Towards a Secure Synchronous Communication Architecture for Low-power Wireless Networks

KASUN HEWAGE
Towards a Secure Synchronous Communication Architecture for Low-power Wireless Networks

Kasun Hewage
kasun.hewage@it.uu.se

February 2016

Division of Computer Systems
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

http://www.it.uu.se/

Dissertation for the degree of Licentiate of Philosophy in Computer Science

© Kasun Hewage 2016
ISSN 1404-5117
Printed by the Department of Information Technology, Uppsala University, Sweden
Abstract

The Internet of Things (IoT) is becoming the future Internet where most day-to-day devices are connected to the Internet. These devices are often resource constrained and use low-power wireless communication. Hence networks of them are called Low-power and lossy networks (LLNs). LLN devices may be used in critical applications such as health care, traffic and industrial plants that concern privacy and security, thus their communication has to be protected from malicious activities. LLNs face threats at different levels ranging from transmitting bits wirelessly to applications.

In this thesis, we primarily explore LLN security issues related to application protocols and attacks that target the availability of LLNs. Particularly, we investigate compressing messages of a transport security protocol, DTLS, to make it efficient for LLNs. The IETF proposes to use DTLS for securing CoAP, a specialized web protocol for constrained devices. Furthermore, we experimentally study disrupting the communication of one of the state of the art LLN protocols, Glossy, by attacking its core mechanism.

Secondarily, we aim at improving the performance of TCP in LLNs with mobility over a reliable data link protocol. To this end, we use a Glossy-based communication protocol, LWB, as a reliable data link protocol. We plan to use the evaluation of this work as a stepping stone towards comparing the performance of secure Glossy-based communication protocols.

The main contributions of this thesis are threefold. We propose novel message compression mechanisms for DTLS messages. We also present novel attacks on Glossy, evaluate the effectiveness of them experimentally, and propose potential counter measures. Finally, we show that a reliable data link protocol can improve the performance of TCP in static and mobile settings.
Acknowledgments

I would like to express my gratitude to my main advisor Thiemo Voigt for supporting me in many directions since I started my graduate education. This thesis would not have been possible without his encouragement, guidance and patience. I would also like to thank my co-advisor Shahid Raza for guiding me through the area of Internet of Things security.

Many thanks to all present and former members of the UNO group Ambuj Varshney, Frederik Hermans, Carlos Pérez Penichet and Venkatraman Iyer for being supportive colleagues. I am grateful to all members of the CoRe group, past and present, Per Gunningberg, Christian Rohner, Martin Jacobsson, Hjalmar Wennerström, Laura Marie Feeney, Volkan Cambazoglu, and others for their support and valuable feedback.

I would like to extend my thanks to members of the NES group at SICS, Simon Duquennoy, Zhitao He, Niclas Finne, Nicolas Tsiftes, Joakim Eriksson, Luca Mottola, Joel Höglund, Niklas Wirström, Liam McNamara, and others for their support on Contiki related problems. I would also like to thank Federico Ferrari and Marco Zimmerling for clarifications related to Glossy and LWB implementations.

Finally, I would like to thank all of my friends and my family for their support and encouragement throughout this study.
Included papers

This thesis is based on the following papers


Reprints were made with permission from the publishers.
List of work not included in this thesis


Contents

I Thesis

1 Introduction

1.1 Low-power and Lossy Networks

1.2 LLN Security

2 Research Questions and Contributions

2.1 Securing CoAP

2.1.1 Research question

2.1.2 6LoWPAN header compression for DTLS

2.2 Security in Glossy

2.2.1 Research question

2.2.2 Attacks on Glossy

2.3 TCP Performance in Mobile Cyber-physical Systems

2.3.1 Research question

2.3.2 Low-power Wireless Bus as a Link Layer for IP

2.4 Contributions

3 Related Work

3.1 LLN security

3.1.1 End-to-End security with DTLS

3.1.2 Denial of service attacks

3.2 TCP performance in LLNs

4 Summary of Papers

4.1 Paper I: Lithe: Lightweight Secure CoAP for the Internet of Things

4.2 Paper II: An Experimental Study of Attacks on the Availability of Glossy
Part I

Thesis
Chapter 1

Introduction

In this chapter, we outline low-power and lossy networks (LLNs) and common challenges faced by the development of them. Moreover, we also describe different security related challenges faced by LLNs as this thesis primarily centers on LLN security.

1.1 Low-power and Lossy Networks

The traditional concept of desktop computing is evolving towards so called ubiquitous or pervasive computing which is the idea of computing on any day-to-day device [33]. This concept envisions devices to communicate with other devices and take computing decisions collaboratively. For example, temperature sensors monitor the temperature of a room and send commands to the heating systems to control the room temperature. As the size of the devices is getting smaller, wireless communication has become the desired communication method due to the convenience and practicality.

The power consumption is a critical factor when the devices become minuscule since they are often powered by batteries or energy harvesting. As the wireless communication plays a major role on the power consumption, reducing the energy cost of wireless communication is essential for increasing the life span of the devices. Moreover, it also provides long term economical and environmental benefits since it reduces hazardous materials, i.e. in batteries used during the life time of those devices. As energy saving is one of the primary goals, these devices are typically resource constrained with limited processing power, memory and short range communication. Due to communication link instability and low-power nature, networks of these devices are often called Low-power and Lossy Networks (LLNs). LLNs can be used in diverse applications such as structural health monitoring, smart grids and industrial/home automation.
The development of LLNs faces a number of technical challenges in designing hardware devices and communication protocols due to the constraints such as physical size and power consumption. The low power consumption compels the devices to be resource constrained which has implications on communication protocols. For example, having low-power hardware components on the device prohibits the communication protocols from using comparatively higher data rates as seen in general purpose communication systems. Therefore, LLNs often use low data rate wireless standards for communication such as IEEE 802.15.4 [1].

1.2 LLN Security

As any other computer system, LLNs should also be able to provide their intended services that they are deployed for. Achieving this requires the LLNs to assure certain information security properties such as availability, confidentiality and data integrity.

Since LLNs communicate wirelessly, LLNs are significantly vulnerable to attacks that try to seize the availability of them or denial of service (DoS) attacks. An attacker may try to disrupt the wireless signals in order to prevent reception of packets to a set of LLN devices. For example, an attacker could use a single or collection of devices that are equipped with high power wireless transmitters to jam or interfere the legitimate wireless signals [35]. However, jamming wireless signals with high power transmitters increases the possibility of localizing the origin of the attack [36]. Therefore, a clever attacker may use other methods to seize the availability of LLNs while making localization of the origin of the attack non-trivial. One such method is depleting the energy source of LLN devices by making the radio turn on unnecessarily. An attacker can send fake requests expecting responses to legitimate nodes of the network repeatedly in order to increase the power consumption of their radios which shortens the life span of the LLN.

Another method an attacker can use to disrupt the availability of the LLNs is a misdirection attack in which the attacker injects false information to the LLN so that legitimate nodes of the network react on the false information. In this thesis, we show that the misdirection attack can be used to disrupt the availability of an LLN that uses one of the state of the art LLN protocols Glossy [15] by desynchronizing the devices. In order to avoid being detected, the attacker can also inject false information by impersonating a legitimate device in the network. The attacker can impersonate a legitimate device or steal the identity by replicating a legitimate device, for example, by gaining access to the device physically. For example, an attacker can clone the Internet Protocol (IP) address of a LLN device and injects fake
information to the network in order to make the data traffic flows through the impersonated device [32].

Attackers can launch more complex attacks on specific protocols when they have knowledge about the protocols used. For example, an attacker can announce a lower rank in an LLN that uses RPL as the routing protocol in order to make all the traffic flow through the attacker [32]. In this way, not only the attacker can gain information from the traffic but also opt to selectively forward the traffic.

Securing LLNs is challenging when designing hardware devices and communication protocols due to the resource constraints on LLN devices. For example, sophisticated cryptographic algorithms used in general purpose computer systems may be inefficient to be used with low-power microprocessors of LLN devices. Due to this reason, recent microprocessor designs such as TI’s CC2538 [30] have added hardware acceleration support for widely used cryptographic algorithms such as RSA and Elliptic curve cryptography (ECC). Moreover, low-rate wireless signaling standards often use small maximum transmission units (MTUs) per data frame such that larger packets have to be fragmented. This makes widely used secure communication protocols such as Transport Layer Security (TLS) inefficient to be used over lossy wireless links.
Chapter 2

Research Questions and Contributions

In this chapter, we describe a set of selected research questions related to LLN security and how we address them. We conduct our research mainly in an experimental way in which we focus on concrete research problems to find answers. We formulate hypotheses by analyzing the research problems and conduct real-world experiments in order to validate the hypotheses. In this thesis, we concentrate on three security aspects of LLNs. Firstly, we present our proposal on compressing packet headers of a transport layer security protocol in order to make it efficient for LLNs. Then, we present several attacks on one of the state of the art LLN communication primitives and possible methods to mitigate them. Finally, we focus on performance evaluation of a secure synchronous wireless communication architecture.

2.1 Securing CoAP

Recently, LLNs have gained a significant momentum towards connecting them to the Internet with the concept of the Internet of Things. In other words, Internet of Things describes that connected devices use the protocols of the Internet protocol suite such as the Internet Protocol (IP) to communicate. IP offers a high degree of interoperability at the network layer since it can operate on heterogeneous link layers to provide end-to-end connectivity. For example, users can use their smart phones to control IEEE 802.15.4-based Internet-connected light bulbs from anywhere of the world [29]. While IP is the most widely used network protocol in the Internet, the Representational State Transfer (REST) architecture [16] is getting popular for application protocols in web-based Internet. The systems that conform to the REST architecture (called RESTful systems) communicate
over Hypertext Transfer Protocol (HTTP) style protocols. Similar to HTTP as a RESTful protocol for web-based Internet, the IETF has introduced the Constrained Application Protocol (CoAP) as a web protocol for resource constrained devices [28]. Unlike HTTP which operates on the Transmission Control Protocol (TCP), CoAP operates on the User Datagram Protocol (UDP).

Similar to HTTP communication that is secured by the Transport Layer Security (TLS), the CoAP standard proposes to use Datagram Transport Layer Security (DTLS) to secure the communication. Securing CoAP communication with DTLS requires an initial handshake which consists of a series of message exchanges between the communicating parties. Then, the involving parties can communicate with CoAP securely over DTLS according to the negotiated parameters during the handshake. DTLS supports a wide range of security primitives such as authentication and payload protection.

2.1.1 Research question

The limited maximum transmission unit (MTU) of IEEE 802.15.4 standard (127 bytes) makes DTLS inefficient in LLNs since large DTLS messages may get fragmented at lower layers such as at the 6LoWPAN adaptation layer. Avoiding packet fragmentation is also important from a security standpoint since attacks can be launched that target the 6LoWPAN packet reassembly mechanism [20]. This leads to research question of how to avoid packet fragmentation to make DTLS efficient.

2.1.2 6LoWPAN header compression for DTLS

In order to address above research question, we propose to compress headers of DTLS messages using 6LoWPAN header compression mechanisms [19]. The RFC 6282 defines two encoding formats, namely LOWPAN_IPHC and LOWPAN_NHC, for compressing IPv6 datagrams over IEEE 802.15.4-based networks [19]. The LOWPAN_IPHC encoding format is used to compress IPv6 packet header while LOWPAN_NHC is being used to compress subsequent or next header of the IPv6 header. Usually, the next header is the transport layer header (e.g. a UDP header). Figure 2.1 shows how the above encoding formats are used in a compressed IPv6 datagram.

The LOWPAN_IPHC encoding contains a flag (NH bit) which indicates if the next header is compressed using the LOWPAN_NHC encoding. The LOWPAN_NHC encoding has a variable-length bit-pattern (ID bits) which is used to distinguish different next headers. The ID bits only indicate the compression state of the next header. In other words, the ID bits of a IPv6 datagram carry-
2.1. Securing CoAP

LOWPAN_IPHC
Encoding
In-line IP Fields
LOWPAN_NHC
Encoding
In-line Next Header
Fields
Payload
IPv6 Header
Next Header (e.g. UDP Header)

Figure 2.1: LOWPAN_IPHC and LOWPAN_NHC encoding formats in a compressed IPv6 datagram.

