragment corresponds to a normal CoAP request, each block must be acknowledged. The failure of a single request will therefore trigger the retransmission of a single block. Each block can only have a size that is a multiple of 16 bytes. The device must ensure to use a block size smaller than the underlying layer 2 PDU [5].

C. SCHC

Very recently, the IETF LPWAN working group started the development of a new standard in order to enable IPv6 connectivity over LPWAN technologies. SCHC was designed for constrained networks with low bandwidth and no layer 2 fragmentation support. The concept of SCHC is further explained in section III-B.

III. Background

A. Multimodal architecture

In order to enable a single device to connect to different LPWANs, the network architecture and infrastructure must be adapted. End-to-end communication should be unified and become independent of the underlying LPWAN technology being used. This has been demonstrated as part of the Virtual Network Operator (VNO) architecture of Hoebeke et al. [9]. Figure 1 shows a high-level overview of the designed architecture. A single stack is used across LPWANs, with the resulting data arriving at technology specific adapters that handle uplink and downlink transmissions. The central broker hides the multimodality towards the Internet and end applications.

B. Static Context Header Compression

Static Context Header Compression is based on the assumption that in LPWANs mostly a static context will be used. Using this assumption, a shared context can be built between the LPWAN devices and the network side. Apart from the header compression scheme, SCHC also offers fragmentation

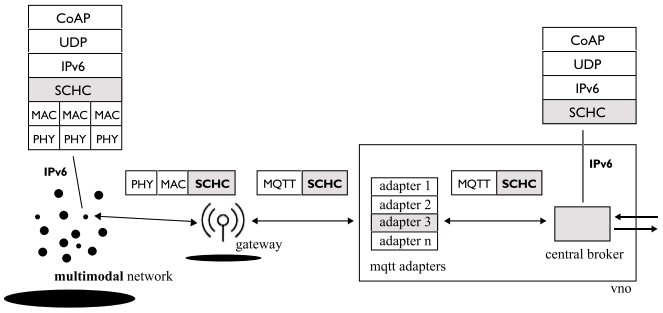

Fig. 1. Multimodal LPWAN architecture

which is used to support the IPv6 MTU requirement of 1280 bytes [12] as well as SCHC packets exceeding the maximum layer two payload size.

1) Compression: In order to perform compression, both sides of the LPWA network share the same context, i.e. a set of rules to compress or decompress the headers. A rule, with corresponding rule ID, describes the compression/decompression behavior for each header field and what possible compression residue to send. This is done by matching the header field values in the order in which they appear, with the expected target value using the Matching Operator (MO). After an exact match, the Compression Decompression Action (CDA) will be applied, which may result in compression residue that could be added to the end of the compressed header. The number of bits sent for each compressed field, is the minimal size necessary to indicate the largest value. If the packet is not L2-word aligned after concatenation of the compressed header bits, padding can be added if not performed by the Fragmenter. The headers are replaced by the rule ID and possible compression residue, which on the other side, after decompression, will result in the exact same packet header that can be forwarded to the global internet.

2) Fragmentation: Once a packet has been compressed or remained uncompressed, the size of the resulting SCHC packet is matched to the underlying layer 2 MTU. A packet exceeding the L2 data unit, may be fragmented by the SCHC Fragmenter. All fragments belonging to the same window carry the same window bit. The reception of a window may or may not be acknowledged, depending on the reliability mode. The Fragment Compressed Number (FCN) is included in all SCHC fragments to indicate the order of the fragments. All FCN bits set to 1, called an all-1 window, indicates the last fragment of a packet. The FCN set to 0, is called an All-0 window and denotes the last fragment of a window. Each window consists of a number of fragments, defined by FCN_MAX_WND_SIZE, which will trigger the receiver to send an acknowledgment. For some technologies (e.g. SigFox) it will be more interesting to use a larger window size, since only 4 downlink messages are possible each day. As the last window will not always be fully filled with fragments, a Message Integrity Check (MIC) is added. A mismatch will trigger the receiving side to acknowledge which packets have been received.

