# Slimfit - A HIP DEX Compression Layer for the IP-based Internet of Things

René Hummen, Jens Hiller, Martin Henze, Klaus Wehrle
Communication and Distributed Systems, RWTH Aachen University, Germany
Email: {lastname}@comsys.rwth-aachen.de

***Abstract*—The HIP Diet EXchange (DEX) is an end-to-end security protocol designed for constrained network environments in the IP-based Internet of Things (IoT). It is a variant of the IETF-standardized Host Identity Protocol (HIP) with a refined protocol design that targets performance improvements of the original HIP protocol. To stay compatible with existing protocol extensions, the HIP DEX specification thereby aims at preserving the general HIP architecture and protocol semantics. As a result, HIP DEX inherits the verbose HIP packet structure and currently does not consider the available potential to tailor the transmission overhead to constrained IoT environments. In this paper, we present Slimfit, a novel compression layer for HIP DEX. Most importantly, Slimfit i) preserves the HIP DEX security guarantees, ii) allows for stateless (de-)compression at the communication end-points or an on-path gateway, and iii) maintains the flexible packet structure of the original HIP protocol. Moreover, we show that Slimfit is also directly applicable to the original HIP protocol. Our evaluation results indicate a maximum compression ratio of 1.55 for Slimfit-compressed HIP DEX packets. Furthermore, Slimfit reduces HIP DEX packet fragmentation by 25 % and thus further decreases the transmission overhead for lossy network links. Finally, the compression of HIP DEX packets leads to a reduced processing time at the network layers below Slimfit. As a result, processing of Slimfit-compressed packets shows an overall performance gain at the HIP DEX peers.**

***Keywords*—Internet of Things, Network Security, Key Management, Compression, HIP, HIP DEX**

## I. Introduction

Peer authentication and secure data transmission are vital aspects for many application scenarios in the Internet of Things (IoT) to prevent leakage of personal information or the execution of harmful actuation tasks, e.g., in building automation or e-health systems. To secure end-to-end connections in the IP-based IoT, a number of lightweight variants of existing IP key management protocols have recently been proposed. These protocol variants most notably include Datagram TLS (DTLS) [1], the HIP Diet EXchange (DEX) [2], and minimal IKEv2 [3]. With its four-way packet exchange, HIP DEX thereby features a more concise handshake than DTLS with 6 round-trips and up to 15 packets and represents an alternative to minimal IKEv2 for establishing an IPsec payload channel.

The specification of HIP DEX follows two central design principles. On the one hand, the basic idea is to adapt the original HIP protocol to the limited processing power, scarce memory resources, and lossy network links in constrained network environments. On the other hand, a key goal is to maintain the general HIP architecture and protocol semantics to remain compatible with the wide range of existing HIP protocol extensions. These extension, among others, include the support for host mobility and multi-homing [4] as well as the integration of IPsec for transport security [5].

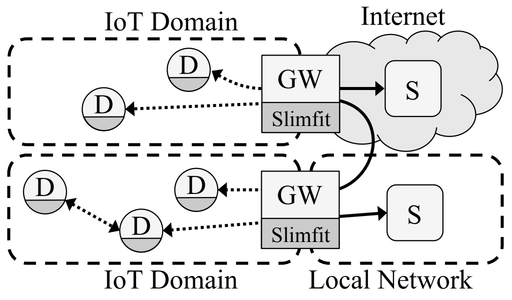
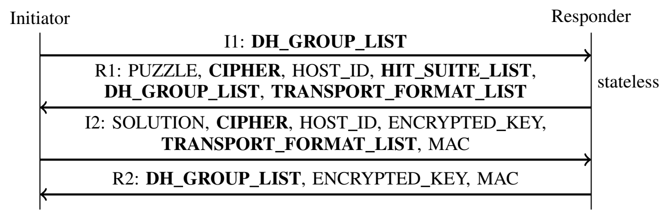


Fig. 1. Resource-constrained devices (D) communicate with each other and with local or Internet-based services (S) via a gateway (GW). Entities belonging to an IoT domain are equipped with our Slimfit layer. Dashed arrows indicate communication paths with compressed HIP DEX packets.

Although HIP DEX already provides a tailored retransmission mechanism and a refined session establishment handshake that reduces the need for public-key-based security primitives to a single Diffie-Hellman operation per peer, its packet structure has not yet been subject to adaptations. As a result, expendable information such as length information of static-size parameters and per-parameter padding still needs to be carried over constrained network links for each packet.

Our contributions in this paper are twofold. First, we analyze the HIP DEX protocol with respect to its packet content and identify to which extent this content can be omitted or compressed before packet transmission. Second, we propose Slimfit, a novel packet compression layer for HIP DEX. Slimfit is located below the HIP DEX layer in the network stack and affords stateless (de-)compression at an on-path gateway and at the communication end-points (see Fig. 1). Moreover, Slimfit preserves the flexible packet structure of the original HIP protocol. Thus, it maintains compatibility with existing and future protocol extensions. Finally, Slimfit is applicable to the original HIP protocol without modifications.

This paper is structured as follows. Section II introduces the network scenario and gives a brief overview of the HIP DEX protocol. In Section III, we discuss and classify expendable protocol information that we identified during our HIP DEX protocol analysis. We then present the design of our proposed Slimfit compression layer and show how to compress expendable HIP DEX protocol information in Section IV. In Section V, we discuss the evaluation results of our proposed Slimfit layer. We cover the security considerations when employing Slimfit in Section VI. Finally, Section VII explores related work and Section VIII concludes our paper.



Initiator
Responder
I1: **DH_GROUP_LIST**
R1: PUZZLE, **CIPHER**, HOST_ID, **HIT_SUITE_LIST**, **DH_GROUP_LIST**, **TRANSPORT_FORMAT_LIST**
stateless
I2: SOLUTION, **CIPHER**, HOST_ID, ENCRYPTED_KEY, **TRANSPORT_FORMAT_LIST**, MAC
R2: **DH_GROUP_LIST**, ENCRYPTED_KEY, MAC

Fig. 2. Protocol diagram of the refined HIP DEX session establishment handshake. Parameters marked bold can be compressed to a single bit. The remaining parameters mainly contain random or cryptographic information.

