Lithe: Lightweight Secure CoAP for the Internet of Things

Shahid Raza, Hossein Shafagh, Kasun Hewage, René Hummen, and Thiemo Voigt

***Abstract*— The Internet of Things (IoT) enables a wide range of application scenarios with potentially critical actuating and sensing tasks, e.g., in the e-health domain. For communication at the application layer, resource-constrained devices are expected to employ the constrained application protocol (CoAP) that is currently being standardized at the Internet Engineering Task Force. To protect the transmission of sensitive information, secure CoAP mandates the use of datagram transport layer security (DTLS) as the underlying security protocol for authenticated and confidential communication. DTLS, however, was originally designed for comparably powerful devices that are interconnected via reliable, high-bandwidth links. In this paper, we present Lithe—an integration of DTLS and CoAP for the IoT. With Lithe, we additionally propose a novel DTLS header compression scheme that aims to significantly reduce the energy consumption by leveraging the 6LoWPAN standard. Most importantly, our proposed DTLS header compression scheme does not compromise the end-to-end security properties provided by DTLS. Simultaneously, it considerably reduces the number of transmitted bytes while maintaining DTLS standard compliance. We evaluate our approach based on a DTLS implementation for the Contiki operating system. Our evaluation results show significant gains in terms of packet size, energy consumption, processing time, and network-wide response times when compressed DTLS is enabled.**

***Index Terms*— CoAP, DTLS, CoAPs, 6LoWPAN, security, IoT.**

I. Introduction

IPV6 over Low power Wireless Personal Area Network (6LoWPAN) [1] enables the use of IP in low-power and lossy wireless networks such as Wireless Sensor Networks (WSNs). Such IP-connected smart devices (*Things*) are becoming part of the Internet hence forming the Internet of Things (IoT) or strictly speaking the IP-connected IoT. TCP performance is known to be inefficient in wireless networks, due to its congestion control algorithm, and the situation is exacerbated with the low-power radios and lossy links found in sensor networks. Therefore, the connectionless UDP is mostly used in the IoT. Further, HTTP, which is primarily designed to run over TCP, is inefficient in lossy and constrained environments. The IETF is working on the connection-less lightweight Constrained Application Protocol (CoAP) [2], a new proposed standard for the IoT. CoAP is designed to meet specific requirements such as simplicity, low overhead, and multicast support in resource-constrained environments. Security is particularly important for the *Things* as they are connected to the untrusted Internet. For instance, medical monitoring denotes a typical security-sensitive application scenario. Here, a smart device, such as an insulin,[1] may be attached to the patient's body and periodically report the condition of the patient to a back-end service in the Internet. In emergency cases, a physician may additionally be able to trigger instant injection of medication into the patient's body.

CoAP proposes to use Datagram Transport Layer Security (DTLS) [2] as the security protocol for automatic key management and for data encryption and integrity protection, as well as for authentication. CoAP with DTLS support is termed secure CoAP (CoAPs). DTLS is a chatty protocol and requires numerous message exchanges to establish a secure session. While DTLS supports a wide range of cryptographic primitives for peer authentication and payload protection, it was originally designed for network scenarios where message length was not a critical design criterion. Therefore, it is inefficient to use the DTLS protocol, as it is, for constrained IoT devices. To cope with constrained resources and the size limitations of IEEE 802.15.4-based networks,[2] 6LoWPAN header compression mechanisms are defined. The 6LoWPAN standard already defines the header compression format for the IP header, IP extension headers, and the UDP header. We believe it is particularly beneficial to apply the 6LoWPAN header compression mechanisms to compress other protocols having well-defined header fields, such as DTLS.

In this paper we provide a lightweight CoAPs by compressing the underneath DTLS protocol [3] with 6LoWPAN header compression mechanisms. We name our lightweight 6LoWPAN compressed CoAPs Lithe. The purpose of DTLS header compression is twofold. First, achieving energy efficiency by reducing the message size, since communication requires

Manuscript received May 24, 2013; revised July 3, 2013 and July 25, 2013; accepted July 31, 2013. Date of publication August 7, 2013; date of current version August 28, 2013. This work was supported by the SICS Center for Networked Systems (CNS), SSF through the Promos project, and CALIPSO, Connect All IP-Based Smart Objects, funded by the European Commission under FP7 under Contract FP7-ICT-2011.1.3-288879. The associate editor coordinating the review of this paper and approving it for publication was Dr. Chonggang Wang.

S. Raza is with SICS Swedish ICT, Kista SE-164 29, Sweden (e-mail: shahid@sics.se).

H. Shafagh is with SICS Swedish ICT, Kista SE-164 29, Sweden, and also with Communication and Distributed Systems, RWTH Aachen University, Aachen 52062, Germany (e-mail: hossein@sics.se).

K. Hewage is with the Department of Information Technology, Uppsala University, Uppsala 751 05, Sweden (e-mail: kasun.hewage@it.uu.se).

R. Hummen is with Communication and Distributed Systems, RWTH Aachen University, Aachen 52062, Germany (e-mail: hummen@comsys.rwth-aachen.de).

T. Voigt is with SICS Swedish ICT, Kista SE-164 29, Sweden, and also with the Department of Information Technology, Uppsala University, Uppsala 751 05, Sweden (e-mail: thiemo@sics.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSEN.2013.2277656

[1] Medtronic probes insulin pump risks, Reuters, October 2011.

[2] The Maximum Transmission Unit (MTU) size of the IEEE 802.15.4 protocol is 127 bytes.

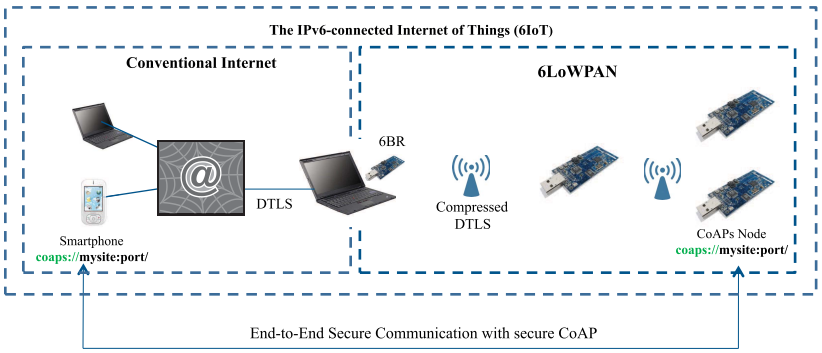

Fig. 1. An IoT setup that uses CoAPs to secure communication between sensor nodes in 6LoWPANs and hosts in the Internet.

more energy than computation. Second, avoiding 6LoWPAN fragmentation that is applied when the size of a datagram is larger than the link layer MTU. Avoiding fragmentation, whenever possible, is also important from the security point of view as the 6LoWPAN protocol is vulnerable to fragmentation attacks [4]. Our compressed DTLS maintains true End-to-End (E2E) security between Lithe enabled hosts in 6LoWPAN networks and typical Internet hosts that use uncompressed CoAPs. Figure 1 shows a typical IoT setup, where a 6LoWPAN network consisting of CoAPs enabled nodes is connected through a 6LoWPAN Border Router (6BR) with the Internet.