In our compression method, we employ the key idea behind 6LoWPAN header compression that is eliding the fields of the header fully or partially based on their usage. As an example for partial elision of a field, we elide unused bits of the 48-bit sequence number (the bits correspond to leading zeros) that is available in each DTLS record. We fully elide a field if it is not used. For example, we fully elide fragment_offset and fragment_length fields in handshake messages if the message is not fragmented. As in LOWPAN_IPHC, we define several base encodings to indicate the elision level (fully or partially) of the fields.

DTLS is a layered protocol in which the lowest layer is a record protocol. Figure 2.2 shows the structure of the protocols and different handshake messages used in DTLS. A DTLS record can contain a message from either of Change Cipher Spec, Alert, Handshake or Application Data protocols. In our compression method, we employ the key idea behind 6LoWPAN header compression that is eliding the fields of the header fully or partially based on their usage. As an example for partial elision of a field, we elide unused bits of the 48-bit sequence number (the bits correspond to leading zeros) that is available in each DTLS record. We fully elide a field if it is not used. For example, we fully elide fragment_offset and fragment_length fields in handshake messages if the message is not fragmented. As in LOWPAN_IPHC, we define several base encodings to indicate the elision level (fully or partially) of the fields.

DTLS always sends the record header in plain text. Therefore, we consider two cases for compressing the record headers: (1) records during the initial handshake (2) records after the initial handshake is finished. We define two base encodings: LOWPAN_NHC_RHS and LOWPAN_NHC_R which corre

1We use 11011 in our implementation to indicate that both UDP header and the payload are compressed. The RFC 6282 uses 11110 to indicate only the UDP header is compressed.
respond to these two cases. Moreover, we also define another two base encodings: LOWPAN_NHC_CH and LOWPAN_NHC_SH for ClientHello and ServerHello handshake messages respectively which can be compressed further. Figure 2.3 shows how these encodings are positioned in different compressed DTLS records. Note that successive re-handshake messages are encrypted using the negotiated cipher suite. Therefore, we cannot inspect the messages in successive re-handshakes for compressing. Other mandatory messages in the initial handshake do not contain compressible fields hence they are sent only with LOWPAN_NHC_RHS.

We describe how the fields in the record header and the handshake messages are compressed in detail, and the evaluation of our compression method in Paper I.

### 2.2 Security in Glossy

The predominant communication paradigm used in conventional LLNs is the tree-based multi-hop communication. In contrast to hop-by-hop communication using the tree-based structures, Ferrari et al. propose Glossy, a communication primitive that uses network-wide flooding for multi-hop communication [15]. Glossy offers highly reliable and low latency communication with built-in time synchronization within microsecond accuracy. In order to achieve its high reliability, Glossy leverages the fact that simultaneous and equal wireless transmissions interfere at receivers constructively.

The node that starts the Glossy flood is the *initiator* and other nodes are the *receivers* of the flood. As shown in Figure 2.4, after receiving a packet from the *initiator*, all nodes re-transmit the same packet simultaneously. The nodes which have received this packet also re-transmit the same packet simultaneously and this process continues for some predefined number of rounds. In this way, the entire LLN is flooded with in few milliseconds. For IEEE 802.15.4-based LLNs, the temporal displacement among simultaneous
transmissions should be less than 0.5 \text{us} in order to generate the required constructive interference.

### 2.2.1 Research question

Glossy is not designed as a secure communication protocol. Therefore, Glossy-based LLNs are vulnerable to DoS attacks as any other wireless communication systems. Unlike other wireless communication systems, correct operation of Glossy depends on the constructive interference and capture effect \[24\]. Therefore, an attacker could launch attacks that try to break constructive interference and capture effect in order to disrupt Glossy communication. This leads to research question of what would be the effect of attacks that try to break the constructively interference on Glossy-based LLNs.

### 2.2.2 Attacks on Glossy

In order to address above research question, we experimentally study different attacks that we can use to break constructive interference. We identify three different ways of breaking the constructive interference: (1) delaying the transmission of packets (2) sending packets earlier and (3) modifying packets so the receiver does no longer receive identical packets.

During all of the three attacks, correct reception of the packet depends on the capture effect \[24\]. The capture effect is the phenomenon associated with packet reception in which the radio is able to receive a packet from one sender despite simultaneous transmissions from other transmitters. The
attacks (1) and (2) do not modify the packets. Therefore, correctly decoded packets are always legitimate. Since the attack (3) modifies the packet, an attacker could use this attack to inject false information into the packet. Glossy packets contain a 1-byte field called *relay counter* which is used for the implicit time synchronization. As Glossy does not have a mechanism to distinguish legitimate and fake values for this field, we consider modifying the *relay counter* for the attack (3).

We experimentally evaluate the effectiveness of these three attacks and suggest possible counter measures to prevent such attacks in Paper II.

### 2.3 TCP Performance in Mobile Cyber-physical Systems

Cyber-physical Systems bridge the gap between computer systems and physical world by creating interactions between them. A typical CPS consists of three main components: sensors to monitor physical phenomena, computational entities to process sensed data and take decisions, and actuators to interact with the physical world. For a real-world example, a set of autonomous robots is working on gardening activities such as watering and picking fruits [7]. The advancements in heterogeneous networking have enabled the components of CPSs to be physically apart even thousands of kilometers. LLNs are getting popular as a connectivity solution for Cyber-physical Systems (CPSs). For example, the TRITon system uses a battery-operated IEEE 802.15.4-based LLN to monitor lighting levels of a road tunnel [5].

As there exist diverse devices that correspond to the three main components of CPSs, the use of standard protocols makes interoperability convenient. When it comes to networking the devices, the Internet Protocol (IP) offers the highest degree of interoperability compared to others since it has been widely deployed and used. Application developers can use existing libraries such as Internet sockets for developing software for CPSs. Endorsing the concept of Internet of Things, LLNs are now capable of communicating with the most recent Internet protocol, IPv6. The Internet protocol suite (TCP/IP) provides a range of protocols at different layers for communicating with Internet connected hosts\(^2\). In the context of data transport layer, the Transmission Control Protocol (TCP) is the widely used protocol since the web-based Internet is built on top of it.

---

\(^2\)The devices are not necessarily be connected to the public accessible Internet, but rather use protocols in TCP/IP suite.
2.3.1 Research question

Though TCP offers a reliable data delivery between two hosts, it performs poorly over lossy wireless links [2, 6]. While there exist mechanisms to improve TCP performance over lossy wireless links such as TCP westwood [26], they require full-blown TCP/IP stack implementations. However, having a full-blown TCP/IP stack implementation on LLN devices is expensive since they are often low-power and resource constrained. Therefore, popular embedded TCP/IP stack implementations such as uIP [9] exclude non-essential features of TCP such as congestion avoiding for simplicity.

Node mobility intensifies the performance problems of TCP in LLNs since wireless channel varies frequently compared to stationary scenarios. Moreover, mobility may cause changes in the neighborhood of the nodes which also affects routing protocols such as RPL [34]. As the research question, we ask if it is possible to improve the performance of TCP in LLNs by using a reliable link layer for IP.

2.3.2 Low-power Wireless Bus as a Link Layer for IP

The Low-power Wireless Bus (LWB) is a communication protocol which uses scheduled Glossy floods to provide a shared communication bus-like infrastructure [14]. LWB supports multiple traffic patterns such as one-to-one, one-to-many, many-to-one and many-to-many while providing the same communication reliability as Glossy. Moreover, LWB does not maintain topology dependent states such as routes. Therefore, LWB supports node mobility without additional cost. Due to these appealing features of LWB, we explore if LWB can be used as a link layer for IP in order to enable high-performance TCP even in mobile settings.

![Composition of communication slots in a LWB round.](image)

In LWB, the time is split into communication rounds and each round consists of a number of communication slots. During a slot, a node becomes the initiator of Glossy flooding and other nodes become the receivers. A dedicated node in the network called *host* prepares a schedule of communication slots for each round and sends it at the beginning of the round.
Figure 2.5 shows the composition of communication slots in a round. Nodes use data slots to send data and contention slots to inform demand for data to the *host*. LWB has built-in radio duty-cycling and medium access controlling features since the communication schedule decides when the nodes can use the wireless medium for Glossy flooding.

We integrate LWB with uIP stack in order to use LWB as a data link protocol for IP. In LWB, all nodes in the network can receive what other nodes send. Therefore, uIP stack does not require a conventional mean of multi-hop routing. We design two TCP-aware schedulers to handle the TCP traffic efficiently and evaluate them in a real-world testbed with mobile settings in Paper III.

### 2.4 Contributions

The theme of this thesis centers primarily on security aspects in LLNs and secondly improving the performance of TCP using synchronous communication. Towards this end, we categorize our contributions through this thesis into two.

- **In the context of LLN Security**, we present novel and standard compliant compression mechanisms to make DTLS more efficient. Moreover, we implement our proposal on Contiki OS and evaluate it on real-world devices. The evaluation shows that it is possible to achieve a significant gain in energy consumption and network-wide response time.

  We present several novel attacks that target breaking constructive interference on which Glossy depends. Through our evaluation, we show that Glossy is not significantly affected by the malicious nodes that do not obey Glossy’s strict timing constraints in order to generate needed constructive interference. However, we show that Glossy is vulnerable to an attack that modifies the packet, the relay counter. We also propose solutions to protect Glossy-based LLNs from such packet modifying attacks.

- **In the context of improving performance of TCP in LLNs**, we show that using LWB as a link layer protocol for IP improves the performance of TCP. Furthermore, we present two TCP-aware schedulers for LWB and evaluate them on real-world LLN devices with mobility. Our evaluation also shows that LWB enables TCP in mobile settings achieving similar throughput as in static scenarios.
Chapter 3

Related Work

In this chapter, we describe work related to this thesis under two main sections: LLN security and TCP performance in LLNs.

3.1 LLN security

We divide this section into work related to end-to-end security with DTLS and denial of service (DoS) attacks.

3.1.1 End-to-End security with DTLS

End-to-end (E2E) security states that only the parties at the end of the communication can read the messages.

The widely used E2E security protocol in the Internet is TLS [8] while DTLS [27] is preferred for LLNs. To address this asymmetry, Brachmann et al. propose a TLS-DTLS translation at the border router [3]. However, they assume that the border router is a trusted party which breaks the true E2E security. Kothmayr et al. present a DTLS-based E2E security architecture for LLNs [22]. However, they use DTLS messages without any compression which makes DTLS inefficient for LLNs. Granjal et al. study the feasibility of using DTLS with CoAP and note that large payloads may fragment the packets making DTLS inefficient [17]. While the above mentioned approaches use DTLS as it is, in this thesis, we propose compression mechanisms to reduce the overhead of DTLS to make it efficient.

3.1.2 Denial of service attacks

Wireless communication is significantly vulnerable to denial of service (DoS) attacks. Wood et al. describe different types of DoS attacks and prevention methods related to LLNs [35]. The most trivial method of DoS attack is the
jamming or interfering the legitimate wireless signals with high power wireless signals. However, jamming wireless signals with high power transmitters increases the possibility of localizing the origin of the attack [36]. Therefore, attackers could launch more sophisticated attacks targeting specific protocol which makes localizing the origin of the attack non-trivial.

He et al. propose Droplet and Drizzle, new DoS attacks that use payloadless PHY layer frame headers to prevent the reception of legitimate PHY layer frames [18]. Yang et al. present, LearJam, an energy efficient jamming method which estimates the distribution of transmission period of the legitimate nodes [37], so that the attack can be launched only when communication happens. Along the same lines, Law et al. also propose energy-efficient jamming attacks targeting different LLN link-layer protocols [23]. They highlight that even encrypted communication can be jammed as their attacks take temporal arrangement of the packets into account rather than taking actions based on the content of the packets. Liu et al. proposes ED-Jam, an effective dynamic jamming method in which the attacker dynamically adjusts the jamming period to maximize its utility [25]. They claim that their jamming method is more cost-efficient than continuous, random and fixed-period jamming on defending mechanisms that dynamically adjust packet retransmission intervals.

Tiloca et al. present JAMMY, a prevention method for selective jamming attacks against TDMA-based LLNs [31]. The key idea behind their method is randomly permuting the slot utilization pattern, so that the attacker could not simply guess the intended victim’s slot usage. We may explore this idea in the future for securing Glossy-based LLNs.

