HIGHLY RELIABLE, LOW-LATENCY COMMUNICATION IN LOW-POWER WIRELESS NETWORKS

A dissertation submitted to
TECHNISCHE UNIVERSITÄT DRESDEN
FACULTY OF COMPUTER SCIENCE

for the degree of
Doktor-Ingenieur (Dr.-Ing.)

presented by
MARTINA BRACHMANN, M.SC.
born on April 14, 1987 in Dresden, Germany

accepted on the recommendation of
Examiner:
Prof. Dr. Silvia Santini (Università della Svizzera italiana, Switzerland)
Co-examiner:
Prof. Dr. Thiemo Voigt (Uppsala University, Sweden)

Dresden, November 9, 2018

DECLARATION

Hiermit versichere ich, dass ich die vorliegende Dissertation zur Erlangung des akademischen Grades "Doktor-Ingenieur (Dr.-Ing.)" mit dem Titel *Highly reliable, low-latency communication in low-power wireless networks* selbstständig und ohne unzulässige Hilfe Dritter verfasst habe. Es wurden keine anderen als die in der Arbeit angegebenen Hilfsmittel und Quellen benutzt. Die wörtlichen und sinngemäß übernommenen Zitate habe ich als solche kenntlich gemacht.

Dresden, November 9, 2018

Martina Brachmann, M.Sc.

ABSTRACT

Low-power wireless networks consist of spatially distributed, resource-constrained devices – also referred to as *nodes* – that are typically equipped with integrated or external sensors and actuators. Nodes communicate with each other using wireless transceivers, and thus, relay data – e. g., collected sensor values or commands for actuators – cooperatively through the network. This way, low-power wireless networks can support a plethora of different applications, including, e. g., monitoring the air quality in urban areas or controlling the heating, ventilation and cooling of large buildings. The use of wireless communication in such monitoring and actuating applications allows for a higher flexibility and ease of deployment – and thus, overall lower costs – compared to wired solutions. However, wireless communication is notoriously error-prone. Message losses happen often and unpredictably, making it challenging to support applications requiring both *high reliability* and *low latency*. Highly reliable, low-latency communication – along with high energy-efficiency – are, however, key requirements to support several important application scenarios and most notably the open-/closed-loop control functions found in e. g., industry and factory automation applications.

Communication protocols that rely on *synchronous transmissions* have been shown to be able to overcome this limitation. These protocols depart from traditional single-link transmissions and do not attempt to avoid concurrent transmissions from different nodes to prevent collisions. On the contrary, they make nodes send the same message at the same time over several paths. Phenomena like *constructive interference* and *capture* then ensure that messages are received correctly with high probability. While many approaches relying on synchronous transmissions have been presented in the literature, two important aspects received only little consideration: (i) reliable operation in harsh environments and (ii) support for event-based data traffic. This thesis addresses these two open challenges and proposes novel communication protocols to overcome them.

When low-power wireless networks are deployed in harsh environments, nodes are exposed to large variations of, e. g., temperature and humidity. It is known from the literature that under these conditions the link quality, and thus, the reliability of communication protocols decreases significantly. This is because environmental parameters may negatively affect the functioning of the electronic equipment of a node and in particular of its clock. Protocols relying on synchronous transmissions, however, require a precise and consistent operation of the nodes' clock to achieve constructive interferences with high probability. Through both simulation and laboratory experiments, we show that even little offsets across the nodes' clocks significantly decrease this probability, especially when the nodes

are exposed to different temperatures. To tackle this problem, we present Flock, a novel mechanism to compensate clock frequency variations across synchronously transmitting nodes. Our results confirm that Flock allows protocols to achieve constructive interference in 98 % of the message transmissions even when nodes are exposed to temperature differences of 30 °C and without introducing any message overhead.