To the best of our knowledge we are the first to propose 6LoWPAN compressed DTLS and enable lightweight CoAPs support for the IoT. We implement our DTLS header compression mechanisms in the Contiki OS [5]. The main contributions of this paper are:

- We provide novel and standard compliant DTLS compression mechanisms that aim to increase the applicability of DTLS and, thus, CoAPs for constrained devices.
- We implement the compressed DTLS in an OS for the IoT and evaluate it on real hardware; the results quantitatively show that Lithe is in many aspects more efficient compared to uncompressed CoAP/DTLS.

The rest of the paper is organized as follows. We first summarize related work in Section II. We give a brief overview of the technologies used in this paper in Section III. In Section IV, we introduce our DTLS header compression mechanisms. Our implementation is outlined in Section V. In Section VI, we describe our network setup and discuss the evaluation results. Finally, Section VII concludes this paper.

II. Related Work

Providing E2E security is a widely explored area in conventional Internet communication. However, there has been comparatively less research conducted in E2E security considering 6LoWPANs. The resource constraints of the devices and the lossy nature of wireless links are among the major reasons that hinder applying general E2E security mechanisms to 6LoWPANs. Recently, the community has presented works on analyzing security challenges in the IP-based IoT [6] and solutions that improve or modify standard IP security protocols for the requirements of resource-constrained devices. In our discussion of related work, we focus on approaches that aim to enable E2E security solutions in the IoT.

In our previous work [7], we propose a header compression method to use IPsec to secure the communication between nodes in 6LoWPAN networks and hosts in the Internet. We define Next Header Compression (NHC) encodings to compress the Authentication Header (AH) and Encapsulating Security Payload (ESP) extension headers. Jorge *et al*. [8] extend our solution and include IPsec in tunnel mode. They implement and evaluate their proposal in TinyOS. IPsec security services are shared among all applications running on a particular machine. Even though our 6LoWPAN compressed IPsec can be used to provide lightweight E2E security at the network layer, it is not primarily designed for web protocols such as HTTP or CoAP. For web protocols TLS or DTLS are common security solutions. TLS works over TCP, whereas in 6LoWPAN networks UDP is preferred.

Brachmann *et al*. [9] propose TLS-DTLS mapping to secure the IoT. However, this requires the presence of a trusted 6BR and E2E security breaks at the 6BR. Kothmayr *et al*. [10] investigate the use of DTLS in 6LoWPANs with a Trusted Platform Module (TPM) to get hardware support for the RSA algorithm. However, they have used DTLS as it is without using any compression method which would shorten the lifetime of the entire network due to the redundant bits in DTLS messages. Granjal *et al*. [11] evaluate the use of DTLS as it is with CoAP for secure communication. They note that payload space scarcity would be problematic with applications that require larger payloads. As an alternative, they suggest to employ security at other layers such as compressed form of IPsec. In a recent work, Keoh *et al*. [12] have discussed the implications of securing the IP-connected IoT with DTLS and propose an architecture for secure network access and management of unicast and multicast keys with extended DTLS.

The above solutions either assess the use of TLS or DTLS in the IoT or present architectures that break E2E security. In this paper, we reduce the overhead of DTLS for the IoT by employing 6LoWPAN header compression mechanisms. In another work [13], we propose design ideas to reduce the energy consumption of the two-way certificate-based DTLS handshake. We suggest (i) pre-validation of certificates at the trusted 6BR, (ii) session resumption to avoid full re-handshake, and (iii) handshake delegation to the owner of the resource-constrained device. That work in making certificate-based authentication viable for the IoT is complementary to

this one. We plan to combine DTLS header compression with those ideas to make the mutual certificate-based handshake more efficient. Recently, Generic Header Compression (GHC) [14], analogous to NHC, is also defined to allow upper layer (UDP payload and above) header compression. 6LoWPAN-GHC is a generic compression scheme for all headers and header-like structures but is a slightly less efficient approach [14]. It is an alternative to our solution and we plan to compare our 6LoWPAN-NHC with the 6LoWPAN-GHC for the DTLS headers as future work.

III. Background

Due to the heterogeneity in the IoT, it is challenging to connect resource-constrained devices in a secure and reliable way. Currently, different protocols such as CoAP [2], 6LoWPAN [15], the IPv6 Routing Protocol (RPL) [16] for Low-power and Lossy Networks (LLNs) are being standardized by the Internet Engineering Task Force (IETF) to enable the IoT. The focus of this paper is to enable secure yet efficient communication among IoT devices that utilize the CoAP protocol. In this section, we highlight the technologies involved in the development of the lightweight CoAPs, the HTTPs variant for the IoT.

A. CoAP and DTLS

CoAP is a web protocol that runs over the unreliable UDP protocol and is designed primarily for the IoT. CoAP is a variant of the most used synchronous web protocol, HTTP, and is tailored for constrained devices and machine-to-machine communication. However, while CoAP provides a REST interface similar to HTTP, it focuses on being more lightweight and cost-effective than its variant for today's Internet. To protect CoAP transmissions, Datagram TLS (DTLS) has been proposed as the primary security protocol [2]. Analogous to TLS-protected HTTP (HTTPs), the DTLS-secured CoAP protocol is termed CoAPs. A web resource on an IoT device can then be accessed securely via CoAPs protocol as:

coaps://myIPv6Address:port/MyResource

As a basis for the discussion of our proposed DTLS compression mechanisms, we give a brief overview of the DTLS protocol.

DTLS guarantees E2E security of different applications on a single machine by operating between the transport and application layers. DTLS consists of two layers: the lower layer contains the Record protocol and the upper layer contains either of the three protocols namely Handshake, Alert, and ChangeCipherSpec, or application data. The ChangeCipherSpec is used during the handshake process to merely indicate that the Record protocol should protect the subsequent messages with the newly negotiated cipher suite and security keys. DTLS uses the Alert protocol to communicate the error messages between the DTLS peers. Figure 2 shows the structure of a DTLS message in an IP/UDP datagram.

The Record protocol [3] is a carrier for the upper layer protocols. The Record header contains among others content type and fragment fields. Based on the value in the content type, the fragment field contains either the Handshake protocol,

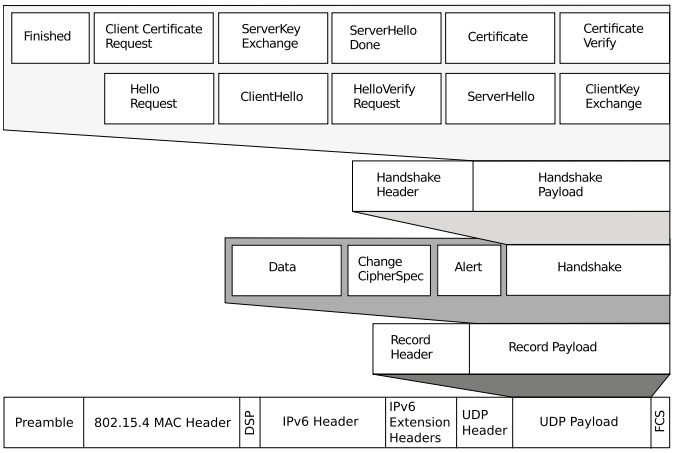

Fig. 2. Layout of a packet secured with DTLS.