3.2 TCP performance in LLNs

While there exist mechanisms to improve TCP performance over lossy wireless links such as TCP westwood [26], they require full-blown TCP/IP stack implementations. However, having a full-blown TCP/IP stack implementation on LLN devices is expensive since they are often low-power and resource constrained. In the context of mobility, Dyer et al. [12] and Elaarag [13] extensively study improving TCP performance over mobile Wi-Fi and cellular networks.

Related to LLNs, Dunkels et al. propose a series of extensions to improve the TCP performance across multiple hops based on caching TCP segments on intermediate nodes and locally regenerating TCP acknowledgments [10]. Hurni et al. [21] implement a variant of distributed TCP caching [4]. Their approach targets static scenarios and hence their evaluation does not cover mobile scenarios. For static scenarios, Duquennoy et al. propose a burst
forwarding mechanism that uses TCP to achieve high throughput [11]. In their approach, the end nodes use a full-blown TCP/IP stack instead of an embedded TCP/IP stack.
Chapter 4

Summary of Papers

4.1 Paper I: Lithe: Lightweight Secure CoAP for the Internet of Things


**Summary:** The Internet Engineering Task Force (IETF) has introduced the Constrained Application Protocol (CoAP) as a web protocol for LLNs. The real deployments of CoAP requires security, thus, IETF suggests to use Datagram Transport Layer Security (DTLS) to protect CoAP communication. In this paper, we integrate CoAP and DTLS. In order to make DTLS efficient in LLNs, we propose compression mechanisms for DTLS messages without compromising the security provided by DTLS. We implement and evaluate the compression mechanisms on real hardware devices.

**Contribution:** We propose novel and standard complaint compression mechanisms based on 6LoWPAN header compression for DTLS messages. Moreover, we implement and evaluate the compression mechanisms on real hardware devices.

**My Contribution:** I implemented the 6LoWPAN-based DTLS message compression for Contiki OS and contributed to the evaluation by conducting some of the experiments. Furthermore, I wrote parts of the paper.
4.2 Paper II: An Experimental Study of Attacks on the Availability of Glossy


**Summary:** As Glossy achieves its superior performance by leveraging constructive interference, attacks could be launched to disrupt Glossy communication by breaking constructive interference. Glossy does not have built-in mechanisms to protect against these kind of attacks. Therefore, in this paper, we propose novel attacks that aim at breaking constructive interference and investigate the effectiveness of them on Glossy.

**Contribution:** We present novel attacks that aim to break constructive interference, the underlying mechanism Glossy depends on. We evaluate the effectiveness of these attacks in a testbed. Moreover, we suggest potential countermeasures to protect Glossy against the attacks that tamper the relay counter of Glossy packets.

**My Contribution:** I implemented the attacks on Contiki the implementation of Glossy and conducted experiments on the testbed. I am the main author of the paper and wrote most parts of the paper in collaboration with my co-authors.

4.3 Paper III: Enabling TCP in Mobile Cyber-physical Systems


**Summary:** Through this paper, we argue that a better link layer protocol enables TCP in LLNs with mobility. To this end, we use LWB as a link layer for IP since LWB provides a highly reliable communication in LLNs. As LWB is originally designed for data collection, we design two TCP-aware schedulers for LWB in order to handle traffic efficiently. We integrate LWB with a widely used LLN IP stack, uIP. We conduct experiments on a testbed with a line following robot. In the evaluation, we compare the performance of TCP on uIP over LWB and uIP with RPL.

**Contribution:** In this paper, we present the integration of LWB and
uIP stack. We also present two TCP-aware schedulers for LWB and the evaluation of them on real-world devices.

**My Contribution:** I revised the original implementation of LWB and integrated it with uIP stack, and conducted experiments on the testbed. I am the main author of the paper and wrote most parts of the paper in collaboration with my co-authors. I presented the paper at IEEE MASS 2015.
Chapter 5
Conclusions and Future Work

In this thesis, we mainly explore security related issues in LLNs. We present novel and standard complaint compression mechanisms for DTLS which is suggested to be used to protect CoAP-based communication. The aim of the compression mechanisms is to reduce the overhead caused by DTLS while not compromising the provided security. Our evaluation shows that CoAP over compressed DTLS achieves a significant gain in terms of energy and network-wide response time compared to CoAP over uncompressed DTLS.

While DTLS is providing end-to-end protection, we also look at the attacks on availability of one of the state of the art LLN protocols, Glossy. We present several attacks that try to break constructive interference that is used by Glossy. We show that breaking constructive interference by modifying a special part of the packet has a severe impact on Glossy. In order to protect Glossy-based communication from this kind of attack, we plan to design an Intrusion Detection System (IDS) that detects anomalies caused by packet modifications.

Apart from security related issues in LLNs, we investigate if we can improve the performance of TCP in LLNs by having a better link layer protocol for IP. To this end, we use LWB as a link layer for IP. We integrate LWB with uIP stack and design two TCP-aware schedulers for LWB. Our evaluation on real-world devices shows that LWB as a link layer protocol for IP improves the performance of TCP even in mobile settings similar to static scenarios. We plan to use the results of this evaluation as a stepping stone towards comparing performance of secure Glossy-based communication protocols.
Bibliography


Part II

Papers
Paper I
Paper I
Lithe: Lightweight Secure CoAP for the Internet of Things
Lithe: Lightweight Secure CoAP for the Internet of Things

Shahid Raza\textsuperscript{1}, Hossein Shafagh\textsuperscript{1,2}, Kasun Hewage\textsuperscript{3}, René Hummen\textsuperscript{2}, Thiemo Voigt\textsuperscript{1,3}
\textsuperscript{1}SICS Swedish ICT, Kista, Sweden
\textsuperscript{2}RWTH Aachen University, Germany
\textsuperscript{3}Uppsala University, Uppsala, Sweden

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 IETF. To protect the transmission of sensitive information, secure CoAP (CoAPs) mandates the use of Datagram TLS (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. At the same time, 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.

1 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 connection-less UDP is mostly used in the IoT. Further, HTTP, which is primarily designed to run over TCP, is inefficient in lossy and constrained environments. IETF is working on 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 pump, 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 mechanism 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 more energy than computation. Second, avoiding 6LoWPAN fragmentation that is applied when the size of datagram is larger than the link layer MTU. Avoiding fragmentation, whenever possible, is also important from the security point

---

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

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 the uncompressed CoAP/DTLS.

The rest of the paper is organized as follows. We first summarize related work in Section 2. We give a brief overview of the technologies used in this paper in Section 3. In Section 4, we introduce our DTLS header compression mechanisms. Our implementation is outlined in Section 5. In Section 6, we describe our network setup and discuss the evaluation results. Finally, Section 7 concludes this paper.

2 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.

3 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.

3.1 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 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, 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].
3.2 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 4, 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 4 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.

4 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.

### 4.1 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.

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

### 4.2 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.

Figure 3: Full DTLS handshake protocol. Messages marked with a * are optional.
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 3) 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 saving\(^3\).

**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

\(^3\)The space saving is \(1 - \frac{\text{compressed\_size}}{\text{uncompressed\_size}}\)
packets in 6LoWPAN networks.

4.3 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.

\(^4\)DTLS 1.2 specification increases the cookie size limit to 255 bytes; however, our implementation and evaluation uses a cookie size of 16 bytes.
Figure 6: An *un-compressed* full IP/UDP datagram containing a DTLS ClientHello Message.

*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...
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. 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.

4.4 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 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 4.3. In order to not compromise security the random field in the ServerHello is always carried inline.

4.5 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.

5 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: \texttt{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 4.

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.

### 6 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 6.2. 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.

### 6.1 Packet Size Reduction

Using 6LoWPAN-NHC compression mechanisms we can significantly reduce the length of DTLS headers. Table 1 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.

<table>
<thead>
<tr>
<th>DTLS Header</th>
<th>Without Comp. [Bit]</th>
<th>With Comp. [Bit]</th>
<th>Space Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record</td>
<td>104</td>
<td>40(^1)</td>
<td>62%</td>
</tr>
<tr>
<td>Handshake</td>
<td>96</td>
<td>24(^1)</td>
<td>75%</td>
</tr>
<tr>
<td>ClientHello</td>
<td>336(^2)</td>
<td>264(^2)</td>
<td>23%</td>
</tr>
<tr>
<td>ServerHello</td>
<td>304</td>
<td>264(^3)</td>
<td>14%</td>
</tr>
<tr>
<td>CertificateRequest</td>
<td>40</td>
<td>0</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 1: Number of bits sent and space saving.

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.
<table>
<thead>
<tr>
<th>Feature</th>
<th>ROM [Byte]</th>
<th>RAM [Byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS Crypto (SHA-256, CCM, AES)</td>
<td>6590</td>
<td>2868</td>
</tr>
<tr>
<td>DTLS</td>
<td>10662</td>
<td>989</td>
</tr>
<tr>
<td>Contiki OS</td>
<td>32145</td>
<td>4979</td>
</tr>
<tr>
<td>CoAP</td>
<td>8632</td>
<td>582</td>
</tr>
<tr>
<td>DTLS Compression</td>
<td>2820</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>60849</td>
<td>9419</td>
</tr>
</tbody>
</table>

Table 2: ROM and static RAM requirements for Lithe.

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 6.3.

6.2 RAM and ROM Requirement

We analyze static RAM and ROM usage with the `msp430-size` and `msp430-objdump` tools in the MSP430 toolchain. As depicted in Table 2, 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 mechanism requires only 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.
Figure 8: The energy consumption of individual compressed DTLS messages: ClientHello (CH), ClientHello with Cookie (CH(C)), ClientKeyExchange (CKE), HelloVerify (HV), ServerHello (SH), ServerHelloDone (SHD)

6.3 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:

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

6.3.1 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. Figure 8 shows the additional energy
Table 3: 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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Without</td>
<td>1756.66</td>
<td>1311.65</td>
<td>3068.31</td>
</tr>
<tr>
<td>With</td>
<td>1467.54</td>
<td>1143.47</td>
<td>2611.01</td>
</tr>
</tbody>
</table>

Figure 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.

6.3.2 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
3 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.

6.3.3 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 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 in 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, except 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.

7 Conclusions

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 uncompressed 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 Lithe in an heterogeneous IoT, and ultimately demonstrate the use of Lithe in security sensitive applications.

References


International Conference on Distributed Computing in Sensor Systems (DCOSS’11), Barcelona, Spain, 2011.


Paper II
An Experimental Study of Attacks on the Availability of Glossy
An Experimental Study of Attacks on the Availability of Glossy

Kasun Hewage\textsuperscript{1}, Shahid Raza\textsuperscript{2}, and Thiemo Voigt\textsuperscript{1,2}

\textsuperscript{1}Uppsala University, Uppsala, Sweden
\textsuperscript{2}SICS Swedish ICT, Kista, Sweden

Abstract

Glossy is a reliable and low latency flooding mechanism designed primarily for distributed communication in wireless sensor networks (WSN). Glossy achieves its superior performance over tree-based wireless sensor networks by exploiting identical concurrent transmissions. WSNs are subject to wireless attacks aimed to disrupt the legitimate network operations. Real-world deployments require security and the current Glossy implementation has no built-in security mechanisms. In this paper, we explore the effectiveness of several attacks that attempt to break constructive interference in Glossy. Our results show that Glossy is quite robust to approaches where attackers do not respect the timing constraints necessary to create constructive interference. Changing the packet content, however, has a severe effect on the packet reception rate that is even more detrimental than other physical layer denial-of-service attacks such as jamming. We also discuss potential countermeasures to address these security threats and vulnerabilities.

1 Introduction

The emergence of the Internet of Things (IoT) has increased the demand on distributed, embedded low power applications as it requires more complex communication patterns than the tree-based many-to-sink communication paradigm that is predominant in conventional wireless sensor networks (WSNs). A recent protocol targeting this domain is Glossy \cite{5}, a highly reliable, low latency network flooding mechanism that offers in-built network-wide time synchronization within a microsecond accuracy. The Low-power Wireless Bus (LWB) \cite{4}, built on top of Glossy, provides a shared communication bus like infrastructure with a flat network hierarchy. Together, these protocols provide a performance that has not been achieved by tree-based protocols. Furthermore, the LWB supports the diverse IoT communication
patterns under a simple single layer communication solution. Network-wide flooding enables the support of one-to-many, many-to-one, and many-to-many communication patterns.