## II. Prerequisites

We now briefly present the target network scenario for our proposed Slimfit compression layer and give an overview of the HIP DEX protocol as the basis of our work.

### *A. Network Scenario*

As shown in Fig. 1, our target network scenario consists of constrained devices in an IoT domain, services that are located in a local network or the Internet, and gateways that interconnect these network domains. We assume that communication in the IoT domains takes place over constrained wireless links as provided, e.g., by IEEE 802.15.4. Furthermore, we assume that constrained devices are IP-enabled and are equipped with 6LoWPAN, an IPv6 adaptation layer for constrained network environments that is standardized at the IETF [6]. The link from the gateway to the local network or the Internet is a commodity broadband connection.

With respect to the available resources, we assume constrained devices to be equipped with only a few MHz of computational power, several kilobytes of RAM and several *tens* of kilobytes of ROM. Moreover, these devices may be battery-powered. Gateways and services, on the contrary, run on common network and server hardware, respectively.

### *B. HIP DEX*

A key requirement for the design of HIP DEX [2] is to maintain the general HIP architecture and protocol semantics. Specifically, HIP DEX inherits the cryptographic HIP namespace that uses the public key of a device as its host identity (HI). This namespace is used to build a new layer in the network stack between the network and the transport layer. The host identifier (HIT), a fixed-length representation of the HI, serves as a stable device identifier at this layer.

To reduce protocol overhead, HIP DEX specifies a refined session establishment handshake. This handshake consists of a four-way packet exchange between an *Initiator* and a *Responder* (see Fig. 2). The Initiator triggers the handshake with an I1 message. The subsequent three messages then implement a standard authenticated Diffie-Hellman (DH) key agreement. HIP DEX thereby replaces the ephemeral DH keys and digital signatures of the original HIP protocol with static DH keys for mutual peer authentication and key agreement.

Additionally, HIP DEX specifies an aggressive retransmission mechanism to handle packet loss in constrained wireless network environments. This mechanism requires the Initiator to continually send I1 or I2 packets at short time intervals until it receives the corresponding response packet.

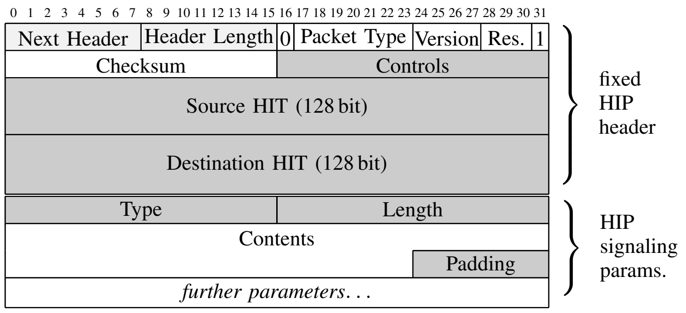


Fig. 3. HIP packet structure consisting of a fixed header and a number of signaling parameters. Each parameter is type-length-value encoded and includes individual padding. Compressible information is marked in gray. The fields marked light gray are handled at the 6LoWPAN layer.

The refined session establishment handshake and the aggressive retransmission mechanism already cater to the limited processing capabilities and the lossy link characteristics of constrained network environments. However, the HIP DEX specification still preserves the verbose packet structure of the original HIP protocol and thus does not yet consider unnecessary transmission overheads. As a result, already the smallest HIP DEX handshake packet exceeds the maximum payload size of an IEEE 802.15.4 frame and must be split into multiple packet fragments for transmission. Packet fragmentation in turn may be harmful [7] and leads to an increased loss probability for the entire HIP DEX packet as the loss of a single fragment results in the loss of the complete packet.

## III. HIP Packet Structure Analysis

As a first step towards optimizing the transmission overhead, we now analyze the HIP DEX packet structure and classify the packet content by its space saving potential during packet transmission. We thereby distinguish three types of packet content: i) semantically irrelevant or static information that can be *omitted*, ii) variable, redundant information that can be represented in a *compressed* form, and iii) information that cannot or should *not be compressed*.

Regarding the packet structure, HIP and HIP DEX share a common packet wire-format for the signaling of protocol information. As depicted in Fig. 3, each packet consists of a fixed protocol header and a varying number of parameters. These parameters carry the actual protocol information (see Fig. 2). Both, the fixed header and the HIP parameters contain information that often is useful in the scope of Internet-based communication, but that constitutes unnecessary overhead during packet transmission in constrained network environments.

### *A. Fixed HIP Header*

Logically, the fixed HIP header is designed as an IPv6 extension header. As such, it starts with the mandatory *Next Header* and *Header Length* fields and requires an 8 byte alignment for the HIP DEX packet content. The HIP-specific part of the header begins with information that is required for the correct parsing of a packet, i.e., the packet type and the protocol version. The two fixed bits enclosing these fields are set for compatibility with other protocols, i.e., SHIM6. The header further contains a checksum for early packet verification and the *HIP Controls* field for information about the packet structure and the host capabilities. As depicted in Fig. 3, the fixed header ends with a HIT address pair that identifies the source and destination at the HIP layer.

**Compressibility:** With 32 bytes, the HIT pair requires the majority of the 40 bytes for the fixed HIP header. This makes HITs the primary target for compression. Regarding their structure, HITs have been designed to resemble IPv6 addresses. As a result, HITs can, for example, be used in combination with existing programming interfaces and as host referrals for IPv6-enabled applications that are unaware of the underlying HIP protocol layer [8]. HITs thereby consist of three components: i) a fixed 28 bit prefix to distinguish HITs from standard IPv6 addresses, ii) a 4 bit HIT generation algorithm identifier, and iii) a 96 bit HI representation that is derived with the indicated generation algorithm. For HIP DEX, this generation algorithm is specified as the left-truncation of the HI to 96 bit.