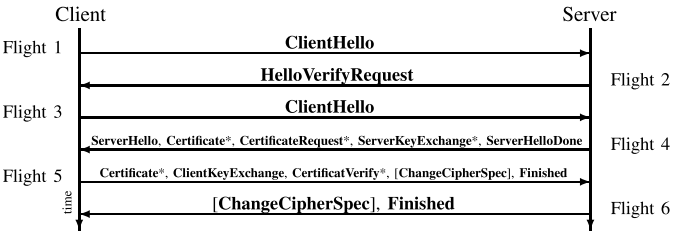

Fig. 3. Full DTLS handshake protocol. Messages marked with a * are optional.

Alert protocol, ChangeCipherSpec protocol, or application data. The Record header is primarily responsible to cryptographically protect the upper layer protocols or application data once the handshake process is completed. The Record protocol's protection includes confidentiality, integrity protection and authenticity.

The DTLS Record is a rather simple protocol whereas the Handshake protocol is a complex chatty process and contains numerous message exchanges in an asynchronous fashion. Figure 3 shows a full handshake process. The handshake messages, usually organized in flights, are used to negotiate security keys, cipher suites and compression methods. The scope of this paper is limited to the header compression only and not the cryptographic processing of Record and Handshake protocols. For details of the individual handshake messages we refer to TLS [17] and DTLS [3].

B. 6LoWPAN

The 6LoWPAN standard [1] defines header compression and fragmentation mechanisms of IPv6 datagrams within IPv6-connected WSNs, also called 6LoWPAN networks. The compression mechanism consists of IP Header Compression (IPHC) and Next Header Compression (NHC). The IPHC encodings can compress the IPv6 header length to 2 bytes for a single hop network and 7 bytes in a multi-hop case (1-byte IPHC, 1-byte dispatch, 1-byte Hop Limit, 2-byte Source Address, and 2-byte Destination Address). Among other encoding bits in the IPHC is the NH bit that, when set, indicates the next header is compressed using NHC. The NHC

is used to encode the IPv6 extension headers and UDP header. The size of NHC encodings is a multiple of octets (mostly one octet) which contain variable length ID bits and the encoding bits for a specific header. There are protocols that are part of UDP payload and have header-like structures similar to IP and UDP, such as DTLS, IKE [18]. It is therefore worth extending the 6LoWPAN header compression mechanisms to compress these protocol headers. The 6LoWPAN standard-defined NHC encoding can be used to compress headers up to UDP, but not the upper layers. A new NHC is needed because there is no NH bit in the NHC for UDP which indicates that the UDP payload is also compressed. In Section IV, we provide 6LoWPAN-DTLS integration and 6LoWPAN NHCs to compress DTLS.

As depicted in Figure 1, the header compression is applied within the 6LoWPAN network only, i.e., between constrained nodes and the 6LoWPAN border Router (6BR). A 6BR is used between 6LoWPAN networks and the Internet to compress/decompress or/and fragment/reassemble messages before forwarding between the two realms. In this IoT setup, the CoAPs enabled devices can securely communicate with internet hosts, such as standard computers, smartphones, etc., which support the CoAPs protocol. In order to adapt chatty security protocols, such as DTLS, for the resource-constrained IoT devices, it is beneficial to apply 6LoWPAN header compression mechanisms to these protocols as well. In Section IV we propose 6LoWPAN header compression mechanisms for DTLS. It is very important to design these header compression mechanisms in a way that complies with the DTLS standard, to be interoperable with existing and new DTLS enabled hosts on the conventional Internet.

IV. DTLS Compression

DTLS header compression, like IPHC, is applied only within 6LoWPAN networks, i.e., between sensor nodes and the 6BR. This is because the DTLS headers are part of the UDP payload and all information required for routing is already extracted at the IP layer. In this section, in addition to describing 6LoWPAN header compression for DTLS, we detail how our compressed DTLS can be linked to 6LoWPAN in a standard compliant way.

A. DTLS-6LoWPAN Integration

In order to apply 6LoWPAN header compression mechanisms to compress headers in the UDP payload, we either require a modification in the current NHC encodings for UDP in the 6LoWPAN standard, or need to define a new NHC for UDP with different ID bits. The first solution requires modification in the current standard and hence is not a favorable solution. The second solution, that we use in this paper, is an extension to the 6LoWPAN standard; a similar approach is adapted to distinguish NHC from GHC [14]. The ID bits 11110 in the NHC for UDP, as defined in the 6LoWPAN standard, indicate that the UDP payload is not compressed. We define ID bits 11011 to indicate that the UDP payload is compressed with 6LoWPAN-NHC. The ID bits 11011 are currently unassigned in the 6LoWPAN standard [1]. Figure 4 shows our proposed NHC for UDP that allows compression of UDP payload; in our case, the UDP payload contains the 6LoWPAN-NHC compressed DTLS headers.

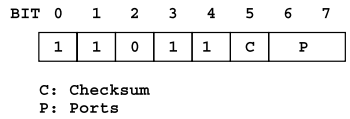

Fig. 4. Our proposed 6LoWPAN-NHC for UDP, where ID bits 11011 indicate that the UDP payload is compressed.

In the following section we define 6LoWPAN-NHC for the DTLS Record header, Handshake header, and handshake messages where applicable.

B. 6LoWPAN-NHC for the Record and Handshake Headers

The Record protocol adds 13 bytes long header fields to each packet that is sent throughout the lifetime of a device that uses DTLS. The handshake protocol, on the other hand, adds 12 bytes of header to handshake messages. We propose 6LoWPAN-NHC for compressing the Record and Handshake headers, and reduce the header length to 5 and 3 bytes, respectively. In case of Handshake, only during the first handshake process the handshake header and handshake messages are compressed. This is because the successive re-handshake messages are encrypted using the negotiated cipher suite, and it is not possible to inspect the payload of the DTLS record for compression at the 6LoWPAN layer. In all cases the Record header remains un-encrypted. Thus it is always compressed by using the mechanism explained in this section.