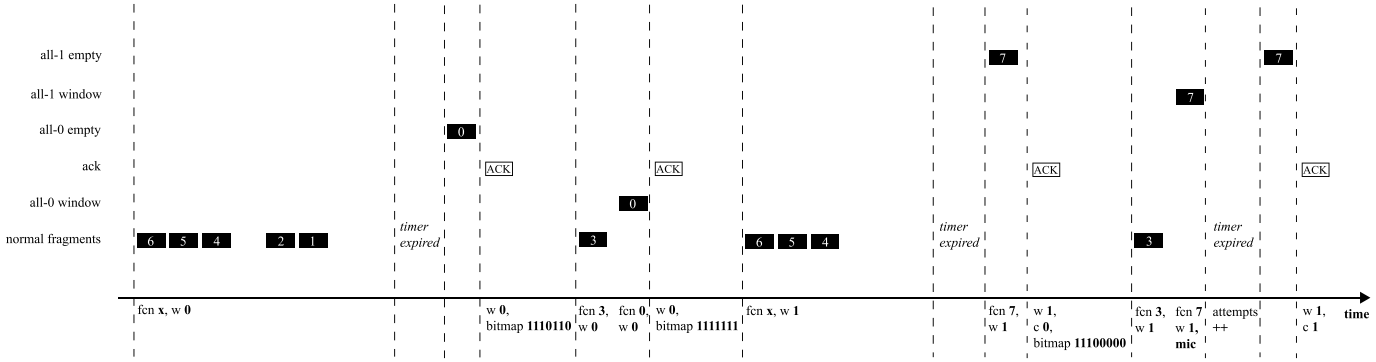

Fig. 2. Illustration of Ack-Always reliability mode.

The SCHC draft defines three reliability modes:

- No-ACK: the receiver will not generate overhead in the form of acknowledgments.
- ACK-Always: each window requires an acknowledgment, regardless of any missing fragments.
- ACK-on-Error: The receiver acknowledges a window only if at least one SCHC Fragment has been lost, except for the last fragment, which must be acknowledged.

Figure 2 illustrates how a packet is sent using reliability mode Ack-Always. The complete SCHC packet is chopped in 12 fragments. A window consists of 7 fragments. While sending the first window, the 4th and last fragment are lost. Since the last fragment is used to identify whether an ACK should be sent, the sender sets a retransmission timer to identify any lost all-x fragments. The expiration of the retransmission timer is also used to indicate whether any acknowledgments were lost. The sender will send an all-0 empty or all-1 empty fragment after an invocation of the retransmission timer. This will trigger the receiving side to resend the acknowledgment. An ACK consists of the rule ID, the Datagram tag (DTAG), the window number and the encoded bitmap. The sender will check the window bit to verify that the ACK belongs to the correct window. The bitmap, depicted at the bottom of the figure and included in the ACK, indicates whether any fragments have been lost or not. After retransmitting the missing fragments, a retransmission timer is set in order to track a missing ACK. The expiration of the timer will trigger the sender to retransmit any lost fragments, as indicated by the received bitmap and considering the number of attempts. This has been illustrated for the second fragment.

IV. Multi protocol analysis

Choosing the correct protocol for a multimodal LPWAN application is not an easy task. Therefore, before having made the decision to continue with SCHC for our multimodal LPWANs, we have evaluated the different protocols described in section II for a device which is able to switch between multiple technologies: SigFox, LoRa and DASH-7. SigFox may be used for cross-country tracking, while LoRa or DASH-7 may be used when a higher throughput is required.

TABLE I
Header overhead and number of exchanged packets for different MTU's and 128 bytes of payload

	6LoWPAN		CoAP block-wise		SCHC	
	(0)	(1)	(0)	(1)	(0)	(1)
MTU 12 bytes						
packets	X	X	X	X	16	18
overhead UL	X	X	X	X	7 B	10 B
overhead DL	X	X	X	X	24 B	25 B
MTU 51 bytes						
packets	5	10	8	10	5	7
overhead UL	36 B	72 B	72 B	90 B	9 B	10 B
overhead DL	23 B	46 B	95 B	115 B	4 B	7 B
MTU 242 bytes						
packets	2	4	4	6	2	4
overhead UL	23 B	47 B	47 B	54 B	1 B	2 B
overhead DL	17 B	34 B	36 B	71 B	1 B	2 B

Figure 3 shows how each protocol will handle header compression, packet fragmentation and packet loss. In this example we consider the best case only for the different protocol stacks. This means that the IPv6 headers may be compressed down to 10 bytes using `LOWPAN_IPHC` and `LOWPAN_NHC` for 6LoWPAN, the CoAP header is limited to 7 bytes, since the CoAP client issues a simple GET request to the shortest URI possible (`/1`), making use of a token with length 2.