While highly useful to differentiate HITs and IPv6 addresses at the end-hosts, the HIT prefix constitutes static information at the HIP layer and can therefore be *omitted* from the HIP header before packet transmission. Likewise, HIP DEX only supports a single HIT generation algorithm in its protocol specification. This renders the generation algorithm identifier *omissible* during transmission. Moreover, the HI representation of a HIT can be derived if the packet contains the HOST_ID parameter carrying the corresponding HI. Thus, the source HIT can be *compressed* as redundant information in the R1 and I2 packets (see HOST_ID parameters in Fig. 3).

Concerning the remaining header information, the mandatory IPv6 extension fields, i.e., the Next Header and Header Length, can be modified with existing 6LoWPAN compression facilities [9]. The HIP Controls field denotes a bit array with each bit indicating distinct signaling information. Currently only the highest order bit of the HIP Controls field is defined. It indicates that the signaled HI is anonymous and should therefore not be stored by the peer. As the overhead of multiple anonymous identities likely is excessive for constrained devices, the HIP Controls field typically does not contain any signaling information and can therefore be *omitted* in constrained network environments. Finally, the packet parsing information should remain *uncompressed* in order to allow for immediate identification of retransmitted packets and to afford checksum-based integrity verification without prior decompression. To this end, the checksum field should cover the compressed packet content instead of the original HIP DEX packet.

### B. HIP Parameters

The HIP parameters are type-length-value encoded and include separate padding information to guarantee 8 byte packet alignment (see Fig. 3). Furthermore, parameters in a packet follow a strict numeric order according to their type number. Hence, the position of a parameter in a packet is well-defined.

A gateway or the communication partner can use this encoding of signaling information to parse and process packet content based on parameter type numbers. Moreover, they can skip over unsupported or unneeded signaling information based on the length information of the respective parameters. The HIP DEX specification builds on this packet structure by defining a limited set of *mandatory parameters* that must be understood by the peer in order to set up a security association. Additional protocol extensions can then easily be specified by adding new parameters that are *optional* to be supported.

**Compressibility:** Although required for the alignment of the IPv6 extension header, the semantically irrelevant per-parameter padding unnecessarily adds to the overall packet size. Hence, this padding should be *omitted* during packet transmission. Moreover, the length field of static-size mandatory parameters contains redundant information as the length of these parameters can be derived from the protocol specification. Likewise, the type field of mandatory parameters constitutes redundant information for packets without optional parameters as the order of mandatory parameters is well-defined. Hence, while necessary for a consistent parameter layout, these redundancies render the type and length fields of mandatory parameters *omissible* during packet transmission. The omission of the type and length fields for optional parameters is *not* possible as their occurrence in a packet may vary for the same packet type. Furthermore, optional parameters are not necessarily supported by the communication partner, rendering length information unavoidable for packet parsing.

Regarding *compressible* parameter content, HIP and HIP DEX both provide cipher suite negotiation mechanisms for the peers to agree on a mutually supported cipher suite and to afford modular protocol evolvability. However, especially in the context of constrained devices, the set of supported cipher suites typically is limited to a small number of cryptographic primitives and cipher modes to reduce ROM requirements. This allows to compress default values in the negotiation parameters before packet transmission. Still, this compression should not only be applicable to the status quo, but should evolve with future security recommendations.

## IV. Slimfit compression layer

As shown in the previous section, the HIP DEX packet structure still includes a substantial amount of unnecessary overhead during packet transmission. Hence, we propose *Slimfit*, a novel compression layer that tailors HIP DEX transmission overhead to constrained network environments. Notably, Slimfit is only executed at network entities that belong to the constrained IoT domain, i.e., constrained communication end-points or gateways (see Fig. 1). Thus, end-points and network elements such as firewalls that are located outside a constrained network domain remain oblivious to our Slimfit layer.

We now show how Slimfit integrates in the processing of HIP DEX signaling packets. We then present the compression mechanisms that Slimfit employs in order to reduce the overhead of the fixed HIP header and the HIP signaling parameters.

### A. Integration of Slimfit in the Network Stack

As illustrated in Fig. 4, our proposed Slimfit compression layer is located between the network and the HIP DEX layer in the network stack. Furthermore, Slimfit does not directly interact with the HIP DEX layer. This design enables a Slimfit-agnostic HIP DEX protocol implementation and a transparent integration of Slimfit in the HIP DEX packet processing.

When the HIP DEX layer at a sending node inside an IoT domain generates a new packet, this packet must pass through our Slimfit compression layer before transmission towards the

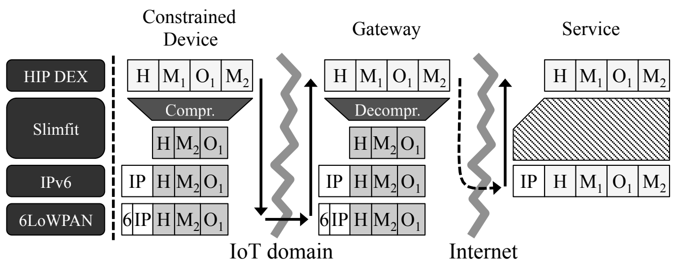


Fig. 4. Integration of our proposed Slimfit compression layer in the network stack. Arrows indicate the processing flow for a HIP DEX packet with an on-path gateway. Slimfit compresses the fixed HIP header (H), reorders the mandatory ($M_i$) and optional ($O_j$) parameters, and compresses or omits these parameters. The dashed arrow indicates that the HIP DEX packet skips the Slimfit layer and is forwarded without compression at the 6LoWPAN layer.