In order to provide header compression for the Record and Handshake header, we consider two cases. In the first case, where the Record header fragment field (see Section III) contains a handshake message, we compress both the Record header and the Handshake header using a single encoding byte and we define 6LoWPAN-NHC for Record+Handshake (6LoWPAN-NHC-RHS). In the second case, we define 6LoWPAN-NHC for the Record header (6LoWPAN-NHC-R) where the fragment field in the Record header is application data and not a Handshake message as in the first case. The 6LoWPAN-NHC-R is applied after the DTLS handshake has been performed successfully, and the subsequent messages are encrypted and integrity protected. Figure 5a shows 6LoWPAN-NHC encodings for the Record+Handshake header and for the Record header. The encoded bits have the following functions: The first four bits represent the ID field that is used to distinguish 6LoWPAN-NHC-RHS from other encodings, and to comply with 6LoWPAN-NHC encoding scheme. In case of 6LoWPAN-NHC-RHS we set the ID bits to 1000, and in case of 6LoWPAN-NHC-R we set the ID bits to 1001.

Version (V): If 0, the version is the DTLS latest version which is 1.2, and the field is omitted. If 1, the version field is carried inline.

Epoch (EC): If 0, an 8 bit epoch is used and the left most 8 bits are omitted. If 1, all 16 bits of the epoch are carried inline. In most cases the actual epoch is either 0 or 1. Therefore, an 8 bit epoch is used most of the time, allowing a higher space.[3]

[3]The space saving is $1 - (compressed\_size/uncompressed\_size)$.

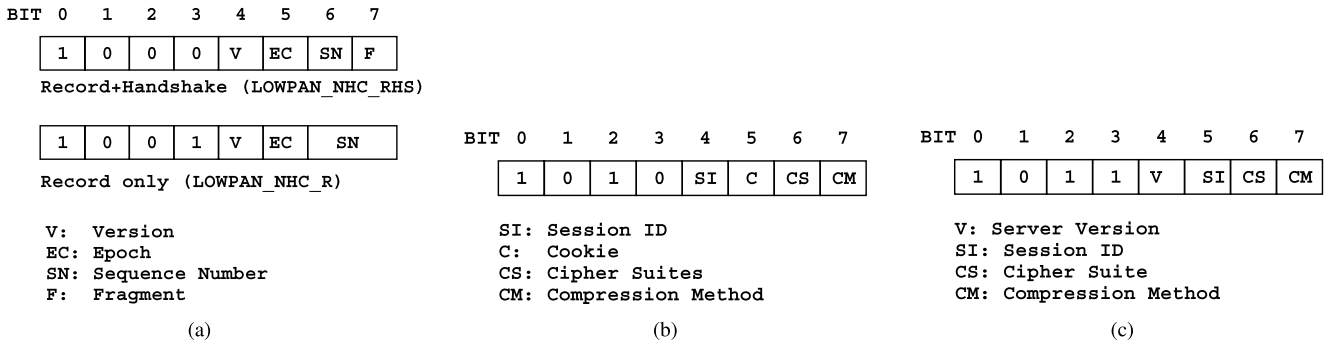

Fig. 5. Our proposed 6LoWPAN-NHC encodings for different DTLS headers. (a) Record and the Handshake header. (b) Client Hello message. (c) ServerHello message.

Sequence Number (SN): The sequence number consists of 48 bits, of which some are leading zeros. If SN is set to 0, a 16 bit sequence number is used and the left most 32 bits are omitted. If 1, all 48 bits of the sequence number are carried inline. In case of 6LoWPAN-NHC-R, as shown in Figure 5a, we use two bits for SN and can more efficiently compress the sequence_number field. Here if SN is set to 00, a 16 bit sequence number is used and the left most 32 bits are omitted. If 01, a 32 bit sequence number is used and the left most 16 bits are omitted. If 10, a 24 bit sequence number is used and the left most 24 bits are omitted. If 11, all 48 bits of the sequence number are carried inline.

Fragment (F): If 0, the handshake message is not fragmented and the fields fragment_offset and fragment_length are omitted. This is the common case, which occurs when the handshake message is not larger than the maximum record size. If 1, the fields fragment_offset and fragment_length are carried inline.

In the Record header, content_type field is always carried inline. Furthermore, message_type and message_seq fields of the Handshake header are always carried inline. The length field in the Handshake headers is always omitted as it can be deduced from the lower layers: either from the 6LoWPAN header or the IEEE 802.15.4 header. We have to uncompress layer-wise from lower to higher layers until the UDP header is uncompressed. Then the length of the UDP payload is known and the DTLS payload length can be calculated. The length field in the Record header may also be omitted as we expect only one DTLS record per UDP packet in constrained environments. While a source device inside a 6LoWPAN sends one DTLS record per UDP packet, a typical destination device on the conventional Internet side may send multiple DTLS records in a single UDP packet. However, as the 6BR performs the compression/decompression of incoming packets, there is the possibility to enforce one DTLS record per UDP packet before routing these packets in 6LoWPAN networks.

C. 6LoWPAN-NHC for ClientHello

We propose 6LoWPAN-NHC for the ClientHello message (6LoWPAN-NHC-C). During the handshake process the ClientHello message is sent twice, the first time without cookie and the second time with the server's cookie. Figure 5b shows 6LoWPAN-NHC encoding for the ClientHello message. The function of each compressed header field is described below:

The first four bits in the 6LoWPAN-NHC-CH represent the ID field which are set to 1010.

Session ID (SI): If 0, the session_id is not available and this field and 8 bits of the prefixed length field are omitted. In the (D)TLS protocol, session_id is empty if no session is available, or if the client wishes to generate new security parameters. The ClientHello message uses session_id only if the DTLS client wants to resume the old session. The actual session_id field in the ClientHello contains 0 to 32 bytes. However, it is always prefixed with an 8 bit field that contains the size of the session_id. If SI is set 1, the session_id field is carried inline.

Cookie (C): If 0, the cookie field is not available and this field and its prefixed 8 bits length field are omitted. The actual cookie field in the ClientHello contains 0 to 255 bytes.[4] However, it always has an 8 bits length field that contains the size of the cookie. If C is set 1, the cookie field is carried inline.

Cipher Suites (CS): If 0, the default (mandatory) cipher suite for CoAP that supports automatic key management is used and this field and the prefixed 16 bits length field are omitted. In the current CoAP draft [2] TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 is a mandatory cipher suite. The actual cipher_suites field contains 2 to $2^{16}-1$ bytes and is always prefixed with a 16 bits field that contains the size of the cipher_suites. If CS is set 1, the cipher_suites field is carried inline.

Compression Methods (CM): If 0, the default compression method, i.e., COMPRESSION_NULL is used and this field and the prefixed 8 bits length field are omitted. The actual compression_methods field contains 1 to $2^8 - 1$ bytes. It is always prefixed with an 8 bits field that contains the size of the compression_methods. If CM is set 1, the compression_methods field is carried inline.

The random field in the ClientHello is always carried inline whereas the version field is always omitted. The version contains the same value as in the DTLS Record header. In case of TLS/SSL the version field was defined to let a TLS client specify an older version to be compatible with an SSL client, which is rarely used in practice. All current versions of web

[4]DTLS 1.2 specification increases the cookie size limit to 255 bytes; however, our implementation and evaluation use a cookie size of 16 bytes.