We will consider the following scenarios in our analysis:

1) CoAP with 6LoWPAN compression and fragmentation
2) CoAP blockwise transfer and 6LoWPAN compression
3) SCHC compressed CoAP, IPv6 and UDP with SCHC fragmentation

In order to achieve reliable communication in all three cases, a confirmable CoAP message is sent in the first case, since no reliability is guaranteed by the lower layers. CoAP block-wise transfer will confirm each block for the second case and as for the SCHC case, the ACK-Always reliability mode is used.

Table I compares the behavior of the different protocols over different technologies. The first column `(0)` represents the case where no packets were lost, the second `(1)` where 1

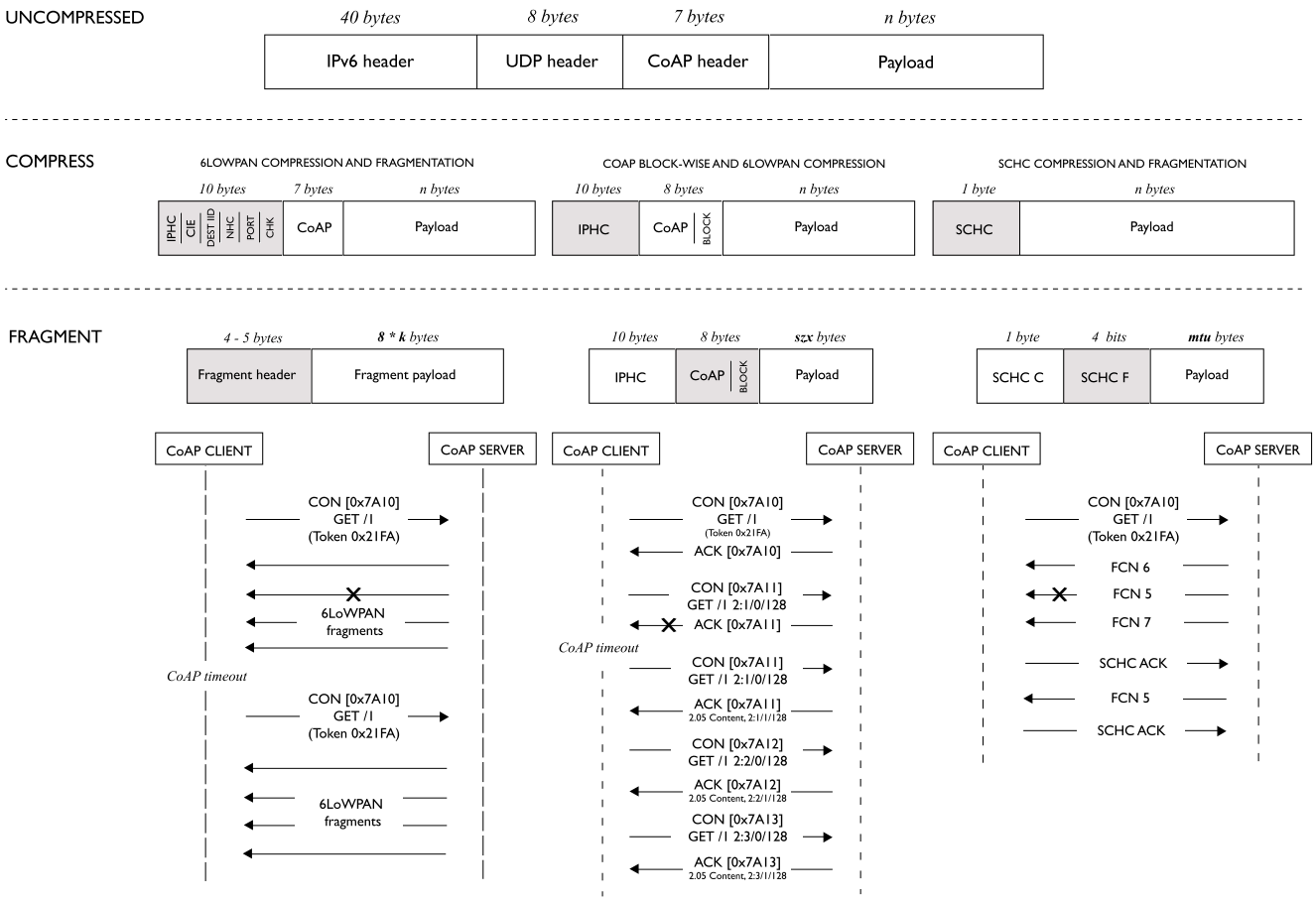

Fig. 3. Multi protocol analysis to enable IPv6 end-to-end connectivity on a LPWAN device. The flow diagram supposes a MTU of 51 bytes and application payload of 128 bytes.

packet was lost.