Real-world deployment of WSNs, and hence Glossy, require security as WSNs are often deployed in unattended environments and wireless packets are easy to intercept. Furthermore, WSNs are connected through lossy wireless links and require multi-hop communication. This makes WSNs vulnerable to network eavesdropping and message modification as well as attacks seeking to disrupt network operations.

Glossy has not been designed as a secure protocol. It is therefore worth investigating the potential threats and vulnerabilities of Glossy. The core features of Glossy are the generation of constructive interference [5] and the capture effect [11]. Constructive interference occurs when a receiver is able to detect and successfully decode a packet even when packets are generated by multiple transmitters at the same time. A pre-condition is that the transmitters send identical packets. The capture effect is a phenomenon that enables the reception of a packet with relatively high signal strength despite other packets being transmitted almost simultaneously [11]. However, its efficiency decreases when the number of concurrent transmissions increases [13].

As Glossy is one of the first protocols to rely on constructive interference, we propose novel attacks on constructive interference and investigate their effectiveness. Towards this end, we present three novel attacks. The attacks try to break constructive interference by (i) delaying the transmission of packets, (ii) sending packets earlier and (iii) modifying packets so the receiver does no longer receive identical packets which is a precondition for constructive interference.

To evaluate the effectiveness of the attacks, we perform experiments in Flocklab [12], a 30-node testbed that was also used for the original Glossy evaluation. Our results show that the first two attacks are not very effective. The capture effect and low variation of the clock skew under these attacks cause nodes to stay sufficiently synchronized to Glossy phases and enable them to turn on their radios when at least one packet transmission is in progress. Hence, nodes are able to receive packets even though an attack is ongoing. Modifying the packet is more effective, in particular when the attacker tampers the relay counter used for time synchronization.

After evaluating the effectiveness of the attacks, we discuss ways of securing Glossy against these and other attacks as Glossy has no built-in security mechanisms. It is important to protect Glossy packets against unauthorized modifications by employing message security services, in addition to intrusion detection systems that guard against security attacks aimed to disrupt networks.

The main contributions of this paper are:
We present novel attacks that aim to break constructive interference, the underlying mechanism of the Glossy protocol.

We evaluate the effectiveness of the attacks in a testbed. Our results show that tampering the relay counter to break time synchronization is the most effective attack while Glossy internals make it surprisingly robust against the other attacks. We also demonstrate that tampering the relay counter is more effective than physical layer denial-of-service attacks such as jamming.

We discuss potential security solutions to protect Glossy-based networks.

The rest of the paper is organized as follows. First, we provide background information about IEEE 802.15.4, capture effect, constructive interference, and Glossy in Section 2. In Section 3, we describe three methods for attacking Glossy. The experiments and their results are presented in Section 4. In Section 5, we discuss security services for Glossy-based networks. Finally, we review related work in Section 6 and conclude in Section 7.

2 Background

2.1 IEEE 802.15.4 physical layer for 2.4 GHz band

IEEE 802.15.4 radios operating in the 2.4 GHz band make use of the Direct-Sequence Spread Spectrum (DSSS) modulation technique with Offset-Quadrature Phase-Shift Keying. Each byte of the data is split into 4-bit segments and mapped to one of 16 symbols in which each symbol is composed of a sequence of 32 chips. The chips are transmitted at 2 MChips/s corresponding to a maximum data rate of 250 kb/s.

The format of a physical layer (PHY) frame of IEEE 802.15.4 is shown in Figure 1. The preamble is defined to be 4 bytes of 0x00 and the Start of Frame Delimiter (SFD) is one byte set to 0xA7. The frame length is a 7-bit field limiting the length of the maximum PHY payload to 127 bytes.
2.2 Capture Effect & Constructive Interference

The *capture effect* [11] is the phenomenon associated with packet reception in which the radio is able to receive a packet from one sender despite simultaneous transmissions from other transmitters. Suppose, two packets A and B from two transmitters in which the strength of packet A is higher than that of packet B at the receiver. Assume that packet A arrives at the receiver earlier than packet B as shown in Figure 2 (a). The reception of packet A is interfered due to the overlap of packet B. As the radio is busy with the reception of packet A, packet B could be considered as noise. If the signal to noise ratio (SNR) of packet A is higher than that of packet B with a certain threshold, the radio is nevertheless able to receive packet A successfully. Son et al. [17] found that this threshold is hardware dependent.

![Figure 2](image)

*Figure 2: Two scenarios where the capture effect can happen. In (a), packet B arrives while A is being received at the receiver. In (b), packet A arrives while B is being received at the receiver.*

Assume now that packet B arrives earlier than packet A (Figure 2 (b)), packet A has a higher signal strength and the difference in signal strength between packet A and packet B is within the threshold. In such a situation packet B is lost. The correct reception of the packet A depends on the detection of the packet’s preamble and the SFD by the radio. Landsiedel et al. state that in such a situation packet A can be decoded by the radio if it arrives no later than 160 µs after packet B [9].

*Constructive interference* in wireless packet reception happens when two or more packets from concurrent transmitters interfere at a receiver in a way that the receiver can decode the packet correctly. This requires that all the packets are identical and transmitted at the same time. Ferrari et
al. [5] have shown that CC2420 radios can decode the packet in interest when
the temporal displacement among identical concurrently transmitted packets
is smaller than 0.5 µs. For higher displacement, the probability of correct
packet detection decreases significantly. Therefore, generating constructive
interference requires tight time synchronization among the radio transmitters.

2.3 Glossy

Glossy is a flooding architecture for WSNs that uses concurrent transmissions
of packets with identical content to benefit from constructive interference.
The node that starts a flood is called initiator and the other nodes are called
receivers. Upon reception of a packet from the initiator, all receivers relay
the packet at the same time. All nodes that receive the relayed packet also
do the same and this process continues for a number of predefined rounds. In
this way, the whole network is flooded within a few milliseconds. For Glossy
to generate constructive interference, it is essential to have a deterministic
relaying time to align transmissions. Therefore, the implementation of Glossy
uses counteraction techniques for the variations of software execution time
and hardware variations such as varying offsets between the MCU clock and
the radio clock.

Time Synchronization  Glossy also provides built-in time synchronization
with less than a microsecond accuracy and without incurring any additional
cost. Each packet in Glossy has a 1-byte field called relay counter. Initially,
this counter is zero and it is incremented at each node before relaying the
packet. When receiving a packet, a node can determine the number of hops a
packet has made. The slot length is defined as the duration between the start
of two consecutive transmissions of the same packet. The slot length is a
network-wide constant since the packet length is not changed during flooding
and locally estimated by the nodes. Therefore, nodes can use the relay
counter to compute the reference time, the time when the flooding is started
by the initiator. With the knowledge about the initiator’s clock value and
the hop count, other nodes are able to achieve absolute time synchronization.

In the Glossy implementation, a received packet is sent to the MCU from
the radio and stored in a buffer at the MCU in order to increment the relay
counter of the packet. Thereafter, the packet is sent back immediately to the
radio for forwarding.

3 Attacks and Vulnerabilities in Glossy

WSNs are wireless networks of distributed resource-constrained nodes usually
deployed in a way that it is practically hard to physically guard each device
or to protect them with hardware-based tamper-resistant technologies such as with the use of smart cards or Trusted Platform Modules (TPM) [18]. It is therefore possible to capture and clone a node or add faked nodes in the vicinity of a WSN. A malicious node in a Glossy network can severely effect the operation of Glossy as Glossy has no built-in security.

Unlike other networking protocols, Glossy depends on constructive interference. While this makes Glossy efficient we exploit Glossy’s dependency on constructive interference to devise three novel attacks that aim at breaking constructive interference; we later experimentally study the effectiveness of these three attacks in a WSNs testbed.

3.1 Delaying Packet Relay Attack (DPR)

In order to break constructive interference we devise the delaying packet relay attack (DPR) that causes a temporal displacement among signals above 0.5 $\mu$s (Section 2.2). To implement this attack, we use no operation instructions (NOPs) to delay transmission requests to the radio. In the MSP430 MCU, NOPs are emulated from other instructions and their execution time can be calculated by using the Digitally Controlled Oscillator frequency of the MCU. Therefore, it is straightforward to compute the number of NOPs required for a delay above 0.5 $\mu$s.

As shown in Figure 3 we use three different offsets. In the first experiment, we delay the attacker’s packets for a time corresponding to the transmission time of two bytes. This causes the preamble of the attacker’s packets to overlap with the preamble of normal Glossy packets destructively. In the
second experiment, the attacker’s packets are delayed for a time corresponding to the transmission time of four bytes. This causes the SFD of normal Glossy packets to overlap with the preamble of the attacker’s packets. In the third experiment, we delay the attacker’s packets for a time corresponding to the transmission time of more than six bytes. This causes the data segment of normal Glossy packets to overlap with the preamble of the attacker’s packet.

3.2 Relaying Packets Earlier Attack (RPE)
Relaying packets earlier than the planned time of the transmissions is also a possible way to increase the temporal displacement and break constructive interference. In the implementation of Glossy, a certain number of NOPs are used to compensate interrupt serving delay and hardware variations. We decrease the number of these NOPs to relay the packet earlier than the planned time. In order to minimize the latency, Glossy’s implementation limits these NOPs to a small number. Therefore, an attacker is able to relay packets only several microseconds earlier. This is, however, enough to break constructive interference.

3.3 Modifying Packets Attack (MP)
In addition to increasing temporal displacement, modifying packets before relaying them can also be used to make packets interfere destructively since constructive interference requires identical packet content. As mentioned in Section 2.3, Glossy uses a 1-byte field as the (relay counter). Since the relay counter is directly related to time synchronization, an attacker can desynchronize nodes with the initiator by putting bogus relay counter values in the packet header. This could result in packet loss since the nodes become unsynchronized with the initiator and turn on their radio at the wrong time.

4 Experiments
In order to evaluate the effectiveness of the proposed attacks, we conduct experiments in the FlockLab testbed [12] using Tmote Sky sensor nodes.
Figure 4: The topology of the Sky nodes in FlockLab. Node 28 is chosen as the attacker since it can reach many nodes using maximum transmit power.

4.1 Experimental Setup and Implementation Details

We select one of the 30 nodes as the initiator and one or more nodes as the attackers that try to break constructive interference by using the attacks presented in the previous section (DPR, RPE and MP). If not explicitly mentioned, we show results for experiments with one attacker. Figure 4 shows the topology of FlockLab.

The initiator periodically starts a Glossy phase flooding the network. At the receivers, the Glossy scheduler periodically turns on the radio to receive flooded packets. After the end of a Glossy phase, nodes estimate the clock skew between their clock and the initiator’s clock by using the old and the
Recently estimated reference times. The next Glossy phase is then scheduled based on the clock skew and the recently estimated reference time. In this way, receivers are always time synchronized with the initiator which ensures the participation of the receivers in the flooding. At each node, we log relevant statistics such as packet reception, CRC failure count, bad-length and bad-header count.

We run all experiments for 600 seconds. The attacker participates cooperatively in the Glossy flooding during the initial 300 seconds and then starts the attack.

We select large payloads in order to analyze the worst case scenario, i.e., a large number of bytes overlap with the attacker’s packet. To this end, we use 104 bytes including a 4-byte sequence number as the Glossy payload. Initial experiments have shown that the impact of the packet size on the results is almost negligible. To avoid external interference we select channel 26 since this channel does not overlap with WiFi [16].

When we run Glossy without an attacker and node 1 as the initiator, the average packet reception is 99.99% with a standard deviation of 0.02.

4.2 Results with a Single Attacker

We select node 28 as the attacker node and node 1 as the initiator node. The rational behind this choice is that node 28 can reach many nodes when it transmits with the maximum output power.
4.2.1 Results Delaying Packet Relaying Attack

To evaluate the severity of the DPR attack, we calculate the average packet reception during the attack for different offsets. The nodes are able to achieve an average packet reception of 99.75 %, 99.75 % and 99.66 % with a standard deviation of 0.86, 0.75 and 1.09 for 2-byte, 4-byte and 16-byte offsets respectively. These results indicate that the effect of DPR on Glossy’s packet reception is almost negligible.