While most protocols based on synchronous transmissions are tailored for periodic communication, aperiodic, event-based traffic occurs frequently in low-power wireless networks. Thereby, the data to transmit is generated at unknown, non-regular time instants, and must be relayed immediately to a central server. To be able to timely detect and handle communication requests, nodes must, nonetheless, regularly switch their radios on and check the channel for incoming transmissions. While reducing the frequency and duration of this channel sampling procedure is mandatory to reduce energy consumption, frequent channel checks are needed to ensure a timely detection and handling of data traffic. To cope with this trade-off, we present Whisper, a communication primitive that allows nodes to efficiently propagate small amounts of data. A key advantage of Whisper is that – compared to existing solutions – it significantly reduces the energy consumption due to efficient channel sampling. In particular, our testbed experiments show that when no data traffic must be relayed, Whisper spends 30 % less time in channel sampling than its closest competitor. At the same time, Whisper can disseminate data twice as fast and with no loss in reliability.

Whisper's ability to support event-based data traffic is particularly useful when Internet protocols like UDP, TCP or CoAP – which serve applications that may generate data aperiodically – are used on top of a routing substrate. However, the direct utilization of such protocols on top of Whisper is cumbersome and inefficient. The payload size of standard Internet protocols can reach hundreds of bytes that are exchanged between two nodes in a one-to-one communication manner. However, Whisper floods the entire network at each data exchange, thus, causing unnecessary overhead. To overcome this issue, we introduce LaneFlood, a communication protocol that establishes a restricted number of paths – collectively called *lane* – between two communicating nodes, and thus, involves in the data distribution only the strictly necessary amount of nodes. Our testbed experiments show that LaneFlood reduces the overall energy consumption considerably compared to state-of-the-art approaches and without any loss in reliability.

In summary, the three contributions of this thesis provide a step further towards the efficient usage of low-power wireless networks in real scenarios. This is achieved by first ensuring high performance in extreme environmental conditions and then by providing solutions that enable energy-efficient, highly reliable, low-latency communication of event-based data traffic.

KURZFASSUNG

Energiearme drahtlose Netze bestehen aus räumlich verteilten, ressourcenbeschränkten Geräten – auch als *Knoten* bezeichnet – die typischerweise mit integrierten oder externen Sensoren und Aktoren ausgestattet sind. Knoten kommunizieren miteinander über drahtlose Sendeempfänger und übertragen somit Daten – z. B. gesammelte Sensorwerte oder Befehle für Aktuatoren – kooperativ durch das Netz. Auf diese Weise können drahtlose Netze eine Vielzahl verschiedener Anwendungen unterstützen, z. B. die Überwachung der Luftqualität in städtischen Gebieten oder die Steuerung der Heizung, Lüftung und Kühlung großer Gebäude. Die Nutzung drahtloser Kommunikation ermöglicht eine höhere Flexibilität und einfacheren Einsatz – und damit insgesamt geringere Kosten – im Vergleich zu kabelgebundenen Lösungen. Allerdings ist die drahtlose Kommunikation fehleranfällig. Nachrichtenverluste treten häufig und unvorhersehbar auf, so dass es schwierig ist, Anwendungen zu unterstützen, die gleichzeitig eine *hohe Zuverlässigkeit* und eine *niedrige Latenz* erfordern. Eine zuverlässige Kommunikation mit niedriger Latenz ist – zusammen mit einer hohen Energieeffizienz – jedoch eine Schlüsselvoraussetzung für die Unterstützung diverser Anwendungsszenarien, insbesondere von offenen oder geschlossenen Regelkreisen wie sie in der Industrie- und Fabrikautomatisierung zu finden sind.

Es wurde bereits gezeigt, dass Kommunikationsprotokolle, die auf *synchronen Übertragungen* beruhen, diese Einschränkungen bewältigen können. Diese Protokolle weichen von traditionellen Single-Link-Übertragungen insofern ab, dass sie Kollisionen durch parallele Übertragungen von verschiedenen Knoten nicht verhindern. Im Gegenteil, sie bewirken, dass Knoten gleiche Nachricht gleichzeitig über mehrere Pfade senden. Phänomene wie *konstruktive Interferenz* und *Capture* sorgen dafür, dass Nachrichten mit hoher Wahrscheinlichkeit korrekt empfangen werden. Während in der Literatur viele Ansätze beschrieben wurden, die auf synchronen Übertragungen beruhen, wurden zwei wichtige Aspekte bisher nur wenig berücksichtigt: (i) der zuverlässige Einsatz in harschen Umgebungen und (ii) die Unterstützung von ereignisbasiertem Datenverkehr. Diese Arbeit befasst sich mit diesen beiden Herausforderungen und schlägt neuartige Kommunikationsprotokolle vor.