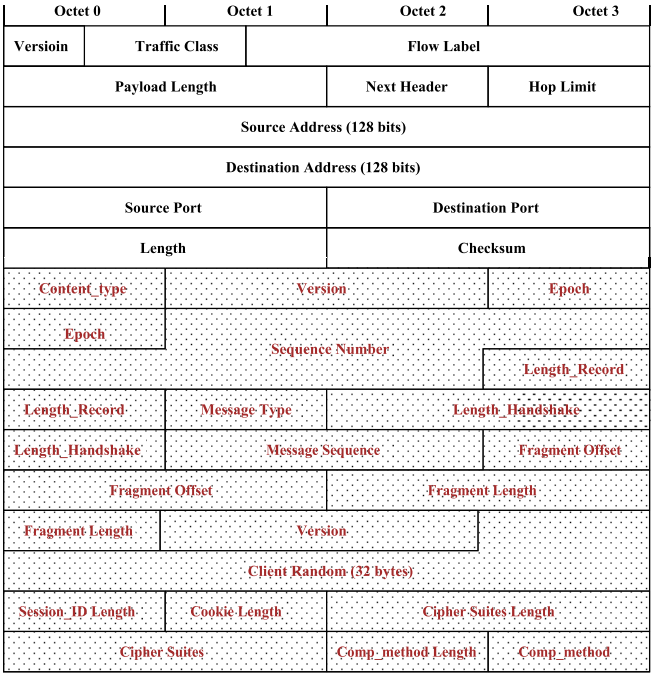

Fig. 6. An *uncompressed* full IP/UDP datagram containing a DTLS ClientHello Message.

browsers use the same TLS version in Record and ClientHello. DTLS 1.2 (adapted from TLS 1.2) [17] mentions that the client sends its latest supported version in the ClientHello message. All DTLS versions (1.0 and 1.2) have compatible ClientHello messages. If the server does not support this version, then the ServerHello message contains its supported version. If the client is not capable of handling the server's version, it terminates the connection with a protocol version alert.

Using 6LoWPAN-NHC-CH, usually only the random field in the ClientHello message is transmitted and all the other fields are omitted. with cookie may also contain the compressible cookie field. Figure 6 shows an uncompressed IP/UDP datagram that contains a ClientHello. A 6LoWPAN compressed IP/UDP datagram, with our proposed compressed DTLS, containing the ClientHello message is depicted in Figure 7. After applying IPHC and 6LoWPAN-NHC header compression, the datagram size is significantly reduced.

D. 6LoWPAN-NHC for ServerHello

We propose 6LoWPAN-NHC for the ServerHello message (6LoWPAN-NHC-SH). ServerHello is very similar to ClientHello except that the length of the cipher_suites and compression_methods fields are fixed to 16 and 8 bits, respectively. Figure 5c shows the 6LoWPAN-NHC encoding for the ServerHello message. The function of each compressed header field is described below: The first four bits in the 6LoWPAN-NHC-SH represent the ID field set to 1011. *Version (V)*: In order to avoid version negotiation in the initial handshake, the DTLS 1.2 standard suggests that the server implementation should use DTLS version 1.0. If V is set to 0, the version is DTLS 1.0 and the version field is omitted. However the DTLS

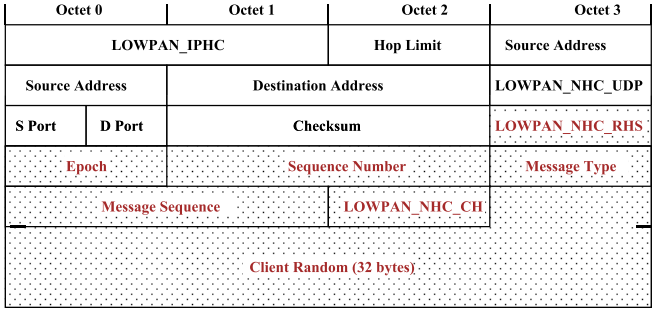

Fig. 7. A 6LoWPAN *compressed* full IP/UDP datagram containing a DTLS ClientHello Message.

1.2 clients must not assume that the server does not support higher versions or it will eventually negotiate DTLS 1.0 rather than DTLS 1.2 [3]. If V is set to 1, the version field is carried inline.

Session ID (SI), *Cipher Suite (CS)*, and *Compression Method (CM)* are encoded in a similar fashion as discussed in Section IV-C. In order to not compromise security the random field in the ServerHello is always carried inline.

E. 6LoWPAN-NHC for other Handshake Messages

The remaining mandatory handshake messages ServerHelloDone, ClientKeyExchange, and Finish have no fields that could be compressed, hence all fields are carried inline. The optional handshake messages Certificate that contains the chain of certificates and CertificateVerify that contains the digital signature of the handshake message are as well carried inline. However, it is possible to compress some of the fields inside a Certificate message which is out of the scope of this paper. Pritikin *et al*. propose a scheme to compress X.509 certificates [19].

The ServerKeyExchange message is mostly not sent, either due to crypto export restrictions or because the server's Certificate message contains enough information to concede the client to exchange the premaster secret. However, if it is sent, all fields are carried inline. In case of the optional message CertificateRequest all fields can be omitted. This is possible since the values for the fields certificate_types, supported_signature_algorithms and certificate_authorities can be pre-defined to a single set of supported and preferred values for a 6LoWPAN network and all nodes in the network use the same set of values. The 6BR can populate the empty CertificateRequest message with the default set of values before sending the message to the destination in the conventional Internet. If no default set of values is defined for the 6LoWPAN network, all fields are carried inline.

V. IMPLEMENTATION

We implement Lithe in Contiki [5], an open source operating system for the IoT. However, our proposed header compression mechanisms in Lithe can be implemented in any OS that supports 6LoWPAN. The Lithe implementation consists of four main components: (i) DTLS, (ii) CoAP, (iii) CoAP-DTLS integration module, (iv) DTLS header compression. For

DTLS we use the open source tinyDTLS [20] implementation which supports the basic cipher suite based on pre-shared keys: TLS_PSK_WITH_AES_128_CCM_8. We adapt tinyDTLS for the WiSMote platform and for the 20-bit address support of msp430-gcc [21] (version of 4.7.0). For CoAP, we use the default CoAP implementation [22] in the Contiki OS. We develop the integration module that connects the CoAP and DTLS implementations and enables the CoAPs protocol. This integration allows the application independent access to CoAPs where outgoing CoAP messages are transparently handed to DTLS that transmits the protected messages to the destination. All incoming CoAP messages are protected through DTLS and therefore are processed first at the DTLS layer and handed transparently to CoAP, which resides in the application layer.

We implement our proposed header compression as an extension to the 6LoWPAN implementation in the Contiki OS. The 6LoWPAN layer resides between the IP and Medium Access Control (MAC) layers. The packets from the IP layer that are ready to be transmitted from the node are considered as output packets. The packets from the MAC layer that are received to the node are considered as input packets. The 6LoWPAN layer processes all UDP packets from both directions. Therefore, we use two ways to distinguish UDP packets that carry DTLS messages as payload from other UDP packets. In the case of input packets, the pre-configured default DTLS port is used to identify CoAPs messages. In the second case when the packet is received from the MAC layer, the DTLS port and the ID bits in the NHC-for-UDP and in the NHC for DTLS headers are used to distinguish the compressed headers from the uncompressed. Details are provided in Section IV.