The resistance of Glossy against the DPR attack can be explained with the capture effect and multiple retransmissions. The attacker is not always able to interfere packets destructively when the capture effect makes the receiver lock onto the packet from a legitimate node. Glossy’s multiple retransmissions of the same packet enables nodes to receive the packet at least once even though the attacker’s packets interfere destructively at other times.

Figure 9 shows CRC failure, bad-length and bad-header counts for node 18 regarding 2-byte, 4-byte and 16-byte offsets. The figure depicts a rapid increment of CRC errors after the attack has been started when the 16-byte offset is used compared to 2-byte and 4-byte offset. Some nodes experience a large number of CRC errors over time compared to others. These results show that the attacker’s packets destructively interfere with packet reception at node 18 but that the nodes still stay synchronized.

In addition to the relay counter, Glossy also uses the packet reception time (the time when the SFD interrupt is raised by the MCU) for estimating the reference time of the initiator. As the DPR attack changes the packet
reception time, we plot the variance of the clock skew of the nodes against the percentage of the nodes as a whole for the attack with 16-byte offset. According to Figure 5, about more than half of the nodes of the network experience a high variance in clock skew.

4.2.2 Results Relaying Packets Earlier Attack

We remove NOPs that are used to compensate the interrupt serving delay and hardware variations to send packets earlier. As a result, the transmission request is sent 3.10 $\mu$s earlier to the radio. As mentioned in Section 3.2, this time duration is theoretically enough to break constructive interference.

However, the nodes are able to achieve an average of 99.87 % packet reception with a standard deviation of 0.55. Except for node 23, there are no noticeable increases of CRC errors. As with DPR, we plot the variance of the clock skew of the nodes against percentage of the nodes as a whole for the attack. Figure 6 shows that all nodes experience almost the same variance of the clock skew during the attack as before. Compared to DPR with 16-byte offset, RPE with 3.10 $\mu$s causes less variance in the clock skew for all nodes during the attack.

4.2.3 Results Modifying Packets Attack

Figure 8 shows the reception rate for all nodes during the MP attack. Except for the attacker and five nodes (2, 4, 8, 15 & 33), all other nodes experience
considerable packet loss. Notably, the attacker (node 28) is also able to achieve comparatively high packet reception while the attack is being performed. The tables 1 and 2 show hop count information from both initiator and attacker nodes. Based on the hop count, the topology of FlockLab and packet reception we note that nodes closer to the initiator, as expected, achieve a higher packet reception.

As with DPR and RPE, we plot the variance of the clock skew of the nodes against the percentage of the nodes as a whole for the attack to describe the packet loss. Figure 7 shows that more than 30% of the nodes experience a significantly high variance in the estimated clock skew. The magnitude of variance of the clock skew for MP is significantly higher than DPR and RPE (Figures 5, 6).

Glossy relays received packets without a validation of the relay counter. This results in propagating fake relay counter values to neighboring nodes. Since the reference time estimation is mainly based on the relay counter, the estimated reference time on the nodes that receive packets with fake relay counter values becomes incorrect. Therefore, nodes lose time synchronization with the initiator which causes packet loss. When many such unsynchronized
nodes serve as relay points to other nodes, it is possible that a part of the network is isolated due to packet loss.

During the attack, some nodes experience a high variance in PRR. According to Table 2 most of the nodes with a low PRR are in direct vicinity (one hop away) of the attacker. As these nodes do not contribute to the Glossy flooding during the attack, other nodes have to rely on packets from synchronized nodes. Since packet receptions under constructive interference are not correlated \([2]\), packet reception on the nodes that are in the edge of the vicinity of synchronized nodes varies which causes a high variance in PRR.

<table>
<thead>
<tr>
<th>Hops</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6, 8, 16, 18, 19, 20, 22, 23, 27, 31, 33, 204</td>
</tr>
<tr>
<td>2</td>
<td>1, 2, 4, 13, 15, 17, 24, 25, 26, 32, 200, 202</td>
</tr>
<tr>
<td>3</td>
<td>7, 11, 14</td>
</tr>
</tbody>
</table>

Table 2: Hop count from the attacker (Node 28)
<table>
<thead>
<tr>
<th># of attackers</th>
<th>Attacker nodes</th>
<th>PRR (%)</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>99.66</td>
<td>1.09</td>
</tr>
<tr>
<td>2</td>
<td>28, 33</td>
<td>98.77</td>
<td>2.10</td>
</tr>
<tr>
<td>3</td>
<td>28, 33, 20</td>
<td>98.38</td>
<td>2.09</td>
</tr>
<tr>
<td>4</td>
<td>28, 33, 20, 25</td>
<td>97.80</td>
<td>3.80</td>
</tr>
<tr>
<td>5</td>
<td>28, 33, 20, 25, 18</td>
<td>92.91</td>
<td>6.75</td>
</tr>
</tbody>
</table>

Table 3: Average packet reception during the DPR attack with 16-byte offset with multiple attacker nodes.

4.2.4 Other Single Attacker Experiments

We also conduct experiments where the attacker modifies other parts of the packets. We let the attacker modify a 50-byte portion of the 104 byte payload instead of the relay counter. This 50-byte portion of data is not used internally by Glossy or our test application. The nodes are able to achieve an average packet reception of 99.97% with a standard deviation of 0.08. The high packet reception shows that modifying other parts of the packet does not affect Glossy’s operation as much as modifying the relay counter. Due to the capture effect, nodes often receive unmodified packets from other neighbours rather than modified ones from the attacker. As the relay counter remains untouched, Glossy’s reference time estimation stays unaffected resulting in correct time synchronization with the initiator.

In order to study the network-wide effects of DPR when the attacker is closer to the initiator, we select node 4 as the attacker and node 1 as the initiator. The nodes are able to achieve an average packet reception rate of 99.94% with a standard deviation of 0.14. This result indicates that the effect of a single DPR attacker that is close to the initiator is also negligible.

4.3 Experiments with multiple attackers

Our previous results show that DPR with a 16-byte offset is more effective than DPR with 2-byte, 4-byte and RPE. Since we have shown that the MP attack affects more than half of the network’s nodes with only a single attacker, we conduct experiments with multiple attacker nodes for DPR with a 16-byte offset. Table 3 shows the average packet reception rate (PRR) and standard deviations for DPR attacks with multiple attackers. The results show that as expected employing multiple attackers increases the effectiveness of DPR with a 16-byte offset.

4.4 Comparison with physical layer jamming attacks

Since the MP attack is the most severe attack, we compare the severity of it with other conventional physical layer denial-of-service attacks. To this
end, we use the Drizzle attack [7], and a jamming attack that continuously transmits a randomly modulated carrier signal [1]. In the Drizzle attack, an attacker transmits bogus packet headers in order to force receivers into an unavailing packet decoding process. This attack is executed with high frequency avoiding the need to be synchronized with duty cycled receivers. We select the randomly modulated carrier signal over the unmodulated carrier signal as jamming attack since its power spectrum evenly spreads over the entire channel bandwidth whereas the power spectrum of the unmodulated carrier signal peaks at the center frequency of the channel. We use a special test mode of the CC2420 radio of Tmote Sky sensor nodes in order to generate the randomly modulated carrier signal.

As in the previous experiments, we select node 28 as the attacker and conduct experiments on the testbed for the same time duration. We start the attack after the experiment has been ongoing for 300 seconds.

Figure 10 depicts the comparison of PRR of the nodes for the three attacks. The figure shows that the MP attack affects significantly more nodes than the other two attacks. Both Drizzle and the modulated carrier jamming attack significantly affect packet reception on the same, small set of neighboring nodes (nodes 22, 16 and 6). Unlike in the MP attack, both Drizzle and the jamming attack do not propagate bogus packets over multiple hops and therefore affect less nodes.

5 Security Services for Glossy

We have seen that Glossy is subjected to different attacks. Most severe of them is tampering with the relay counter that is primarily used for time synchronization. A fake relay counter value de-synchronizes the network.
timing. In glossy-enabled networks, it is necessary to keep the temporal displacement below 0.5 $\mu$s to generate the constructive interference.

Currently, Glossy has no built-in security mechanisms to protect Glossy floods against unauthorized modification. In real deployments it is important that networks are protected against known attacks.

The CIA triad model is a widely accepted information security model which describes three properties that a secure system should adhere to [14]. Confidentiality describes that no unauthorized parties should be able to view or make sense of the information. Integrity describes that information resources should not be altered without being detected. Availability describes that an information system should be available to serve its purpose when needed.

In the case of Glossy the most important service is availability which is necessary for the operation of Glossy. The compromise of the integrity service, i.e., to tamper with the relay counter, also results in the compromise of the availability service. Protecting the integrity of Glossy packets in general and the relay counter in particular is necessary for the smooth operation of Glossy.

As we have shown in Section 4.4, conventional physical layer denial-of-service attacks such as jamming can also be launched against Glossy; however, these attacks are energy-consuming [10], and at the same time are comparatively easy to detect. On the other hand, launching de-synchronization attacks against Glossy or LWB by occasionally tampering the Glossy packet is hard to detect.

The first defense mechanism to guard against security attacks, aimed to disrupt networks, is to prevent them by implying proper security mechanisms. In addition to the prevention mechanisms, intrusion detection systems are also important to detect and terminate successful attacks.

5.1 Attack Prevention in Glossy

Among the CIA security services, integrity protection of the Glossy flooding packet especially the relay counter is most important. Source integrity, also called authentication, is important to grant access to a node to become part of Glossy flooding. Later, for each flood data integrity of the relay counter should be ensured, which protects the modification of the relay counter beyond the legal allowed increment of one.

Considering the strict timing requirement in Glossy it is hard to build security into Glossy. In addition to other parameters, Glossy relies on the deterministic software delays to keep the temporal displacement below 0.5μs. Also for additional security processing, Glossy should be able to measure the delays deterministically. For cryptographic processing it is challenging
to measure these delays with such strict timing requirements. However, such a built-in security is a clean solution for Glossy and a potential future addition in Glossy. As a starting point, such novel solutions can exploit the mechanisms introduced in Glossy to determine the software delay [5]. The Glossy property that each node relays packets immediately after receiving and these concurrent transmission must be aligned is challenging to achieve when executing cryptographic operations during the Glossy flooding.

IEEE 802.15.4 security [15] can also be used to protect the Glossy flooding packet using hardware encryption; however, this too requires deterministic delays. Both security solutions discussed above require a shared secret among the nodes, and compromise of a single node will compromise the whole network.

5.2 Attack Detection in Glossy

It is generally hard to embed cryptographic security in routing protocols; usually security is provided at the layers below the routing layer. For example, in IEEE 802.15.4-based networks the routing information as well as the application data can be protected at the link layer. On the other hand, intrusion detection systems (IDSs) are used to detect attacks aimed to disrupt network operation. While most conventional WSN network architectures deploy routing protocols [6], Glossy is unique in that it does not require any explicit routing protocol since it solely relies on flooding. Differently from other routing protocols [6], Glossy and hence LWB are inherently robust against some of the conventional routing attacks such as selective-forwarding attacks [8], sinkhole attacks [8], and wormhole attacks [8]. This is due to the fact that every node immediately relays the packet it receives and the packet quickly propagates through the network.

IDSs can run on top of Glossy and perform overall monitoring without disturbing the normal operation of Glossy. However, for an IDS for Glossy-based networks an interface is required between Glossy and detection modules to exchange security parameters. For example, Glossy can update an IDS module about the relay counter values it receives. In return, an IDS can monitor the activities of individual nodes by cooperatively sharing the information in a centralized IDS module. Once malicious nodes are detected, they can be replaced, for example, by adding them in a blacklist maintained by a centralized IDS module. Later, these blacklisted nodes can be physically tracked and removed. While the application needs to be aware of such replacements, Glossy itself is stateless which simplifies node replacement since Glossy does need to keep any state about parents, link qualities etc.
6 Related Work

We split the related work section into two parts. First we present a review of Glossy-based and similar protocols and then we discuss Denial-of-Service (DoS) attacks for wireless sensor networks as they are the attacks that are most similar to those we present in this paper in that they also aim at disrupting availability.