Wenn drahtlose Netze in harschen Umgebungen eingesetzt werden, sind die Knoten großen Schwankungen von Temperatur und Feuchtigkeit ausgesetzt. Aus der Literatur ist bekannt, dass in dieser Umgebung die Verbindungsqualität und damit die Zuverlässigkeit von Kommunikationsprotokollen signifikant abnimmt. Ursache hierfür ist, dass Umweltparameter die Funktionsweise des elektronischen Equipments eines Knotens und insbesondere dessen Clock negativ beeinflussen können. Protokolle basierend auf synchroner Übertragung benötigen jedoch eine präzise und konsistent betriebene Clock, um mit hoher

Wahrscheinlichkeit konstruktive Interferenzen zu erzielen. Sowohl durch Simulations- als auch durch Laborexperimente zeigen wir, dass selbst kleine Clock-Offsets der Knoten diese Wahrscheinlichkeit signifikant verringert. Um dieses Problem zu lösen, präsentieren wir Flock, einen Mechanismus der Abweichungen der Taktfrequenz von synchron sendenden Knoten kompensiert. Unsere Ergebnisse bestätigen, dass mit Flock – ohne Nachrichten-Overhead zu verursachen – in 98 % der Übertragungen konstruktive Interferenzen erzielt wird, selbst wenn Knoten Temperaturunterschieden von 30 °C ausgesetzt sind.

Während die meisten Protokolle, die auf synchronen Übertragungen basieren, für periodische Kommunikation zugeschnitten sind, tritt in drahtlosen Netzen aperiodischer, ereignisbasierter Verkehr häufig auf. Dabei werden zu übertragende Daten zu unbekannten, nicht regulären Zeitpunkten erzeugt, die dann sofort an einen zentralen Server weitergeleitet werden müssen. Um Nachrichten rechtzeitig erkennen und weiterleiten zu können, müssen die Knoten regelmäßig ihre Empfänger einschalten und den Kanal auf eingehende Übertragungen prüfen. Während zur Verringerung des Energieverbrauchs die Häufigkeit und Dauer dieser Kanalabtastung reduziert werden muss, sind häufige Kanalabtastungen erforderlich, um eine rechtzeitige Erkennung der Datenübertragung sicherzustellen. Wir lösen diesen Konflikt mit Whisper, einer Kommunikationsprimitive, welche es Knoten ermöglicht, kleine Datenmengen effizient zu übertragen. Ein wesentlicher Vorteil von Whisper ist, dass im Vergleich zu bestehenden Lösungen der Energieverbrauch durch Kanalabtastungen deutlich reduziert wird. Insbesondere zeigen unsere Testbed-Experimente, dass wenn kein Datenverkehr weitergeleitet werden muss, Whisper 30 % weniger Zeit für die Kanalabtastung benötigt als sein nächster Konkurrent. Gleichzeitig kann Whisper Daten doppelt so schnell und ohne Verlust an Zuverlässigkeit verbreiten.

Whispers Fähigkeit, ereignisbasierten Datenverkehr zu unterstützen, ist besonders nützlich, wenn Internetprotokolle wie UDP, TCP oder CoAP verwendet werden. Die direkte Verwendung solcher Protokolle mit Whisper ist jedoch umständlich und ineffizient. Die Nutzlast von Standard-Internet-Protokollen kann hunderte von Bytes erreichen, die zwischen zwei Knoten – also One-to-One – ausgetauscht werden. Whisper flutet jedoch bei jedem Datenaustausch das gesamte Netz und verursacht so unnötigen Overhead. Um dieses Problem zu lösen, präsentieren wir LaneFlood, ein Kommunikationsprotokoll, das eine beschränkte Anzahl von Pfaden – kollektiv als *Lane* bezeichnet – zwischen zwei kommunizierenden Knoten erzeugt und somit bei der Datenverteilung nur die unbedingt notwendige Menge an Knoten involviert. Unsere Testbed-Experimente zeigen, dass LaneFlood den Gesamtenergieverbrauch im Vergleich zu State-of-the-Art-Ansätzen ohne Verlust an Zuverlässigkeit deutlich reduziert.