HIP DEX peer. For a constrained device, this can be achieved by placing our compression layer at the corresponding code point in the embedded network stack. A similar integration can be achieved for commodity operating systems by hooking into the network stack via dedicated networking facilities such as *netfilter* for Linux. Slimfit then performs the required HIP DEX packet compression and informs the peer that the packet structure has been modified. To this end, it indicates the used compression mechanisms in the unoccupied bits of the HIP Controls field (see Fig. 5). The Slimfit layer then passes the compressed packet to the network layer for further processing.

As shown in Fig. 4, the compressed HIP DEX packet then reaches the 6LoWPAN layer. Here, the HIP DEX-related IPv6 extension header is further compressed as defined in [9]. Specifically, 6LoWPAN recomputes the Header Length as a multitude of 1 byte units. Thus, compressed HIP DEX packets no longer require padding to obey the 8 byte extension header alignment. To indicate this next header compression, the 6LoWPAN layer prepends an additional byte to the HIP DEX header (i.e., the LOWPAN_NHC field [9]). We propose to use one of the free Extension Header IDs (EID) in this header, e.g., an EID value of 6, to indicate the payload-independent 6LoWPAN extension header compression described above. After 6LoWPAN processing has finished, the packet traverses the network stack without further modification.

On-path nodes, that merely forward packets towards their final destination, do not process HIP DEX packet content. Hence, no special attention is required for compressed packets on the forwarding path. This is unless a gateway is reached that bridges the constrained network domain with an unconstrained network such as the Internet. In this case, Slimfit decompresses the packet after it has been processed at the network layer of the gateway as depicted in Fig. 4. To this end, our proposed compression layer directly hooks into the network layer, e.g., using *netfilter*. Likewise, if a constrained peer receives a compressed HIP DEX packet, Slimfit decompresses this packet before passing it to the HIP DEX layer. Slimfit thereby first determines the indicated compression mechanisms and then applies their respective decompression counterparts.

## B. Compression of the Fixed HIP Header

As depicted in Fig. 5, Slimfit uses the 2 byte HIP Controls field to signal the compression mechanisms that were applied to the original HIP DEX packet. Still, the second byte of this field can be omitted before transmission if all flags at this byte are unset. Slimfit signifies this HIP Controls compression in the first bit of the compressed HIP Controls field. Moreover, Slimfit removes the static prefix of the source and destination HIT for all HIP DEX packets and omits the HI representation of the source HIT for R1 and I2 packets. This compression is indicated by setting the second bit in the HIP Controls field to 1. Finally, Slimfit elides the HIT generation algorithm for the HIP DEX algorithm *"ECDH/DEX"* and indicates this omission in bit 3. We defer the discussion about the evolvability of the latter compression step to Section IV-D.

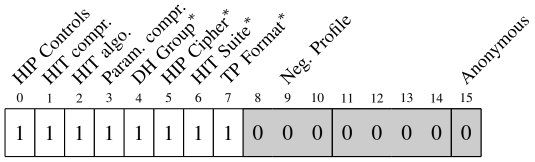


Fig. 5. The compression bits set in the HIP Controls field indicate a HIP DEX packet with maximum Slimfit compression. Negotiation parameter compression flags are marked with an asterisk. Omitted content is highlighted gray. Bits 11 to 14 remain unused.

## C. HIP Parameter Compression

To achieve a high compression rate for the general HIP parameter fields (i.e., type, length, and padding), Slimfit first reorders the parameters of the uncompressed HIP DEX packet. More precisely, Slimfit breaks the strict parameter order of HIP DEX and moves the optional parameters behind the mandatory ones (see Fig. 4). Mandatory parameters thereby maintain their relative position towards each other. Hence, their order remains well-defined for each packet type. The order among the optional parameters is irrelevant for the subsequent compression steps. Slimfit then removes the padding information from all parameters. Furthermore, Slimfit dismisses the length field from all fixed-length mandatory parameters and removes the type field from all mandatory parameters. The latter omission is only possible due to the new parameter structure in Slimfit-compressed HIP DEX packets. Slimfit indicates this *general parameter compression* by setting the fourth HIP Controls bit to 1. The compression is unambiguously reversible by performing the above steps in reverse order.

Slimfit also represents entire HIP parameters as flags in the HIP Controls field (see omitted parameter $M_1$ in Fig. 4 and asterisks in Fig. 5). Specifically, Slimfit can compress the four negotiation parameters DH_GROUP_LIST, HIP_CIPHER, HIT_SUITE_LIST, and TRANSPORT_FORMAT_LIST to a single bit each if the respective parameter only contains default values. We now briefly discuss the main reasons behind our choice of the specific default values for these parameters and refer to Section IV-D for a description of a simple mechanism that provides evolvability of selected default values.

The DH_GROUP_LIST parameter contains a list of well-known DH groups that a device supports. With HIP DEX, this parameter is limited to elliptic curve-based DH groups. Additionally, the possible values of this parameter can further be narrowed down when considering recent recommendations by NIST [10]. Specifically, NIST P-256 is the smallest elliptic curve that is recommended for use after 2013 and that is also defined for HIP DEX. This curve furthermore is expected to be secure until 2030 according to NIST. Hence, we expect constrained devices to use this curve in the HIP DEX handshake and compress the DH_GROUP_LIST parameter if it consists of the corresponding DH group ID. For each of the remaining negotiation parameters, HIP DEX currently only defines a single valid suite ID. Hence, we propose to declare these IDs as default values for the HIP DEX protocol and compress these parameters as individual HIP Controls flags.

## D. Compression Evolvability

On the one hand, Slimfit packet compression reduces HIP DEX packet sizes by omitting and compressing general aspects of the HIP packet structure. This type of compression remains applicable as long as the packet structure does not fundamentally change in future protocol iterations. On the other hand, Slimfit compresses header fields and parameters that contain default values. This choice of default values will not hold indefinitely and may also be scenario-specific. To enable the modification of default values, we additionally propose the use of profiles that define a new set of default values and overwrite our specific choice above. We reserve three bits in the HIP Controls field to indicate the use of such profiles by means of profile IDs. As Slimfit is only deployed inside IoT domains, profiles and their IDs may be domain-specific. Most importantly, our negotiation parameter compression can evolve over time by leveraging these profiles.