Several other protocols have used constructive interference for building communication primitives based on Glossy. Doddavenkatappa et al. propose Splash, a fast data disseminating protocol that is based on Glossy [3]. Splash integrates Glossy to extend a pipeline transmission scheme into a tree-based pipelining scheme for disseminating large data objects. Landsiedel et al. propose Chaos, an all-to-all data sharing and in-network processing primitive for low-power wireless networks [9]. Chaos makes use of both the capture effect and constructive interference at different stages of the protocol operation. As these works are based on Glossy, they also inherit Glossy’s security vulnerabilities. Instead of constructive interference, Flash Flooding relies solely on the capture effect to ensure that each node receives the flooded packets from at least one of its neighbors [13]. Studies have also shown that the reliability of packet reception in glossy-based protocol decreases when the number of concurrent transmitters increases [3, 19]. While these studies try to improve the packet reception during normal protocol operation, we propose and evaluate attacks that aim at breaking constructive interference.

Wood et al. describe different types of DoS attacks in wireless sensor networks [20]. Among them, our attacking methods are related to collision (DPR and RPE attacks) and misdirection (MP attack) attacks. The attacks that disrupt the service by intentionally colliding packets are categorized into collision attacks while attacks that inject false information to disrupt the service are categorized into misdirection attacks. He et al. propose Droplet and Drizzle, new DoS attacks that use payload-less PHY layer frame headers to prevent the reception of legitimate PHY layer frames [7]. The receivers synchronize into attacker’s payload-less PHY layer frames which will be eventually dropped due to CRC errors. Drizzle is the high-frequency variant of Droplet that we have used in our experiments.

7 Conclusions and Future Work

In this paper we have presented novel attacks that aim to break constructive interference in Glossy, a recent extremely efficient protocol for distributed wireless sensor networks. Our experimental evaluation shows that modifying the packet content is more effective than temporal displacement when relaying packets. We have discussed security solutions that protect Glossy and other
networks relying on similar mechanisms against these attacks. Future work will then focus on preventing and detecting these attacks.

Acknowledgements

We would like to thank Federico Ferrari and Marco Zimmerling for clarifications regarding Glossy internals. This work was supported by SSF and the EC project CALIPSO with contract number FP7-ICT-2011.1.3-288879.

References


Paper III
Paper III
Enabling TCP in Mobile Cyber-physical Systems
Enabling TCP in Mobile Cyber-physical Systems

Kasun Hewage¹, Simon Duquennoy², Venkatraman Iyer¹ and Thiemo Voigt¹,²

¹Uppsala University, Uppsala, Sweden
²SICS Swedish ICT, Kista, Sweden

Abstract

Cyber-physical systems consist of several wirelessly connected components such as sensors that monitor physical phenomena, computational entities that make decisions based on sensed information and actuators that interact with physical processes. Connecting cyber-physical systems to the Internet using IP protocols increases interoperability by avoiding the need for protocol translation gateways. Unfortunately, in this context TCP has been disregarded since it is known to perform poorly in wireless scenarios as it treats packet loss as an indicator for network congestion rather than poor link quality. In this paper, we use the Low-power Wireless Bus (LWB) as a link layer for TCP/IP, taking advantage of its reliability and its routing-free communication. We design a system that integrates LWB with a low-power IP stack and includes TCP-aware schedulers for LWB. We evaluate our system with experiments on real hardware using uIP, a popular embedded Internet protocol stack. Our results demonstrate high TCP throughput in mobile and static scenarios and, furthermore, show that mobility does not decrease TCP performance.

1 Introduction

Cyber-physical systems (CPSs) are designed to enable interactions between computational entities and the physical world. CPSs have gained an increased commercial acceptance in recent years, and are widely used in various monitoring and actuation applications in residential and industrial settings. Typically, CPSs consist of sensors that monitor physical phenomena, computational entities that make decisions based on sensed information and actuators that interact with physical processes. An example application is a set of autonomous robots working on gardening activities such as watering and picking fruits [8]. Each plant is attached with sensors to monitor the soil condition and sensed data are sent to a decision making system. When the
decision making system detects the water levels are low, autonomous robots are instructed to water the plants.

The recent push towards the Internet-of-Things (IoT) calls CPSs to be accessible through the Internet. In this context, the use of standard Internet protocols enables interoperability among vendors and link-layer technologies, and eases maintainability. For example, application developers can use existing libraries such as network sockets for communication which also allows to network so called heterogeneous smart objects together with other components of the CPSs.

Link reliability in low-power wireless networks (LWNs) depends on physical factors such as non line-of-sight signal propagation which induces multi-path effects. Moreover, LWNs often experience cross-technology interference [19, 20, 21]. Therefore, links in LWNs are often lossy in nature. TCP was originally designed for wired networks that exhibit communication link reliability several orders higher than that of LWNs. Early results of using TCP over lossy wireless links have shown a degraded application performance because of increased packets loss [3, 7, 29]. Due to this discouraging performance in IoT scenarios, the Constrained Application Protocol (CoAP) [33] protocol was designed as a lightweight alternative of HTTP for the IoT, based on UDP rather than TCP. Nevertheless, a well-performing TCP would simplify tasks that require reliable data transfer such as remote reprogramming of sensor nodes.

Mobility exacerbates the performance problems of TCP since wireless channel dynamics are usually harsher in mobile compared to stationary scenarios. Moreover, mobility may cause changes in the neighborhood of the nodes which also affects routing protocols. In this context, the standard LWN routing protocol for IPv6, RPL [34], performs poorly in mobile scenarios because it is not designed for frequent route changes [4, 25]. The latter cause longer delays and disconnections in application protocols. Therefore, RPL cannot be used with highly mobile CPSs such as a team of areal drones where reliable and predictable delivery is required.

In this paper, we argue and demonstrate that by using adequate lower layer protocols we enable high-performance TCP even in mobile settings. In particular, we use the Low-Power Wireless Bus (LWB), a communication protocol that supports several traffic patterns with high reliability for multi-hop LWNs [17]. LWB uses scheduled Glossy floods [18] in which no topology dependent states, including routes, are kept. We design TCP-aware schedulers for LWB that are able to handle traffic efficiently, even in the presence of fragmentation. Our evaluation with Contiki’s uIP [11] shows that LWB is a good candidate reliable link layer protocol for TCP/IP in mobile CPSs. Our contributions are the following:
We integrate LWB in a widely used TCP/IP stack.

We design two TCP-aware schedulers for LWB.

We experimentally show that using LWB as link layer protocol enables TCP in mobile settings achieving similar throughput as in static scenarios.

The remainder of this paper is organized as follows. First, we provide background information about TCP, 6LoWPAN packet fragmentation, and LWB in Section 2. In Section 3, we describe how we use LWB as a link layer protocol for an IPv6-based TCP/IP stack. Section 4 presents details of our implementation on Contiki OS and uIP stack. We present the evaluation of our system on real hardware in Section 5. Before concluding we review related work in Section 6.

2 Background

To enable reliable TCP communication in mobile cyber-physical systems, we use the Low-power Wireless Bus (LWB) [17] as the underlying data link layer. The TCP protocol stack also features the 6LoWPAN adaptation layer, whose main task comprises the fragmentation of IPv6 packets larger than the 802.15.4 maximum frame size. This section reviews necessary background, on TCP, 6LoWPAN fragmentation, and LWB.

2.1 Transmission Control Protocol (TCP)

TCP provides reliability and guaranteed delivery order over the underlying best-effort IP network. Towards this end, TCP hosts first establish a connection via a three-way handshake, and then start sending application data in several segments, each with a sequence number. End-to-end acknowledgments and retransmissions provide reliability. Because of resource constraints, most TCP implementations for LWN devices are simplified, and typically omit features such as sliding window and congestion control. For example, in uIP, the TCP/IP stack we use, TCP has only a single segment in flight which reduces the maximum achievable throughput [9].

2.2 6LoWPAN Packet Fragmentation

Low power CPS such as in [6] have deployed radio transceivers that comply with the IEEE 802.15.4 [22] standard, and feature a maximum maximum transmission unit (MTU) of 127 bytes. As IPv6 requires support for packet sizes much larger than the MTU for IEEE 802.15.4 compliant radios, economizing on packet header space is necessary. To this end, 6LoWPAN is
used as an adaptation layer, that features a header compression mechanism for IPv6 packets. In addition, 6LoWPAN also includes fragmentation of large IPv6 packets that do not fit into maximally-sized single IEEE 802.15.4 frame. Unlike traditional IPv6 fragmentation where each fragment carries IPv6 headers, only the first fragment carries IPv6 headers in 6LoWPAN fragmentation. 6LoWPAN fragmentation effectively reduces the overhead of carrying IPv6 headers, leaving more room for data from upper layers that includes also the TCP header. In our experiments, we have implemented 6LoWPAN packet fragmentation to increase the TCP goodput.

2.3 Low-Power Wireless Bus and Glossy Floods

The low-power wireless bus (LWB) uses scheduled slots to control network-wide Glossy floods [18]. Glossy floods use the fact that simultaneous transmissions of the same packet interfere constructively at a receiver to achieve very fast and reliable network floods. In addition, nodes in Glossy retransmit every data packet several times (we use 3 transmissions in this paper) to further increase robustness.

In LWB, time is cut into communication rounds, which in turn are divided in slots. Every slot consists in a number (3 in our case) of repeated floods. The schedule of slots for each communication round is prepared by a dedicated controller node called host. Figure 1 depicts the composition of slots in a LWB round. Each round begins with a slot for schedule that specifies round period \( T \), the time at the host, data slots and contention slots.

Nodes use data slots for sending/receiving application data. The assignment of a data slot to a node indicates that the node becomes the initiator of a Glossy flood while other nodes become receivers of the Glossy flood. In this way, all other nodes receive what the initiator node has sent and then can filter the data locally based on interest. By operating fully statelessly, LWB offers highly reliable multi-hop communication even in mobile settings.

A schedule may have contention slots during which any node can send its demand for slots to the host. In contention slots, packet collisions are possible. However, one packet typically reaches the host due to capture effect [28]. The host acknowledges the reception of information of demand for data to the nodes in the next round. In order to speed up the process, this information is also piggybacked in the application data to be sent in data slots as necessary.

At the end of each round, the host computes a new schedule for the next round based on the demand of the nodes. The slot allocation for a schedule is dictated by the scheduling policy at the host and can be based on factors such as throughput, latency and energy efficiency. In our work, we design TCP-aware schedulers for LWB.
Figure 1: Composition of slots in a LWB round. Each round begins with a slot for the schedule. There can be several data slots and few contention slots.

We can define the round period $T$ as in Equation 1 where $n$ is the number of data slots and $m$ is the number of contention slots. The variables $T_{SCHED}$, $T_{GAP}$, $T_{DATA}$, $T_{CONT}$ and $T_{COMP}$ correspond to the timing components shown in Figure 1 and they collectively determine the round period $T$.

$$T = T_{SCHED} + (m + n)T_{GAP} + nT_{DATA} + mT_{CONT} + T_{COMP}$$ (1)

3 LWB as a Link Layer Protocol for TCP/IP

In this section, we introduce a scheduler for the LWB protocol which facilitates a seamless operation of an IPv6-based TCP/IP stack over the low-power wireless network. In what follows, we describe the protocol stack architecture, and elaborate on two specific scheduling policies for allocating time slots for TCP/IP connections. We introduce two LWB scheduling algorithms tailored for TCP traffic. For brevity, we denote the composition of the TCP/IP stack and LWB as $ToL$ (TCP/IP over LWB).

3.1 Protocol Stack

The protocol stack of $ToL$ differs from conventional LWN TCP/IP stacks since LWB provides multi-hop communication, medium access control (MAC) and radio duty cycling at the same layer. Figure 2 compares the protocols used in the conventional LWN TCP/IP and our $ToL$ stack with the TCP/IP reference model.