Furthermore, it is important to emphasize, that while applying header compression, the E2E security of DTLS is not compromised. This is due to the design of DTLS and our effort to remain standard-compliant. The header fields are, after final negotiation of the cipher suite, integrity protected within the Record layer. During the compression/decompression process the original headers are not modified and the integrity protection is maintained. After decompression in the 6LoWPAN layer, the integrity of the packet is checked in the DTLS layer. The correctness of integrity protection serves as well as a proof of correct decompression.

VI. Evaluation

We evaluate Lithe on real sensor nodes running the Contiki OS. We use WiSMote [23] as our hardware platform. WiSMotes are equipped with (i) a 16 MHz, MSP430 5-Series, 16-bit RISC microcontroller, (ii) 128/16 kB of ROM/RAM, and (iii) an IEEE 802.15.4 (CC2520) transceiver. We select WiSMotes because of the RAM and ROM requirements of the DTLS implementation, which is discussed in more detail in Section VI-B. The network setup consists of two WiSMotes which communicate directly through the radio. The CC2520 transceiver provides an AES-128 security module. However, for our evaluation we do not use the AES hardware support and rely on software AES. Leveraging the AES hardware support for the cryptographic computations involved in DTLS would lead to higher performance. The focus of our evaluation is on the impact of DTLS header compression on response time and energy consumption of nodes. Therefore, the performance loss due to software AES is not affecting our evaluation. Furthermore, we do not enable link layer security support, in order to be able to analyze the processing overhead of compression separately. In our previous work [24], we have evaluated the performance gains when using the AES support in hardware. There, we implement and evaluate the IEEE 802.15.4 link layer security.

TABLE I
Number of Bits Sent and Space Saving

DTLS Header	Without Comp. [Bit]	With Comp. [Bit]	Space Saving
Record	104	40[1]	62%
Handshake	96	24[1]	75%
`ClientHello`	336[2]	264[2]	23%
`ServerHello`	304	264[3]	14%
`CertificateRequest`	40	0	100%

[1] An additional byte is required to encode both the Record and Handshake headers.
[2] Some fields have a variable length. Here we only consider bits that are always sent.
[3] We do not compromise on security and send full size `random`. All other fields can be omitted.

A. Packet Size Reduction

Using 6LoWPAN-NHC compression mechanisms we can significantly reduce the length of DTLS headers. Table I shows that our proposed DTLS header compression significantly reduces the number of header bits which results in a similar reduction of radio transmission time.

The Record header, included in all DTLS messages, can be compressed by 64 bits saving 62% of space for each message. In the case of the Handshake header, a space saving of 75% is achieved. Application data constitutes the highest amount of DTLS messages. Reducing the Record header from 104 to 40 bits, allows for transmission of 64 bits more payload per packet. Packets that are larger than the link layer MTU are fragmented. Fragmentation does not only introduce more overhead to the node and the network, it brings also security vulnerabilities [4] along. Therefore, it is preferable to avoid fragmentation, whenever possible. Using compression we avoid fragmentation or decrease the number of fragments when the payload is slightly above the fragmentation threshold. Furthermore, reducing the transmitted bits in constrained networks has a huge impact on the performance and lifetime of the network. Radio communication typically has an about 10 times higher energy consumption than in-node computations [23]. The tradeoff with compression is between additional in-node computation overhead for compression/decompression vs. reducing radio transmissions. The impact of this tradeoff is discussed in more detail in Section VI-C.

B. RAM and ROM Requirement

We analyze static RAM and ROM usage with the *msp430-size* and *msp430-objdump* tools in the MSP430 toolchain.

TABLE II
ROM AND STATIC RAM REQUIREMENTS FOR LITHE

Feature	ROM [Byte]	RAM [Byte]
DTLS Crypto (SHA-256, CCM, AES)	6590	2868
DTLS	10662	989
Contiki OS	32145	4979
CoAP	8632	582
DTLS Compression	2820	1
Total	60849	9419

As depicted in Table II, in total 59.4 kB of ROM and 9.2 kB of RAM are required for Lithe.

The DTLS implementation including the cryptographic functionalities and the DTLS state-machine requires 16.8 kB of ROM and 3.7 kB of RAM. This makes DTLS the major contributor of ROM after the OS. The CoAP-Server requires 8 kB of ROM and 0.5 kB of RAM. Our CoAP-Server provides a single resource, that upon a CoAP *GET* request, sends back a response message with variable payload lengths. This is used in our evaluation to analyze the effect of compression on CoAPs messages with different payload lengths. The footprint of the CoAP implementation depends on the offered resources. The implementation of our DTLS header compression 2820 B of ROM and 1 B of static RAM. The 1 B of static RAM holds the compression state of the UDP header. The total ROM used by 6LoWPAN in Contiki for compression and fragmentation (without DTLS compression) is 3782 B. This verifies that the compressed DTLS uses the same order of ROM as standard 6LoWPAN. Today's sensor nodes, such as WiSMote, with 128 kB of ROM can surely accommodate compressed CoAPs along with other operating system components, and still offer significant space to applications.

C. Run-Time Performance

We look at the run-time performance gains that we achieve when compressed DTLS is used and compare it with uncompressed DTLS. We conduct these experiments in a 6LoWPAN network with enabled Radio Duty Cycling (RDC) and respectively with no RDC. When RDC is used, the radio is off most of the time and is turned on either in certain intervals to check the medium for incoming packets or to transmit packets. We use the duty cycled MAC protocol, X-MAC [25] with its default settings, provided in the Contiki OS. In our run-time performance evaluation, we focus on sensor node's energy consumption and network-wide round trip time. For the evaluation of energy consumption, we use the energy estimation module [26] provided by Contiki OS. This module provides the usage time of CPU, LPM, transmitter and transceiver for a certain function call. The absolute timer values for each of these components can be converted to energy with the following equation:

$$\text{Energy } [mJ] = \frac{\text{ticks} \times \text{I } [mA] \times \text{Voltage } [V]}{\text{ticks per second}} \quad (1)$$

1) DTLS Compression Overhead: The overhead caused through in-node computation for compression and decompression of DTLS headers is almost negligible. However, we measure and show it for the sake of completeness.

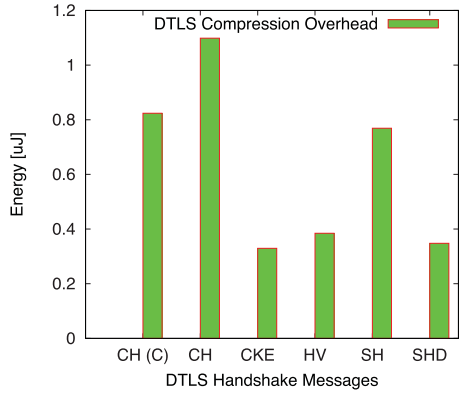

Fig. 8. The energy consumption of individual compressed DTLS messages: ClientHello (CH), ClientHello with Cookie (CH(C)), ClientKeyExchange (CKE), HelloVerify (HV), ServerHello (SH), ServerHelloDone (SHD).