## E. Applicability of Slimfit to the original HIP protocol

HIP and HIP DEX share the same packet structure and define mandatory as well as optional parameters for the specified protocol exchanges. Hence, our proposed *fixed header* and *general parameter compression* also apply to the original HIP protocol. However, both protocols differ with respect to their target network scenarios. While HIP DEX focuses on constrained network environments, the original HIP protocol aims at securing communication between comparably powerful Internet hosts. As a result, the cryptographic primitives employed in HIP and, thus, the content of the negotiation parameters differ from the default values for HIP DEX. To achieve similar compression results for the original HIP protocol as for HIP DEX, we propose to define a new negotiation parameter profile as discussed above. Notably, this profile must capture the potential use of RSA or DSA for the HIs of the peers as the main difference between HIP and HIP DEX.

## V. Evaluation

For our evaluation, we implemented the HIP DEX protocol as specified in [2] for the Contiki OS and extended this implementation with our Slimfit compression layer. The HIP DEX implementation employs the *relic*[1] library with elliptic curve NIST P-256 for public-key operations. We used Zolertia Z1 motes that are equipped with a 16 MHz MSP430 microcontroller, 8 kB of RAM, 92 kB of ROM, and an IEEE 802.15.4 radio interface as our evaluation platform for constrained devices. Moreover, we implemented a simple Linux application that utilizes the generic compression libraries *lz77*, *lzma*, *lzss*, and *zlib* (which implements *deflate*) to compress real HIP DEX packet traces. We used this application as a means to compare the compression ratio of our protocol-specific Slimfit layer against alternative generic compression approaches.

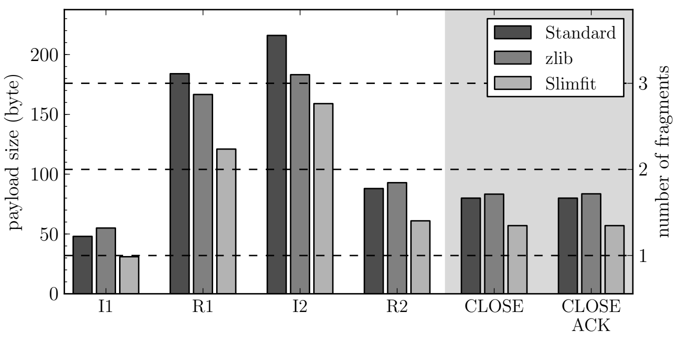


Fig. 6. Packet sizes of the HIP DEX handshake and the session tear down exchange (marked with a grey background) for the standard protocol, with *zlib* compression, and with Slimfit compression. Dashed lines indicate the number of fragments required for the transmission of the HIP DEX packet content.

Regarding packet transmission, we assumed that IoT networks employ *link layer* security to prevent network attacks against, e.g., the local routing structure. To simulate the corresponding header overhead, we decreased the payload size at the 6LoWPAN adaptation layer by 21 bytes. As a result, our evaluation shows packet fragmentation that would occur in secure network scenarios. In such scenarios, the first fragment of a packet contains up to 32 bytes and subsequent fragments at maximum 72 bytes of HIP DEX packet content.

## A. Transmission Reductions

To quantify the transmission reductions of our Slimfit compression layer, we measured the packet sizes of a *standard* HIP DEX handshake and a subsequent session tear down between two wirelessly connected Z1 motes. We then compared this baseline against the size of Slimfit-compressed packets in order to determine the compression ratio of our Slimfit layer. Furthermore, we compared the Slimfit compression ratio to the compression ratio of generic compression algorithms when applied to HIP DEX packets. To this end, we ran our Linux application with the above algorithms against a captured packet trace consisting of 100 HIP DEX handshake and session tear down exchanges. For all generic algorithms we used the highest available level of compression. During our evaluation, we found that, regarding the considered algorithms, *zlib* achieves the best overall space savings for HIP DEX packets. Hence, we focus our discussion on this algorithm.

As shown in Fig. 6, our proposed Slimfit layer is able to compress all HIP DEX packets and outperforms the *zlib* algorithm. More precisely, Slimfit achieves a compression ratio that ranges from 1.36 for I2 packets up to 1.55 for I1 packets. Note that these compression ratios can be achieved for any HIP DEX packet containing the previously defined default values. In contrast, *zlib* has a considerably lower compression ratio of 1.18 in its best case and even adds a small overhead of up to 7 bytes to short HIP DEX packets (see I2 and R2 in Fig. 6). This is due to the constant 11 byte overhead introduced by *zlib*. Moreover, we observed a maximum standard deviation of 0.69 byte for *zlib*. This signifies the modest dependency of *zlib* on the actual packet content.

Regarding the overall transmission overhead, our Slimfit compression layer decreases the HIP DEX handshake size

[1]http://code.google.com/p/relic-toolkit/

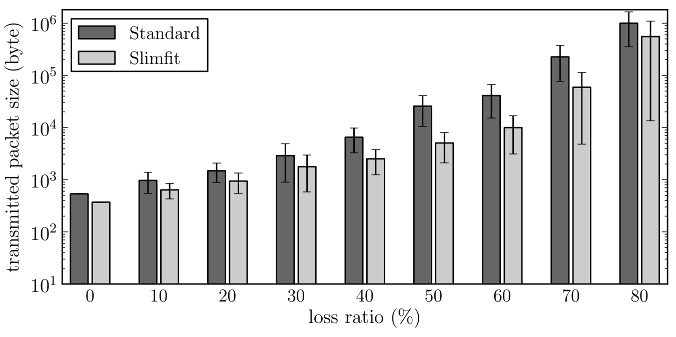


Fig. 7. Overall handshake transmission overhead for the standard HIP DEX protocol and with our Slimfit compression layer for different packet loss probabilities. Note the logarithmic scale of the y-axis. Error bars depict the standard deviation of our measurements.