The link layer of the conventional LWN stack is composed of the IEEE 802.15.4 physical signaling standard, medium access control and radio duty-cycling. The IEEE 802.15.4 standard uses carrier sense multiple access
(CSMA) as the MAC protocol. The radio duty-cycling protocol often depends on the LWN operating system. For example, the Contiki OS uses ContikiMAC [10] as the default radio duty cycling protocol. In contrast to this composition, LWB functions as both MAC and radio duty cycling protocols in the ToL protocol stack. The LWB schedule determines the initiator of a Glossy flood and this way performs access control of the radio medium by scheduling Glossy floods. Each LWB slot corresponds to a distinct Glossy flood. Therefore, the radio duty cycle depends on the number of slots in a round making LWB the radio duty cycling protocol.

In the conventional LWN stack, IP datagram handling, multi-hop routing and compressing IP headers are network layer tasks. The Internet Engineering Task Force (IETF) has standardized RPL as the routing protocol for multi-hop routing in LWNs. RPL is implemented in most LWN operating systems such as Contiki and TinyOS [24]. Unlike conventional hop-by-hop packet communication, the packet communication in LWB is end-to-end since it is based on Glossy floods. Therefore, our ToL stack does not require an explicit protocol for routing IP datagrams over multiple hops.

The protocols in both the transport and the application layers of the conventional LWN stack do not differ from the protocols of the ToL stack. Therefore, the applications that run on the conventional LWN stack can run on the ToL stack without or very little modifications.

We connect the ToL network with external IP networks via a border router (BR). We select the LWB host node as the interface to the ToL network at the border router since the host node can schedule data slots according to the demand from the external IP networks. Unlike other nodes in the ToL network in which the TCP/IP stack processes IP datagrams, the
host does not have an active TCP/IP stack. Instead, the host simply sends all IP datagrams to the BR and the BR takes routing decisions accordingly to forward IP packets to the external IP network. The host node merely performs time-critical LWB functions such as starting and stopping Glossy communication.

We offload the schedule computation to the BR for mainly two reasons. Firstly, the BR receives all communicated IP datagrams from the ToL network as well from the external IP network, and therefore, has a global knowledge of TCP connections involving the ToL network. Moreover, the BR is not resource constrained when compared to the nodes in ToL network, and can therefore conveniently bear the scheduling overhead.

3.2 Multiplexing and Buffering IP Datagrams

In our system, we multiplex IP datagrams from all applications in one LWB stream, with an objective of keeping the protocol stack lean and simple. We defer the case of maintaining multiple LWB streams for different applications as future work. Within our design, nodes send LWB stream add requests to the LWB scheduler on demand and the scheduler uses a simple slot recycling method that is based on the number of consecutively unused data slots to remove inactive streams. LWB uses one queue for IP datagrams at each node until they are actually transmitted, i.e., when data slots are available for the node.

3.3 Scheduling Data Slots

Scheduling data slots within an LWB round directly impacts a node’s energy consumption, as its radio is turned on during a data slot. In mobile CPSs, however, actuation often consumes several hundred orders of magnitude more energy than a typical low-power radio. For example, an electric motor of a popular areal drone platform, Parrot AR.Drone consumes 15 W \(^2\) while the power consumption of a widely used low-power radio, CC2420 is only 62.64 mW (17.4 mA current draw at a supply voltage of 3.6 V \(^1\)) when transmitting at the maximum output power. Therefore, the relative power consumption of the radio is almost negligible and we do not need to consider duty-cycling the radio in our scheduling strategies. Instead, we aim for maximum throughput even with mobile systems and hence use all possible data slots in a LWB round for communication.

3.3.1 Naive Scheduler

As a baseline, we define a simple scheduler, the naive scheduler, which simply allocates all possible data slots for the nodes uniformly in a round-robin
Figure 3: Timeline showing how data slots are allocated when establishing a TCP connection between a TCP client (node 1) and a server (node 2). The naive scheduler allocates slots uniformly between the client and server, in a round-robin manner. To make the illustration simple, we set the LWB round period to one second.

manner. Unlike the original LWB data collection scheduler where data slots for a given node are allocated consecutively [17], the naive scheduler allocates data slots evenly throughout the entire data slot space.

Figure 3 shows how the naive scheduler allocates data slots when establishing a TCP connection between two Internet hosts. In this example, both Internet hosts belong to the low power wireless network, and use the ToL stack. Assume, the client (node 1) initiates the connection at a time between $t$ and $t+1$. Initially, the schedule does not contain data slots for any of the nodes. Therefore, the client uses the contention slot to send a LWB stream_add request to indicate its interest to send data at the time $t+1$. In the next round (at $t+2$), the scheduler allocates all possible data slots for the client. However, the client uses only the first data slot in the round to send the TCP SYN packet and waits until it receives the TCP SYN-ACK packet. The server (node 2) receives the TCP SYN packet from the client in the same round. As the server does not have any data slots allocated in the schedule, it sends an LWB stream_add request during the contention slot at the end of the round. At $t+3$, the scheduler allocates data slots for both client and server in a way that every other slot belongs to the client or the server. The server uses the second data slot of the round to send the TCP SYN-ACK and the client sends the TCP ACK in the subsequent slot which completes the TCP handshake. After the completion of the TCP handshake, both client and server use data slots to send application data and TCP acknowledgments.

In Algorithm 1, we outline the naive scheduler. The procedure
Algorithm 1: Naive scheduling algorithm

Input : $S$ is a list of available streams
        $B$ is the backlog, a list of associations between data slots and streams

Output : Schedule is a list of data slots

1. $n_{PossibleSlots} \leftarrow ComputePossibleSlots()$
2. $n_{AllocatedSlots} \leftarrow 0$
3. while not $IsEmpty(B)$ and $n_{AllocatedSlots} < n_{PossibleSlots}$ do
4.     $d \leftarrow RemoveFirst(B)$
5.     $Append(Schedule, d)$
6.     $n_{AllocatedSlots} \leftarrow n_{AllocatedSlots} + 1$
7. end
8. foreach stream $s \in S$ do
9.     if $n_{AllocatedSlots} < n_{PossibleSlots}$ then
10.        $d \leftarrow MakeSlot(s)$
11.        $Append(Schedule, d)$
12.        $n_{AllocatedSlots} \leftarrow n_{AllocatedSlots} + 1$
13.     else
14.        $Append(B, d)$
15.     end
16. end

$ComputePossibleSlots()$ computes the maximum number of data slots that can be allocated in a round ($n$ in Equation 1). When allocating data slots, the scheduler repeatedly iterates all available LWB streams such that a data slot is added to the schedule corresponds to each stream. The scheduler maintains a backlog for the data slots that cannot be included in the current schedule due to unavailability of space for more data slots. The backlog is simply an ordered list of associations between data slots and the streams. In the next round, the scheduler first allocates data slots from associations in the backlog prior to allocate slots by iterating the streams (lines 3–7 in Algorithm 1).

3.3.2 Max-queue-length Scheduler

The naive scheduler, by allocating slots fairly among all nodes, achieves sub-optimal goodput when nodes have uneven traffic load. Table 1 shows an example where node 2 transmits continuously and node 1 merely acknowledges the data it receives. We show increasing TCP payload per segment from 70 to 370 bytes. As the payload increases, the segment becomes fragmented up to 4 fragments, resulting in node 2 sending significantly more traffic than node 1. The naive scheduler results in unused slots between every consecutive fragment.

To address this shortcoming, we extend the naive scheduler and allocate multiple consecutive data slots based on the demand at the nodes. We use the queue length of a node to represent a node’s demand for data slots. In each
Table 1: Usage of slots when sending and acknowledging a TCP data segment when the naive scheduler is used. 1 represents an unused data slot.

<table>
<thead>
<tr>
<th>TCP payload / # fragments</th>
<th>Slot Usage</th>
<th>Segments per sec.</th>
<th>Goodput (B/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70 B / 1</td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>170 B / 2</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>270 B / 3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>370 B / 4</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Arrangement of slots when sending and acknowledging a TCP data segment when the max-queue length scheduler is used.

<table>
<thead>
<tr>
<th>TCP payload / # fragments</th>
<th>Max queue Node1; Node2</th>
<th>Slot Usage</th>
<th>Segments per sec.</th>
<th>Goodput (B/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>170 B / 2</td>
<td>1; 2</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>270 B / 3</td>
<td>1; 3</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>370 B / 4</td>
<td>1; 4</td>
<td></td>
<td></td>
<td>4.8</td>
</tr>
</tbody>
</table>

data slot, the node that owns the data slot and hence transmits, announces the queue length to the host node in the LWB header. The max-queue-length scheduler allocates data slots for the streams (nodes) based on the windowed average of the maximum queue length. We use a window to smooth out temporary queue length changes caused by TCP retransmissions.

Table 2 shows the schedules obtained max-queue-length in the same scenarios as in Table 1. All fragments are now sent in consecutive slots, supporting fragmented traffic at full speed.

We define the number of data slots assigned by the max-queue-length scheduler at the $i^{th}$ round for the stream $k$ as $k_i$ as follows where $W$ represents the window in terms of number of rounds and $l_{i-w}$ represents the maximum queue length at the $(i-w)^{th}$ round. The $k_i$ has an upper bound which is
Algorithm 2: Max-queue length scheduling algorithm

Input: $S$ is a list of available streams
       $B$ is the backlog, a list of associations between data slots and streams

Output: Schedule is a list of data slots

1. $nPossibleSlots \leftarrow ComputePossibleSlots()$
2. $nAllocatedSlots \leftarrow 0$
3. while not $IsEmpty(B)$ and $nAllocatedSlots < nPossibleSlots$ do
   4. $d \leftarrow RemoveFirst(B)$
   5. Append(Schedule, $d$)
   6. $nAllocatedSlots \leftarrow nAllocatedSlots + 1$
   7. end
8. foreach stream $s \in S$ do
   9. $avg \leftarrow GetAvgQueueLength(s)$
   10. $d \leftarrow MakeSlot(s)$
   11. for $i := 0$ to $avg$ step 1 do
    12.     if $nAllocatedSlots < nPossibleSlots$ then
    13.         Append(Schedule, $d$)
    14.         $nAllocatedSlots \leftarrow nAllocatedSlots + 1$
    15.     else
    16.         Append($B$, $d$)
    17.     end
   18. end
19. end

the maximum number of data slots in a round.

$$k_i = \max \left( \frac{\sum_{w=1}^{W} l_{i-w}}{W}, 1 \right)$$ (2)

In Algorithm 2, we extend Algorithm 1 to outline the max-queue length scheduler. Similar to the naive scheduler, the max-queue length scheduler uses a backlog of data slots and iterates all available streams. However, data slots are allocated consecutively based on the average maximum length of the queue of each stream.

4 Implementation

We integrate Contiki’s LWB implementation with the uIPv6 stack. Note that uIP has only a single segment in flight which reduces the maximum achievable TCP throughput [9].

4.1 Timing Considerations

For proper operation of LWB, our system must adhere to the timing constraints imposed by the lower parts of the system. We implement the ToL
Table 3: Values of the timing components in equation 1.

<table>
<thead>
<tr>
<th>Timing component</th>
<th>$T$</th>
<th>$T_{SCHED}$</th>
<th>$T_{GAP}$</th>
<th>$T_{DATA}$</th>
<th>$T_{CONT}$</th>
<th>$T_{COMP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value (ms)</td>
<td>1000</td>
<td>30</td>
<td>4</td>
<td>33</td>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

stack on the the *Tmote Sky* that features an IEEE 802.15.4-compliant radio. The resulting timing parameters are detailed in Table 3. In order to reduce the delay imposed by frequent schedule dissemination and allow new LWB streams to be added quickly, we limit LWB round period ($T$) to 1000 ms. We choose the active duration of a *Glossy* flood in a data slot ($T_{DATA}$) to be 33 ms which leaves room for three *Glossy* transmissions (see Section 2).

4.2 Communication with the Border Router

In order to facilitate access from the external IP network, the host node requires a fast means of connection with the BR. Importantly, host communicating with the BR should not interfere LWB operations. If host communication with the BR interferes LWB operations, one could extend $T_{GAP}$ duration necessarily. However, as seen from Equation 1, this results in a reduction of the number of available data slots in an LWB round and hence reduces the achievable throughput. Therefore, we focus our evaluation on end-to-end communication within the *ToL* network. For the same reason, we set $T_{COMP}$ to 50 ms since we use the serial interface of the nodes for logging purposes.