TABLE III
AVERAGE ENERGY CONSUMPTION FOR PACKET TRANSMISSION DURING DTLS HANDSHAKE FOR THE PSK CIPHER SUITE WITH NO RDC. IN AVERAGE 15% ENERGY SAVING FOR THE TRANSMISSION IS ACHIEVED BY COMPRESSION.

Compression	Client-side [uJ]	Server-side [uJ]	Total [uJ]
Without	1756.66	1311.65	3068.31
With	1467.54	1143.47	2611.01

Figure 8 shows the additional energy consumed for compression (compressing/decompressing) of the handshake messages. Each handshake message consists of the both Record and Handshake headers. For a DTLS handshake based on pre-shared keys, on average, 4.2 uJ of energy is consumed for compression.

2) CoAPs Initialization: During the CoAPs initialization phase a secure session is established between the two communicating end-points using the DTLS handshake protocol. The handshake process uses both the Record and Handshake headers, which means that both of these headers can be compressed. The tradeoff between additional in-node computation vs. reduced packet sizes shows itself in the energy consumption for packet transmission in a DTLS handshake. Table III compares the energy consumption required for transmission for the case compression is applied and respectively for the case, where compression is not applied. On average 15% less energy is used to transmit (and receive) compressed packets. This is due to smaller packet sizes achieved through compression.

3) CoAPs Request-Response: Once the CoAPs initialization phase is completed, i.e., the handshake has been performed, a sensor node can send/receive secure CoAP messages using the DTLS Record protocol. Although the Handshake protocol is, compared to the Record protocol, a more resource hungry protocol, it is performed only once during the initialization phase and/or later (rarely) for re-handshake.

In order to measure the performance of compression of the Record Header, we measure the energy consumption and the round trip time (RTT) for the processing of CoAP request-response messages. We start our measurements when the client prepares the CoAP request, and stop after the server's

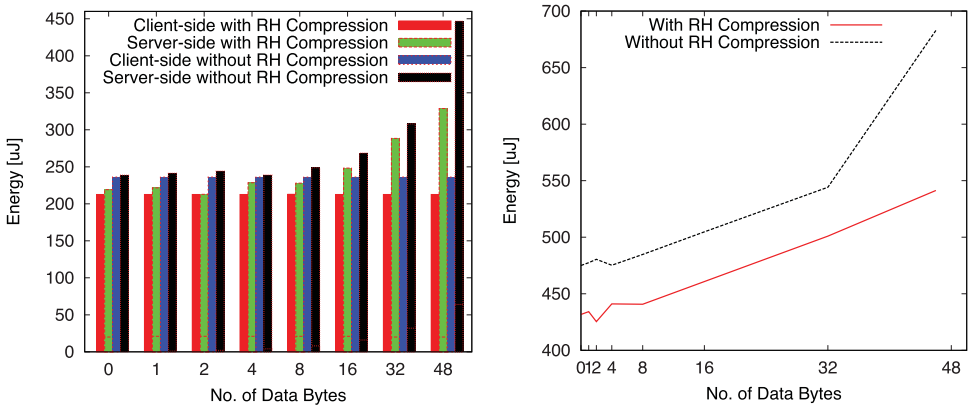

Fig. 9. The energy consumption of CoAPs messages when radio duty cycling is off shows that the compressed CoAPs message consumes less energy; the difference is significant when the messages are fragmented at the 6LoWPAN layer. (a) Energy consumed by client and server on transmission while sending compressed and uncompressed CoAPs messages of different data sizes. (b) Combined energy consumed by client and server on transmissionwhile sending compressed and uncompressed CoAPsmessages of different data sizes.

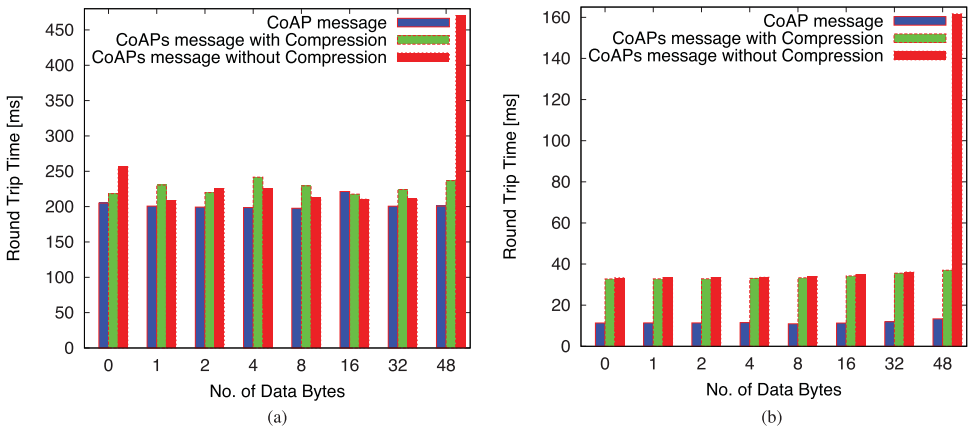

Fig. 10. Comparison of round trip time for Lithe, plain CoAPs, and CoAP. (a) With radio duty cycling. (b) Without radio duty cycling.

response is received and processed. The corresponding CoAP response contains varying payload lengths. To be more precise, eight different payload sizes in the range of 0 to 48 bytes are used. We select 48 bytes, because with 48 byte CoAP payload 6LoWPAN fragmentation is performed in case of plain CoAPs. However, Lithe does not trigger fragmentation, due to reduced bits by means of compression. This effect is visible in Figure 9a, which shows the average in-node energy consumption on CoAPs' client and server for transmitting compressed and uncompressed CoAPs request and response pairs of different sizes with no RDC. The transmission of CoAP *GET* requests has the same amount of energy consumption since the size of request messages are always constant. Hence, energy consumption for CoAPs requests is always reduced by 10% using compression. The energy savings for the CoAPs response messages depend on the payload length and whether compression can prevent fragmentation. The latter is the case for a payload length of 48 byte. Hence, the energy saving is in the range of 4-26%, where the highest energy saving is for 48 byte.

For analyzing the overall energy consumption savings for CoAPs request-responses, we sum up energy consumption for packet transmission on the server and client, as depicted in Figure 9b. We observe that in average energy savings of about 7% are achieved. However, in the case where fragmentation is avoided through compression, the savings increase to 20.6%. This is due to the fact, that with 48 byte payload, 6LoWPAN transmits the packet within two fragments, whereas with compression the packet is transmitted without fragmentation.