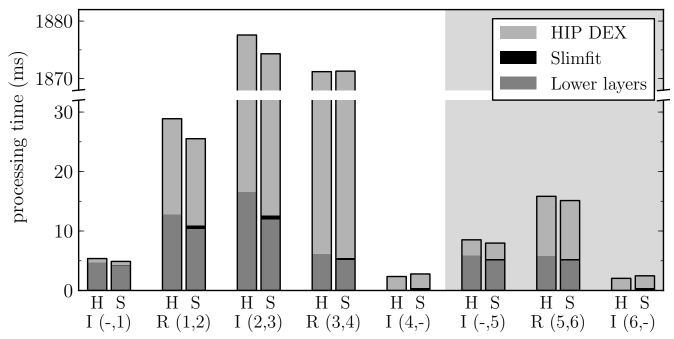


Fig. 8. Processing time of the HIP DEX handshake and the session tear down for the standard HIP DEX protocol (H) and with our Slimfit compression layer (S) for the Initiator (I) and the Responder (R). Numbers in brackets denote $i$-th packet processing and $(i + 1)$-th packet generation.

from 536 bytes to 372 bytes. This is a reduction of 30.6 %. In contrast, *zlib* only achieves a reduction of 7.14 % on average. For the HIP DEX session tear down, Slimfit decreases the overhead from 160 bytes to 114 bytes, a reduction of 28.75 %. With *zlib*, instead, the transmission overhead increases by a total of 6.9 bytes (4.31 %). Thus, Slimfit substantially outperforms the considered generic compression mechanisms.

**Impact on packet fragmentation.** Our proposed Slimfit layer decreases HIP DEX handshake fragmentation by 3 fragments (see I1, R1, and I2 packets in Fig. 6). This constitutes a reduction of 25 % compared to the uncompressed handshake. As a result, the radio utilization and the packet processing overhead caused by HIP DEX on the forwarding path is reduced due to the transmission of fewer packet fragments. Additionally, the reduced packet fragmentation has a positive impact on HIP DEX packet retransmissions. To further quantify this aspect, we analyzed the overall size of a HIP DEX handshake including retransmitted packets between an Initiator and a Responder in the Cooja network simulator for Contiki. We thereby placed both nodes within the direct radio range of each other and measured the average transmission overhead for end-to-end loss probabilities ranging from 0 % to 80 % for individual packet fragments. For each loss probability we performed 5000 handshakes. We decided for simulation over a real testbed to achieve well-defined packet loss probabilities without side-effects on the wireless medium. Note that already the loss of a single fragment results in the loss of the entire corresponding HIP DEX packet as fragment transmission follows the best effort semantics of IPv6.

As depicted in Fig. 7, the decreased packet fragmentation with Slimfit considerably reduces HIP DEX retransmissions. The main reason is that the overall loss probability of a compressed HIP DEX packet decreases as the packet consists of fewer fragments compared to an uncompressed packet. As a result, the average transmission overhead with Slimfit already constitutes about 63.5 % of an uncompressed handshake at a loss probability of 20 %. Notably, this ratio further improves for increasing loss probabilities. Hence, Slimfit not only reduces the packet size of individual HIP DEX packets, but also improves the HIP DEX performance for lossy network links.

## *B. Processing Overhead*

To evaluate the computation overhead introduced by our proposed Slimfit layer, we measured the packet processing time on two wirelessly connected Z1 motes over 100 measurement runs. More precisely, we considered the processing time for packets belonging to the HIP DEX handshake and to the session tear down. We thereby measured the computation time at the HIP DEX layer, our Slimfit layer, and at the layers below Slimfit in the Contiki network stack. Concerning the lower layers, we restricted our evaluation to the processing overhead of outbound HIP DEX packets to prevent overhead misattribution for inbound packet fragments that belong, e.g., to the RPL routing protocol. Hence, inbound HIP DEX packets commonly involve additional computation overhead at the lower layers compared to the overhead depicted in Fig. 8. The standard deviation of our measurements was below 23.09 ms (1.25 %) for public-key-based operations of 1840.86 ms on average and below 0.09 ms for the remaining measured operations.

Fig. 8 illustrates the average packet processing time for uncompressed and Slimfit-compressed HIP DEX packets. Curiously, Slimfit packet compression does *not* result in an overall performance penalty. Instead, it even leads to a modest overall *performance gain* that amounts to 6.58 ms for the complete HIP DEX handshake and to 0.83 ms for the session tear down exchange. During our analysis of this phenomenon, we found that this performance gain mainly stems from a reduced computation overhead at the lower layers that outweighs the low processing overhead at our Slimfit layer (see "Lower layers" and "Slimfit" in Fig. 8). Specifically, Slimfit-compressed packets require less computation overhead at the 6LoWPAN layer due to reduced packet fragmentation. Likewise, the MAC layer has to handle less IEEE 802.15.4 frames for tasks such as transmission scheduling and carrier sensing. The corresponding performance gains amount to 8.61 ms for the complete HIP DEX handshake and to 1.67 ms for the session tear down. These gains would further increase when also considering the overhead of inbound HIP DEX packets at the lower layers.

At the same time, the additional processing time at our Slimfit layer only constitutes a minimum of 0.12 ms for the compression of an I1 packet and a maximum of 0.73 ms for the decompression of an R1 packet as well as the compression of an I2 packet (see "I (-,1)" and "I (2,3)" in Fig. 8). Hence, the performance benefits at the lower layers outweigh the computation cost at our Slimfit layer, resulting in an overall performance gain for Slimfit-compressed HIP DEX packets. To conclude, our Slimfit compression layer not only decreases transmission overhead and packet fragmentation, but also decreases overall processing time of HIP DEX packets.

| Extension | ROM (byte) | RAM (byte) |
|---|---|---|
| Contiki OS incl. HIP DEX | 58659 | 7624 |
| + Slimfit compression layer | 61157 (+2498) | 7624 (+0) |

TABLE I. RAM AND ROM COMPARISON FOR OUR PROPOSED SLIMFIT COMPRESSION LAYER. NUMBERS IN BRACKETS DENOTE ADDITIONAL OVERHEAD COMPARED TO THE UNMODIFIED HIP DEX PROTOCOL.