As the 6LoWPAN fragment offset can only express a multiple of eight bytes [6], the first packet minimally contains 12 bytes, i.e. 4 bytes header and 8 bytes payload. All consecutive packets carry 5 bytes header and therefore contain at least 13 bytes. Since SigFox has a MTU of 12 bytes, 6LoWPAN fragmentation is not suitable for this technology. CoAP block-wise transfer could be used to prevent 6LoWPAN fragmentation. Nevertheless, the smallest block size CoAP supports is 16 bytes and will therefore not be transferable over SigFox. SCHC only requires 12 bits header, which leaves room for another 84 bits of payload and seems the only suitable protocol which supports IPv6 connectivity for all technologies of a multimodal LPWAN device.

An MTU of 242 bytes will leave the packet unfragmented for the SCHC and the 6LoWPAN case. We suppose that no fragmentation means that the packet over SCHC gets acknowledged when using a CoAP CON request.

From this table we can conclude that SCHC imposes a lot less overhead than the other protocols and requires a lot less packet exchanges for a highly fragmented packet where one or more fragments were lost.

In Figure 4, the number of exchanged packets is illustrated for varying payloads. This figure clearly shows that packet retransmissions are much more efficient when using SCHC. If no packets are lost, 6LoWPAN may be more efficient, as SCHC in Ack-Always mode requires a separate ACK for the last fragment, on top of the CoAP ACK in the response.

A. Overhead

Figure 5 shows the header overhead of each protocol stack for zero retransmissions and the retransmission of 1 packet, following the diagram of figure 3. For the first case, the overhead quickly increases, as all fragments require a retransmission and each fragment requires the 10 byte 6LoWPAN header. CoAP block-wise transfer with 6LoWPAN compression will quickly perform better than the first case, as only the missing fragment is sent. However, CoAP block-wise transfer limits the size of the fragments and will therefore require 1 extra fragment in order to send the complete packet. The efficiency of the SCHC fragmentation scheme can clearly be seen from the left graph. A retransmission of 1 packet will trigger a new acknowledgment, which results in a higher response overhead, as seen in the figure on the right.

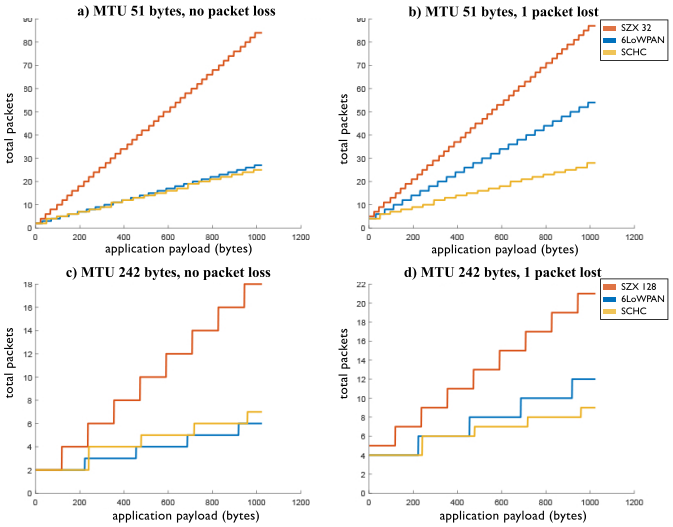

Fig. 4. Number of exchanged packets for different payload sizes and MTUs

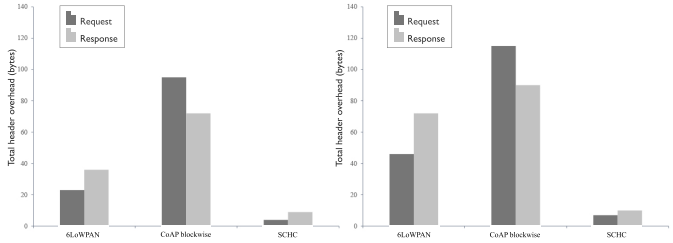

Fig. 5. Total header overhead for a packet of length 128 and MTU 51 for different protocol configurations, left: without retransmissions, right: 1 lost fragment