The reduced transmission time affects as well the RTT for a CoAPs request-response message. In the case of no RDC, as shown in Figure 10b, the RTT is in average 1.5% smaller, except for 48 byte payload. There, the RTT with compression is even 77% smaller, since fragmentation is avoided. In order to assess the overall overhead caused through security, we have as well added values for CoAP without security. The RTT in CoAP without security is on average 1/3 of the CoAPs, as long as no fragmentation is needed. Looking at the RTT with RDC, as shown in Figure 10b we see that for all three cases of: *(i)* CoAP without any security, *(ii)* plain CoAPs, and *(iii)* CoAPs with DTLS compression (Lithe), RTT values are in the same range, expect for CoAP response messages with 48 byte payload. This is a side-effect of RDC. RDC saves energy by putting the radio into sleep for the most of the time.

However, this happens at the cost of higher latency. Packets in RDC networks are not transmitted directly. The sender has to wait until the receiver wakes up and in the worst case this might be the whole sleeping interval of the receiver. As a result, the overall RTT is higher than when no RDC is used. We observe that in networks with RDC, in case compression prevents fragmentation or decreases the number of fragments, the RTT is significantly reduced. For example, in Figure 10b for 48 byte payload, compression leads to 50% shorter RTT.

VII. Conclusion

CoAP enabled hosts will be an integral part of the Internet of Things (IoT). Furthermore, real world deployments of CoAP enabled devices require security solutions. To this end, DTLS is the standard protocol to enable secure CoAP (CoAPs). In this paper, we investigate the possibility of reducing the overhead of DTLS by means of 6LoWPAN header compression, and present the first DTLS header compression specification for 6LoWPAN. We quantitatively show that DTLS can be compressed and its overhead is significantly reduced using 6LoWPAN standardized mechanisms. Our implementation and evaluation of compressed DTLS demonstrate that it is possible to reduce the CoAPs overhead as the DTLS compression is efficient in terms of energy consumption and network-wide response time, when compared with plain CoAPs. The difference between compressed DTLS and uncompressedd DTLS is very significant, if the use of uncompressed DTLS results in 6LoWPAN fragmentation.

As future work we plan to deploy Lithe in a real world IoT system with a real application scenario. Such an IoT setup consists of constrained devices, standard computers, and smartphones. A real world deployment helps us to thoroughly evaluate in an heterogeneous IoT, and ultimately demonstrate the use of Lithe in security sensitive applications.

References

[1] *Compression Format for IPv6 Datagrams Over IEEE 802.15.4-Based Networks*, RFC Standard 6282, Sep. 2011.

[2] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. (2013, May). *Constrained Application Protocol (CoAP). Internet-Draft draft-ietf-core-coap-16* [Online]. Available: http://datatracker.ietf.org/drafts/current/

[3] *Datagram Transport Layer Security Version 1.2*, RFC Standard 6347, Jan. 2012.

[4] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle, "6LoWPAN fragmentation attacks and mitigation mechanisms," in *Proc. 6th ACM Conf. Security Privacy Wireless Mobile Netw.*, Apr. 2013, pp. 55–66.

[5] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, Nov. 2004, pp. 455–462.

[6] T. Heer, O. Garcia-Morchon, R. Hummen, S. Keoh, S. S. Kumar, and K. Wehrle, "Security challenges in the IP-based internet of things," *Wireless Pers. Commun. J.*, vol. 61, no. 3, pp. 527–542, 2011.

[7] S. Raza, S. Duquennoy, A. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing communication in 6LoWPAN with compressed IPsec," in *Proc. 7th Int. Conf. DCOSS*, Barcelona, Spain, Jun. 2011, pp. 1–8.

[8] J. Granjal, E. Monteiro, and J. S. Silva, "Network-layer security for the internet of things using TinyOS and BLIP," *Int. J. Commun. Syst.*, 2012, doi: 10.1002/dac.2444.

[9] M. Brachmann, S. L. Keoh, O. G. Morchon, and S. S. Kumar, "End-to-end transport security in the IP-based internet of things," in *Proc. 21st ICCCN*, Aug. 2012, pp. 1–5.

[10] T. Kothmayr, C. Schmitt, W. Hu, M. Brunig, and G. Carle, "A DTLS based end-to-end security architecture for the internet of things with two-way authentication," in *Proc. IEEE 37th Conf. Local Comput. Netw. Workshops*, Oct. 2012, pp. 956–963.

[11] J. Granjal, E. Monteiro, and J. S. Silva, "On the feasibility of secure application-layer communications on the web of things," in *Proc. IEEE 37th Conf. LCN*, Oct. 2012, pp. 228–231.

[12] S. Keoh, S. Kumar, and O. Garcia-Morchon. (2013, Feb.). *Securing the IP-Based Internet of Things with DTLS*, [Online]. Available: http://www.ietf.org/1id-abstracts.html

[13] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Making certificate-based authentication viable for the web of things," in *Proc. 2nd ACM Workshop HotWiSec*, Apr. 2013.

[14] C. Bormann. (2012, Mar.). *6LoWPAN Generic Compression of Headers and Header-Like Payloads. Internet-Draft draft-bormann-6lowpan-ghc-04.txt* [Online]. Available: http://datatracker.ietf.org/drafts/current/

[15] *IPv6 Over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, RFC Standard 4919, Aug. 2007.

[16] *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, RFC Standard 6550, Mar. 2012.

[17] *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC Standard 5246, Aug. 2008.

[18] *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC Standard 5996, Sep. 2010.

[19] D. McGrew and M. Pritikin. (2010, May). *The Compressed X.509 Certificate Format. Internet-Draft draft-pritikin-comp-x509-00.txt* [Online]. Available: http://www.ietf.org/ietf/1id-abstracts.txt

[20] O. Bergmann. (2013, Feb. 15). *tinyDTLS* [Online]. Available: http://tinydtls.sourceforge.net/

[21] Texas Instruments. (2013, Feb. 15). *MSPGCC*, Dallas, TX, USA [Online]. Available: http://sourceforge.net/projects/mspgcc/

[22] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," in *Proc. IEEE 8th Int. Conf. MASS*, Oct. 2011, pp. 855–860.

[23] LCIS and Aragosystems. (2013, Feb. 15). *WiSMote Sensor Node*, Valbonne, France [Online]. Available: http://wismote.org/

[24] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt, "Secure communication for the internet of things—A comparison of link-layer security and IPsec for 6LoWPAN," *Security Commun. Netw.*, Jan. 2012, doi: 10.1002/sec.406.

[25] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A short preamble mac protocol for duty-cycled wireless sensor networks," in *Proc. 4th Int. Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2006, pp. 307–320.

[26] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proc. 4th Workshop Embedded Netw. Sensors*, New York, NY, USA, 2007, pp. 28–32.

Shahid Raza, photograph and biography not available at the time of publication.

Hossein Shafagh, photograph and biography not available at the time of publication.

Kasun Hewage, photograph and biography not available at the time of publication.

René Hummen, photograph and biography not available at the time of publication.

Thiemo Voigt, photograph and biography not available at the time of publication.