### *C. RAM and ROM Overhead*

To derive RAM and ROM estimates for our proposed Slimfit layer, we first analyzed the Contiki binary of the unmodified HIP DEX protocol with the *msp430-size* tool. We then compared the results to the binary that also includes our Slimfit layer. Table I summarizes the results of our analysis.

Most importantly, the unchanged, static RAM overhead proves that Slimfit indeed is stateless. This claim is further substantiated by the fact that our implementation does not use dynamic memory allocation, e.g., by means of *malloc*, to maintain per-connection state. However, our Slimfit layer generates a ROM overhead of about 2.5 kB. This constitutes a modest and, in fact, the only tradeoff of our proposed Slimfit compression layer that we identified during our evaluation.

## VI. SECURITY CONSIDERATIONS

We now briefly discuss attacks that an adversary can mount against devices that employ our Slimfit compression layer.

**Exploiting the Slimfit-compressed packet structure.** Content compression before encryption has been shown to leak information about the uncompressed content [11]. Slimfit only compresses or omits packet content that would otherwise be transmitted in plaintext. Hence, it is immune against such side-channel attacks and does not reveal sensitive packet content.

Likewise, packet loss and out-of-order packets may cause an invalidation of the compression context for stateful compression mechanisms [12]. An adversary could aim at exploiting this fact to prevent a Slimfit-compressed HIP DEX packet exchange from completing successfully. However, Slimfit packet compression does not require compression state across HIP DEX packets and thus is resistant to such attacks.

**Targeting the Slimfit packet processing overhead.** *Any* networked adversary could flood a target device with Slimfit-compressed I1 or I2 packets in a Denial of Service (DoS) attack targeting the resulting packet processing overhead. As shown in Sec. V-B, Slimfit modestly reduces the overall packet processing overhead and, thus, would in fact decrease the impact of this attack. Hence, Slimfit does not open a new vector of attack compared to the standard HIP DEX protocol.

**Impact of Slimfit on the DoS protection in HIP DEX.** The HIP DEX protocol offers two handshake mechanisms to protect the Responder against a DoS attack targeting the protocol computation and memory overhead. On the one hand, HIP DEX enables the Responder to delay state creation until the reception of the I2 packet. As a result, the Responder only has to store HIP DEX session state after a successful authentication of the Initiator. On the other hand, HIP DEX employs a puzzle mechanism that enables the Responder to demand an adjustable resource commitment from the Initiator. This allows the Responder to only invest resources into the processing of a received I2 packet *after* the Initiator has committed to the handshake by solving the issued puzzle.

Slimfit does not alter these DoS protection properties of the HIP DEX protocol. Specifically, as Slimfit does not require any compression context to be maintained across packets, a Slimfit-enabled Responder can remain stateless before receiving an I2 packet. Moreover, Slimfit does not modify the puzzle mechanism and even decreases the processing cost of an I2 packet at the network layers that are located below Slimfit. Hence, Slimfit-enabled peers are equally protected against DoS attacks as peers that run the uncompressed HIP DEX protocol.

**Slimfit and on-path network entities.** Slimfit enables gateways that are located at the edge of the IoT domain to perform HIP DEX packet compression. An adversary could flood these gateways with Slimfit-compressed HIP DEX packets in order to exhaust the available gateway resources. As a result, the adversary could hamper packet forwarding between the IoT domain and the external network. However, such flooding attacks would be limited to the computation overhead of Slimfit as it does not require state information for packet compression. Yet, the modest processing cost of Slimfit renders the 6LoWPAN layer a more attractive target.

On-path security appliances such as firewalls may aim at inspecting HIP DEX packets to provide additional protection for the IoT domain. Slimfit changes the packet structure of HIP DEX packets for packet compression. Hence, security appliances that support the filtering of HIP DEX, do not support Slimfit-compressed packets per se. However, Slimfit compression is limited to the IoT domain and, thus, does not impact middleboxes that are located in external networks. Furthermore, security appliances inside the IoT domain can still identify the packet length, type, and destination of a Slimfit-compressed HIP DEX packet without decompression. Hence, blocking of flooding attacks targeting a specific peer or dropping of exceedingly large packets can still be achieved. However, if filtering should occur on additional packet information, the packet must first be decompressed. Yet, the overhead of this operation is modest as shown in our evaluation.

## VII. RELATED WORK

For our discussion of related work, we distinguish three research directions: i) protocol-specific compression mechanisms for the IP-based IoT, ii) generic protocol compression schemes, and iii) further related protocol mechanisms.

Several *protocol-specific compression mechanisms* have recently been proposed in the context of the IoT. The compression of IPv6 headers and extension headers as well as of UDP headers with 6LoWPAN is standardized in [6], [9]. Regarding the compression of IP security protocols, Raza et al. presented initial ideas on 6LoWPAN-compressed DTLS in [13]. Likewise, header compression for IPsec payload channels was presented in [14], [15] and was recently proposed for standardization in [16]. Our work distinguishes itself from these approaches by introducing an evolvable compression scheme for default values and by achieving high compression ratios via design-level changes of the packet structure for compression purposes. Moreover, IPsec compression is complementary to our work and improves the applicability of IPsec as the default payload protection mechanism for HIP DEX.