V. Implementation

Our second part of the evaluation looks at the implementation of the SCHC protocol. The goal of our implementation is to have a generic library, which can be used both on top of a constrained device, as well as on a powerful server device. At the LPWAN side, the OSS-7 operating system is used [13], the application server uses the Click modular router [14] for packet processing and IPv6 forwarding. OSS-7 is designed for constrained devices, whereas Click can forward up to 333,000 64-byte packets/second [14]. The constrained nodes will mostly have one connection and will often not have a Memory Management Unit (MMU) on board. In order not to load the CPU with memory allocation and deallocation, preferably a fixed block of memory is used. Since Click is designed on top of a Linux kernel, memory allocation is not an issue. Apart from that, the constrained node will mostly have one connection, whereas the server will have to support simultaneous packet reassembly from multiple senders. These constraints were taken into account while designing the library, nevertheless making it as generic as possible.

A. Network Memory Buffers

Upon reception of a new fragment, the receiver will allocate a chunk of memory for that fragment. The constrained node will take a chunk of a fixed memory block, the server will allocate a chunk of memory from the heap. As none of the fragments contains the total packet length, a Network Memory Buffer (mbuf) chain is utilized, derived from the FreeBSD operating system [15], which allows trimming network headers and concatenating fragments with very little overhead. The fragment is added to the end of the linked list. Each fragment selects a slot from the mbuf pool, sets the data pointer and attaches this mbuf as the last one in the chain for this connection. In order to return a byte aligned mbuf chain to the application, with the bytes concatenated as before transmission, thus without the fragmentation headers and using the same memory buffer, the headers of each mbuf are removed and the payload of the next mbuf in the chain is shifted in.

B. Connection State

Each SCHC connection requires a certain amount of state information. The state information is kept to a minimum, in order to reduce the amount of RAM required for each connection. Two connections are required: a TX and an RX connection. Both connections require 72 bytes of data. Connection state information is changed by the state machine, but may also be changed by the application. This can be useful when the underlying communication technology changes, which may result in a larger or smaller MTU. The library must acquire this information in order to send fragments of a correct size.

C. Timers and Retransmissions

The main loop is driven by the duty cycle timer. Once the first fragment is sent, the duty cycle timer is set, depending on the local regulations. Since the connection struct keeps track of the duty cycle time, this can be changed by the application at run-time, allowing the device to respond to dynamic requirements of the application. After an invocation of the timer, the timer calls the fragmentation state machine, in order to continue the transmission of the fragments. The implementation does not keep track of packet contents after they have been sent by the underlying LPWAN technology. Once an acknowledgment arrives, the bitmap included in the acknowledgment is used to re-generate the corresponding fragment. From an application's point of view, retransmitting data is performed the same way as the data was sent originally. Hence, the same code can be used for sending and retransmitting data.

VI. Evaluation

A. Connections

In order to test the implementation, we created a python script, which generates fragments for a number of devices. The fragments are sent to the MQTT broker from figure 1, which will forward them to the server. The experimental setup consists of a computer running Ubuntu 16.04 with 16 GB of RAM and a 2.8 GHz Intel Core i5-7440HQ CPU. We successfully evaluated the server to keep track of 100.000

TABLE II
CODE SPACE REQUIRED FOR EACH SCHC COMPONENT

	Compressor	Fragmenter	Rules	Total
RAM	1061 B	528 B	0 B	1589 B
ROM	6082 B	6692 B	3256 B	16030 B

device connections, including fragments and timer callback information.

B. Memory Footprint

Table II shows the memory required to implement the full standard including all reliability modes and the use of variable window sizes. Some optimization can be done, mostly for the compressor. The RAM used for the fragmenter depends upon the configuration and does not include cumulative RAM usage. The current calculations illustrate the required memory for a constrained device. Each RX connection will add 72 bytes of RAM. Each mbuf requires 16 bytes of memory. The current `MBUF_POOL` consists of 8 mbuf's, resulting in 128 bytes of RAM. Currently, the rules are implemented in human-readable fashion, which requires 180 bytes for each IPv6 rule, 48 bytes for a UDP rule and 400 bytes for each CoAP rule. An obvious scenario would require 2 UDP rules and 2 IPv6 rules: one rule to handle the default configuration and one to handle less frequent traffic. The number of CoAP rules are very application dependent. In order to achieve a higher compression ratio, more rules are required. For this example, 7 CoAP rules are considered. An extra 3.256 bytes are thus required for this scenario. Ludovici et. al report that their 6LoWPAN implementation requires 22.584 bytes of ROM and 3.421 bytes of RAM. [10] The compressor currently requires recursive allocated RAM which is not considered best practice, which is not included in the calculations and sometimes causes memory overflows.