Zusammengefasst bieten die drei Beiträge dieser Arbeit einen weiteren Schritt zur effizienten Nutzung von drahtlosen Netzen in realen Szenarien. Dies wird erreicht, indem zuerst eine hohe Performanz unter extremen Umweltbedingungen sichergestellt wird und dann Lösungen bereitgestellt werden, die eine energieeffiziente, zuverlässige und latenzarme Kommunikation von ereignisbasierten Datenverkehr ermöglichen.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
6LoFlood	6LoWPAN for LaneFlood
CAN	Controller Area Network
CIWA	Chinese Industrial Wireless Alliance
CoAP	Constrained Application Protocol
CRC	Cyclic Redundancy Check
CSMA	Carrier-Sense Multiple Access
CSMA/CA	CSMA with Collision Avoidance
CXFS	Concurrent Transmission Forwarder Selection
DAE	Destination Address Encoding
DCO	Digitally Controlled Oscillator
DPE	Destination Port Encoding
DSSS	Direct Sequence Spread Spectrum
FCS	Frame Check Sequence
FDMA	Frequency Division Multiple Access
Flock	On-the-Fly Clock Offset Compensation
HART	Highway Addressable Remote Transducer
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation and Air Conditioning
ICMP	Internet Control Message Protocol
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv4	IP Version 4
IPv6	IP Version 6
ISA	International Society of Automation
ISM	Industrial, Scientific and Medical
LAN	Local Area Network
LR-WPAN	Low-Rate Wireless Personal Area Network
LWB	Low-power Wireless Bus
MAC	Medium Access Control
MCU	Microcontroller Unit

MPDU	MAC Protocol Data Unit
MSK	Minimum Shift Keying
MTU	Maximum Transfer Unit
NOP	No Operation
O-QPSK	Offset-Quadrature Phase Shift Keying
OSI	Open System Interconnection
PN	Pseudo-random Noise
QoS	Quality of Service
RA	Router Advertisement
RAM	Random Access Memory
ROM	Read-Only Memory
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
RS	Router Solicitation
RSS	Received Signal Strength
RTS/CTS	Request to Send / Clear to Send
SA	Short Acknowledgement number
SAE	Source Address Encoding
SFD	Start-of-Frame Delimiter
SLAAC	StateLess Automatic Address Configuration
SMA	SubMiniature Version A
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SPE	Source Port Encoding
SPI	Serial Peripheral Interface
SS	Short Sequence number
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TI	Texas Instruments
TSCH	Timeslotted Channel Hopping
TSMP	Time Synchronized Mesh Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VHT	Virtual High Resolution Time
WIA-PA	Wireless Networks for Industrial Automation – Process Automation
WISA	Wireless Interface for Sensors and Actuators
WSN	Wireless Sensor Network