4.3 Integration in the uIPv6 Stack

The default routing protocol for LWNs, RPL is an integrated part of the uIPv6 stack. LWB, however, inherently includes multi-hop communication and hence the nodes in a *ToL* network do not operate as routers. Therefore, we remove RPL related features from uIPv6. We use stateless address autoconfiguration [32] for generating unique IPv6 addresses for the nodes. The *Tmote Sky* platform has a 48-bit unique serial number that we use to generate an unique IPv6 address with a given prefix for each node.

After the uIPv6 stack has produced an IP datagram from the application message, the 6LoWPAN adaptation layer compresses and fragments it accordingly. Then, the resulting packets are queued at LWB for transmission. On the receiving side, at the end of a data slot, LWB delivers the received data to the 6LoWPAN adaptation layer for decompressing the IPv6 headers and reassembling fragments. After decompressing and reassembling, the
6LoWPAN adaptation layer delivers the IP datagram to the uIPv6 stack which eventually hands it to the application.

5 Evaluation

We experimentally evaluate the end-to-end TCP performance of ToL. We conduct experiments on a testbed consisting of 15 Tmote sky nodes that are located in our office corridor. The topology of the network is shown in Figure 4. For experiments involving mobility, we use a robot that follows a line [31] and carries node 13. The robot moves between positions A and D which are 30 m apart at the speed of approximately 300 mm/s. We set the nodes’ transmission power to -6 dBm and use channel 15 of the IEEE 802.15.4 band since it is the least interfered channel in the testbed. When all nodes are connected, the network has a diameter of three hops.

In our experiments, we run TCP servers on different nodes in the testbed including the mobile node. When a TCP client is connected to a particular server, the client sends a request for data and the server sends fixed size application messages back-to-back. Each message includes a sequence number. At the client, we collect various statistics such as the inter-arrival time of two back-to-back TCP application messages (segment interval) and the number of retransmissions. Unless explicitly mentioned, we always use node 9 as the LWB host in our experiments.
Figure 5: Application goodput comparison for various static positions and with mobility. Thanks to LWB’s stateless operation, mobility has no significant effect on the goodput.

5.1 Naive Scheduler

5.1.1 Without 6LoWPAN Fragmentation

First, we measure the end-to-end TCP performance without fragmentation. We run experiments in both static and mobile settings. The static scenario serves as a baseline since the nodes do not experience frequent connectivity changes over the time. We run a TCP server on node 13 and a client on node 2. The server sends 70-byte large application packets back-to-back to the client when connected. After 6LoWPAN header compression, the size of the entire IPv6 datagram is 110 bytes. In the static scenario, we place node 13 at different places along the path of the robot depicted by the letters A, B, C, and D in Figure 4. In the mobile scenario, the robot continuously moves along the line between A and B.

Figure 5 shows the measured TCP goodput in all scenarios. The comparison shows that the movement of the robot does not cause any considerable changes to the goodput. We attribute this to the underlying communication primitive of LWB, Glossy, which operates without keeping any topology state. Therefore, variations of the topology caused by the position or movement of the robot does not affect LWB. As a result, the movement of the robot does not negatively impact the performance.
Figure 6: Comparison of the segment interval for various positions of the robot and while the robot is moving. The segment interval is below 80 ms for 85% of the time for all positions of the robot.

In order to observe the responsiveness of the TCP connection, we plot the segment inter-arrival time (called segment interval for brevity) as shown in Figure 6. The figure shows that the segment interval is below 80 ms for 85% of the time for all positions of the robot even when the robot is moving. We explain this result by looking at the minimum segment interval. The segment interval is minimum when data slots of both TCP data and acknowledgment segments fall into the same LWB round while it is maximum when data slots of TCP data and acknowledgment segments are spread over two rounds. The figure shows that in 85% of the time data slots of both TCP data and acknowledgment segments fall into the same round consecutively.
6LoWPAN fragmentation reduces the per IPv6 packet overhead when sending large packets since only the first fragment contains headers, and since fewer ACKs need to be sent. Therefore, we measure the TCP goodput for different application message (payload) sizes which lead to a different number of 6LoWPAN fragments. Figure 7 depicts the results for scenarios where the robot is moving.

Figure 7: Comparison of application goodput for various number of 6LoWPAN fragments while the robot is moving, with 1 to 4 fragments per segment.

5.1.2 With 6LoWPAN Fragmentation

Figure 8: Comparison of the segment intervals for various number of 6LoWPAN fragments while the robot is moving.

6LoWPAN fragmentation reduces the per IPv6 packet overhead when sending large packets since only the first fragment contains headers, and since fewer ACKs need to be sent. Therefore, we measure the TCP goodput for different application message (payload) sizes which lead to a different number of 6LoWPAN fragments. Figure 7 depicts the results for scenarios where the robot is moving.
robot is moving. The payload sizes of 70, 170, 270 and 370 bytes lead to no fragmentation, or 2, 3 and 4 6LoWPAN fragments respectively. The figure shows a noticeable increase in goodput when fragmentation happens. However, one can see that there is no significant improvement beyond two fragments. We attribute this to the fair slot allocation among the nodes, unable to handle individual nodes traffic load, as discussed in Section 3.3.2.

We also plot the segment interval for various number of 6LoWPAN fragments while the robot is moving in Figure 8. The figure shows that the segment interval increases when the number of 6LoWPAN fragments is higher. This, again, is a consequence of the naive scheduler allocating interleaved slots evenly among nodes.

5.1.3 Concurrent TCP Connections

We stress our system in scenarios with multiple concurrent TCP connections. Figure 9 shows the cumulative goodput for all five TCP connections. Connections 2, 3, 4, and 5 start after 60, 120, 180 and 240 seconds respectively after connection 1 has started. The figure shows that all TCP connections share the total bandwidth evenly. The figure also shows a temporary drop in goodput each time a new connection starts. This is attributed to the TCP handshake overhead. Moreover, nodes send LWB stream add requests at the end of the schedule as mentioned in Section 3.3.1. As a result, TCP
handshakes takes 2-3 LWB rounds (2-3 seconds in our implementation) to complete.

5.2 Max-queue-length Scheduler

![Graph comparing TCP goodput for naive vs. max-queue-length schedulers](image)

Figure 10: Comparison of TCP goodput for naive vs. max-queue-length schedulers in the mobile scenario, for 2 to 4 fragments.

We evaluate our max-queue-length scheduler in the mobile scenario. When 6LoWPAN fragmentation does not happen, the max-queue-length scheduler is equivalent to the naive scheduler since the maximum length of the queue at the nodes (client and server) is consistently one. Therefore, we consider only the scenarios where 6LoWPAN fragmentation happens.

Figure 10 compares the performance of the naive and max-queue-length schedulers. With the max-queue-length scheduler, the TCP goodput increases with the number of fragments, whereas the naive scheduler has constant performance for 2 to 4 fragments.

Unlike the naive scheduler, the max-queue length scheduler allocates data slots consecutively based on the maximum queue length at the nodes (see Section 3.3.2). This allows to fully exploit large fragmented segments as a means to reduce per-segment overhead. As a result, the goodput increases with the segment size, even beyond 2 fragments per segment.
Figure 11: Comparison of the segment intervals for various number of 6LoWPAN fragments for naive and max-queue length schedulers.

We also compare the segment intervals for various number of 6LoWPAN fragments for the naive and max-queue length schedulers as Figure 11 depicts. The figure shows that the segment interval also decreases when the max-queue length scheduler is used. We explain the decrease in the segment interval by the reduction of the number of unused data slots when using the max-queue length scheduler. Moreover, the reduction of the number of unused data slots also increases the number of times that the data slots of both TCP data and acknowledgment segments fall into the same LWB round.

5.3 Comparison with RPL

We compare the performance of TCP on ToL against the conventional low-power IP stack, with RPL as routing protocol (TCP with RPL is denoted by TwR). RPL is not designed with mobility in mind, nevertheless we consider it to understand how the standardized routing protocol performs in mobile scenarios.

In TwR experiments, we use the default configurations of the RPL implementation that is available in Contiki. Moreover, we select different TCP payload sizes and different combinations of nodes for client, server and LWB host/RPL root. We let the TwR network stabilize for 5 minutes before we start the TCP connection. As in previous ToL experiments, node 13 is attached to the mobile robot. In static scenarios, TwR’s performance is comparable to ToL’s.

Figure 12 shows the TCP goodput for both ToL and TwR in mobile scenarios. With ToL, the performance is the same independent of the nodes
Figure 12: Comparison of TCP goodput of both ToL and TwR for different TCP payload sizes (one and two 6LoWPAN fragments). We select different combinations of nodes for client, server and LWB host/RPL root in our experiments. The tuple \([x, y, z]\) represents a combination of nodes where \(x\) is the LWB host (or RPL root) and \(y\) and \(z\) are TCP client and server respectively.

involved. As Glossy communication is topology invariant, ToL delivers roughly similar TCP goodput in all cases. Slight differences are caused by the exact positions of the nodes which impact the quality of the wireless channels. In RPL, nodes build a tree-like topology anchored at the root, using a routing metric such as ETX (used in our experiments). All traffic intended for the root is sent upwards, i.e., to the parent. Traffic to other nodes is sent downwards, i.e. to a child, using the reverse links. Any-to-any routing is supported by sending first upwards until a common ancestor, and then downwards to the final destination.

Figure 12 also shows that the TCP performance of TwR is lower when 130-byte payload (two 6LoWPAN fragments) is used compared to 30-byte payload (one 6LoWPAN fragment). When sending multiple fragments, the loss of a single fragment results in the loss of the whole IP datagram and hence the entire TCP data segment. As the movement of the robot changes the links frequently, losses of entire TCP data segments are more frequent resulting in lower TCP goodput.
6 Related Work

Our work is built on LWB [17]. While the authors have shown the performance of LWB in mobile scenarios, they have not considered throughput as a metric. Furthermore, our work brings LWB to the low-power IPv6 stack, and addresses TCP-aware scheduling strategies.

Duquennoy et al. propose a burst forwarding mechanism for low-power radios that uses TCP to achieve high throughput [13]. Burst forwarding uses full-scale TCP/IP implementations at the end nodes and is hence not hampered by the restriction of having only a single segment in flight. Therefore, it achieves a considerably higher TCP throughput than our system, but is not applicable in mobile scenarios. Dunkels et al. propose a series of extensions to improve the TCP performance across multiple LWN hops based on caching TCP segments on intermediate nodes and locally regenerating TCP acknowledgments [12]. Hurni et al. [23] implemented a variant of distributed TCP caching [5]. Their approach targets static scenarios and hence their evaluation does not cover mobile scenarios. Moreover, our system is able to achieve considerably higher TCP throughput than theirs.

Though it is not particularly focused on improving TCP performance, Lee et al. study on improving performance of RPL under mobility on vehicular networks [27]. Moreover, Ben Saad et al. and Korbi et al. also investigate on improving performance of RPL in mobile scenarios based on simulations [4][25] whereas we present results based on experiments with real hardware and using LWB as the link layer. Improving TCP performance over mobile Wi-Fi and cellular networks is studied extensively in the literature [14][15].

At the other end of the spectrum, reactive routing is a traditional approach to routing in mobile scenarios with e.g., AODV [30], LOADng [16], or RRPL [26]. In reactive routing protocols, routes are build on-demand via network flooding and then used for data transfer. LOADng and RRPL are designed as replacement and extension of RPL in low-power IPv6 networks. Existing work in low-power reactive routing focus on best effort rather than reliable communication. Enabling continuous TCP streaming over such protocol is a challenge, which to the best of our knowledge has not been addressed yet.

7 Conclusion

Cyber-physical systems benefit from using Internet protocols since this way the components such as sensors, computational entities and actuators become interoperable with other systems. In this paper, we focus on enabling TCP for mobile cyber-physical systems by improving its performance. We use the Low-power Wireless Bus as link layer, integrate it with the uIP stack and
design TCP-aware schedulers. Our evaluation with real world experiments shows that we achieve a high TCP throughput both in static and mobile scenarios.

Acknowledgements

We thank Federico Ferrari and Marco Zimmerling for the LWB implementation and the help with FlockLab and Frederik Hermans for the help with the mobile testbed. This work was partly funded by .SE, the Internet Infrastructure Foundation.

References