VII. CONCLUSION

We presented SCHC compression and its accompanying fragmentation mechanism and compared its overhead and number of packet exchanges to CoAP block-wise transfer and 6LoWPAN compression and fragmentation. Our analysis shows that SCHC is the only protocol which is able to support end-to-end IPv6 connectivity for a multimodal device, supporting SigFox, LoRa and DASH-7 and therefore requires a new compression technique as proposed by the LPWAN WG. We showed that SCHC is a more suitable compression and fragmentation mechanism for LPWAN devices in terms of header overhead, reliability and total number of packets exchanged than 6LoWPAN. Our analysis can even be used to justify SCHC's right of existence.

Apart from that, we evaluated our SCHC implementation, which is designed in a generic fashion with a relative small footprint for an embedded device and is suited to handle a growing number of devices at the server side, nevertheless able to suit the low-memory needs of a constrained device. Compared to a 6LoWPAN implementation, SCHC also requires a lot less memory.

VIII. FUTURE WORK

The human-readable fashion in which the rules are implemented result in a lot of memory overhead and can be optimized if an efficient encoding mechanism is used.

In order to intelligently switch between multiple technologies, some kind of Quality of Service (QoS) is required. While switching technologies, a change of the underlying MTU has to be taken into account, which could affect an ongoing transmission.

IX. ACKNOWLEDGMENT

Part of this research was funded by the Flemish FWO SBO S004017N IDEAL-IoT (Intelligent DEnse And Longe range IoT networks) project, and by the ICON project MuSCLe-IoT. MuSCLe-IoT is a project realized in collaboration with imec, with project support from VLAIO (Flanders Innovation and Entrepreneurship). Project partners are imec, Flash Private Mobile Networks, Engie M2M, Sensolus and Aertssen.

REFERENCES

[1] A. Minaburo, L. Toutain, et al. 2018. LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP. Internet-Draft draft-ietf-lpwan-ipv6-static-contexthc-16. IETF Work in Progress.

[2] A. Minaburo and L. Toutain. 2018. LPWAN Static Context Header Compression (SCHC) for CoAP. Internet-Draft draftietf-lpwan-coap-static-contexthc-04. IETF Work in Progress.

[3] C. Gomez, J. Paradells, C. Bormann and J. Crowcroft, "From 6LoWPAN to 6Lo: Expanding the Universe of IPv6-Supported Technologies for the Internet of Things" in IEEE Communications Magazine, vol. 55, no. 12, pp. 148-155, December 2017.

[4] I. Suciu, X. Vilajosana, F. Adelantado. 2017. An Analysis of Packet Fragmentation in LPWAN.

[5] Block-Wise Transfers in the Constrained Application Protocol (CoAP). C. Bormann, Z. Shelby. August 2016.

[6] G. Montenegro, J. Hui, D. Culler and N. Kushalnagar, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", September 2007.

[7] IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch. P. Thubert, R. Cragie. November 2016.

[8] IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header. P. Thubert, C. Bormann, L. Toutain, R. Craggie. April 2017.

[9] J. Hoebeke, J. Haxhibeqiri, B. Moons, M. Van Eeghem, J. Rossey, A. Karaagac and J. Famaey, "A Cloud-based Virtual Network Operator for Managing Multimodal LPWA Networks and Devices", July 2018.5

[10] A. Ludovici, A. Calveras, M. Catalan, C. Gmez, and J. Paradells, Implementation and Evaluation of the Enhanced Header Compression (IPHC) for 6LoWPAN, in The Internet of the Future, vol. 5733, M. Oliver and S. Sallent, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 168177.

[11] I. Ishaq, D. Carels, G. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman and P. Demeester. (2013). "IETF Standardization in the Field of the Internet of Things (IoT): A survey." Journal of Sensor and Actuator Networks. 2. 235-287. 10.3390/jsan2020235.

[12] S. Deering and B. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", July 2017

[13] (Last visited 18/8/2018), "The OSS-7 Website". [Online]. Available: "http://mosaic-lopow.github.io/dash7-ap-open-source-stack/"

[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, The click modular router, ACM Trans. Comput. Syst., vol. 18, no. 3, pp. 263297, August 2000

[15] B. Milekic, "Network Buffer Allocation in the FreeBSD Operating System", May 2004