*Generic protocol compression schemes* have been proposed for IPv6 as well as for upper layer protocols. The IP Payload Compression Protocol (IPComp) [17] allows to apply generic compression algorithms, including *deflate*, to IPv4 and IPv6 packets in order to compress their payload. As shown in our evaluation, such generic algorithms do not leverage domain knowledge about the packet structure and thus typically achieve lower compression ratios for HIP DEX packets than our Slimfit layer. The RObust Header Compression (ROHC) framework [18] consists of protocol-specific compression profiles that, in contrast to our work, often employ stateful compression approaches. Slimfit could be integrated into ROHC as a stateless compression profile for HIP and HIP DEX. Recently, the generic 6LoWPAN-GHC header compression mechanism has been proposed for standardization [19]. This approach introduces a generic compression algorithm that facilitates protocol-specific dictionaries. However, while applying domain knowledge similar to our approach, 6LoWPAN-GHC is limited to compressing the original HIP DEX packet structure and cannot modify the packet structure to achieve a higher compression efficiency. Still, such a generic approach could be used in combination with Slimfit to further compress optional parameters that, e.g., contain certificates.

Apart from packet compression, *further protocol mechanisms* have been proposed that aim at reducing protocol transmissions. In [20], [21], the authors propose TLS extensions that allow clients to cache static server information such as public keys. Furthermore, session resumption mechanisms [20], [22], [23] allow to decrease the transmission overhead of the protocol handshake. However, in contrast to our stateless approach with Slimfit, these mechanisms require an initial handshake to establish the necessary state information at the communication end-points. Still, these mechanisms can be used in combination with our Slimfit layer to further reduce the HIP DEX transmission overhead after an initial handshake.

## VIII. Conclusion

The use of standard protocols in the IoT affords interoperable communication between constrained IoT devices and services. However, device and network constraints necessitate the employed protocols to be tailored towards the special requirements of IoT network environments. In this paper[2], we analyzed and reduced the transmission overhead of HIP DEX, a key management protocol that aims at securing end-to-end connections in the IoT. In our protocol analysis, we identified expendable information in the HIP DEX packet structure. While useful in the scope of Internet-based communication, the transmission of this information is undesirable in constrained network environments. Our proposed Slimfit compression layer removes the identified redundancies in the HIP DEX packet before transmission by: i) applying domain knowledge derived from the protocol specification, ii) modifying the packet structure to increase compression efficiency, and iii) introducing an evolvable compression scheme for cipher suite negotiation parameters. Our evaluation shows that Slimfit achieves a HIP DEX packet compression ratio of up to $1.55$ and reduces packet fragmentation by 25 %. Moreover, Slimfit considerably decreases HIP DEX retransmissions and modestly reduces the overall packet processing overhead. Notably, the marginal ROM overhead for our Slimfit implementation is the only tradeoff for the achieved transmission and computation gains. Finally, our design is not limited to the packet compression of the status quo, but additionally considers the adaption of Slimfit to future security recommendations. To conclude, the integration of our Slimfit layer in the network stack is highly beneficial when securing the IoT with HIP or HIP DEX.

[2]This research is funded by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC).

## References

[1] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, IETF, IETF, 2012.

[2] R. Moskowitz, "HIP Diet EXchange (DEX)," draft-moskowitz-hip-dex-00 (WiP), IETF, 2012.

[3] T. Kivinen, "Minimal IKEv2," draft-kivinen-ipsecme-ikev2-minimal-01 (WiP), IETF, 2012.

[4] P. Nikander, T. Henderson, C. Vogt, and J. Arkko, "End-Host Mobility and Multihoming with the Host Identity Protocol," RFC 5206, IETF, 2008.

[5] P. Jokela, R. Moskowitz, and P. Nikander, "Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)," RFC 5202, IETF, 2008.

[6] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, IETF, 2007.

[7] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle, "6LoWPAN Fragmentation Attacks and Mitigation Mechanisms," in *Proc. of ACM WiSec*, 2013.

[8] T. Henderson, P. Nikander, and M. Komu, "Using the Host Identity Protocol with Legacy Applications," RFC 5338, IETF, 2008.

[9] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, IETF, 2011.

[10] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for Key Management – Part 1: General (Revision 3)," NIST Special Publication 800-57, NIST, 2012.

[11] J. Kelsey, "Compression and information leakage of plaintext," in *Fast Software Encryption*, ser. LNCS. Springer Berlin Heidelberg, 2002.

[12] M. Degermark, H. Hannu, L. Jonsson, and K. Svanbro, "Evaluation of CRTP Performance over Cellular Radio Links," *IEEE Pers. Commun.*, vol. 7, no. 4, 2000.

[13] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN Compressed DTLS for CoAP," in *Proc. of IEEE DCOSS*, 2012.

[14] J. Granjal, E. Monteiro, and J. Sa Silva, "Enabling Network-Layer Security on IPv6 Wireless Sensor Networks," in *Proc. of IEEE GLOBECOM*, 2010.

[15] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing Communication in 6LoWPAN with Compressed IPsec," in *Proc. of IEEE DCOSS*, 2011.

[16] S. Raza, S. Duquennoy, and G. Selander, "Compression of IPsec AH and ESP Headers for Constrained Environments," draft-raza-6lowpan-ipsec-00 (WiP), IETF, 2013.

[17] A. Shacham, B. Monsour, R. Pereira, and M. Thomas, "IP Payload Compression Protocol (IPComp)," RFC 3173, 2001.

[18] K. Sandlund, G. Pelletier, and L.-E. Jonsson, "The RObust Header Compression (ROHC) Framework," RFC 5795, IETF, 2010.

[19] C. Bormann, "6LoWPAN Generic Compression of Headers and Header-like Payloads," draft-bormann-6lowpan-ghc-06 (WiP), IETF, 2013.

[20] H. Shacham, D. Boneh, and E. Rescorla, "Client-side caching for TLS," *ACM TISSEC*, 2004.

[21] S. Santesson and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension," draft-ietf-tls-cached-info-14 (WiP), IETF, 2013.

[22] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards Viable Certificate-based Authentication for the Web of Things," in *Proc. of ACM HotWiSec*, 2013.

[23] R. Hummen, H. Wirtz, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Tailoring End-to-End IP Security Protocols to the Internet of Things," in *Proc. of IEEE ICNP*, 2013.