LIST OF SYMBOLS

Δ_{Td}	Temporal displacement of incoming signals.
$\Delta_{Td \leqslant 0.5\mu s}$	Maximum temporal displacement of incoming signals for achieving constructive interference.
c	Relay counter in Glossy and LaneFlood. It indicates how often a packet has been relayed.
d_{df}	In LaneFlood, distance in hops between destination and the node that has received the current message.
d_{sd}	In LaneFlood, distance in hops between source and destination.
d_{sf}	In LaneFlood, distance in hops between source and the node that has received the current message.
E_{rx}	Number of clock cycles that elapse during T_{rx} with a frequency of f_{DCO}.
E^*_{rx}	Number of clock cycles that actually elapse during T_{rx} with a frequency of f^*_{DCO}.
f_r	Nominal frequency of the radio clock. In this thesis, f_r is set to 8 MHz.
f_{DCO}	Nominal frequency of the DCO clock. In this thesis, f_{DCO} is set to 4 194 304 Hz.
f^*_{DCO}	Actual frequency of the DCO clock.
I	Number of clock cycles and thus, number of instructions that must elaps during T_{sw} with f_{DCO}.
I^*	Number of clock cycles and thus, number of instructions that must actually elaps during T_{sw} with f^*_{DCO}.
k_p	Variable and unpredictable delay at which the MCU detects the SFD signal from the radio. The delay is caused by the unsynchronized running MCU and radio clocks. It is a uniformly distributed random variable in the interval $0 < k_p \leqslant 1$.
N_{tx}	Number of consecutive packet/packlet transmissions before a node turns its radio off.
p	Packlet counter in Whisper. It indicates the packlet a receiver has intercepted.
s	The slack s determines the width of a lane in LaneFlood and thus, the number of nodes that actively participate in the message exchange.
s_i	The integer part of the slack. It declares the nodes that must be within the lane.

s_d	The fractional part of the slack. It declares the nodes that are part of the lane with probability s_d.
T_c	Duration of a LaneFlood communication slot.
T_g	Duriation of the guard time in LaneFlood at the end of a session.
T_r	Duration of a LaneFlood round.
T_s	A LaneFlood session.
T_{sw}	Software delay between the end of a packet reception and the transmission request to the radio. During the T_{sw} the Microcontroller Unit (MCU) is in charge.
T_{rx}	The interval where the SFD pin is active during reception. It spans the reception duration of the length field and the MPDU.
T_{turn}	The rx/tx turnaround time, where the radio switches from receive mode to transmit mode (and vice versa). The IEEE 802.15.4 standard defines the rx/tx turnaround to be at most 192 µs.

CHAPTER 1

INTRODUCTION

Low-power wireless networks consist of resource-constrained, battery-operated devices that are equipped with a wireless transceiver. Such devices are called *nodes* and together with integrated or external sensors and actuators, the nodes jointly perform tasks like the monitoring and control of physical events. With regard to wired solutions, low-power wireless networks have great potential to improve productivity and operational efficiency, e. g., in *"industrial and factory automation, distributed control systems, automotive systems and other kinds of networked embedded systems [116, 184]"* [183]. The advantages of low-power wireless networks lie in the cost- and time-effective installation and maintenance of a high number of wireless nodes [58, 63, 139, 184]. The nodes are cheap, which allows for installing hundreds or thousands of these devices for comprehensive sensing and monitoring of complex systems [58, 151]. The absence of cables facilitates node mobility, e. g., the deployment of the nodes on rotating equipment or freely moving devices. Thus, low-power wireless networks can operate in dangerous and for humans' inaccessible environments [10, 46].

For nodes to operate unattended and reliably for several month or years – as needed by many applications [10, 14, 125, 151] – an energy-efficient operation of the network is mandatory. This implies that communication protocols used in low-power wireless networks must be designed to be energy-efficient. This is because wireless communication is the main cause of energy consumption in these networks [58, 183]. However, the wireless communication channel is inherently unreliable and lossy due to interference, attenuation and unpredictable signal propagation effects. This causes packet losses and delays in packet deliveries. However, applications like open-/closed-loop control rely on timely feedback and thus, have high requirements in *reliability* and *low latency* [58, 63, 139, 183].

Several approaches [36–38, 50, 52, 60, 73, 75, 90, 189, 191] tackle this problem with a technique called *synchronous transmissions*. Using synchronous transmissions, multiple nodes forward identical packets simultaneously to let their packets precisely overlap and collide constructively at the same receiver(s). The constructive collisions significantly increase the likelihood of packets being correctly received. A short radio activity, and thus, the energy-efficient operation of synchronous transmission-based protocols relies on the simultaneous wake-up and the quick and reliable network-wide packet dissemination.

In this thesis, we build upon the promising results of synchronous transmission-based protocols with respect to energy-efficiency, reliability and latency. In particular, this thesis deals with two major aspects that have only received little consideration in this field: reliable operations in harsh environments and the support for event-based data traffic.

1.1 PROBLEM STATEMENT

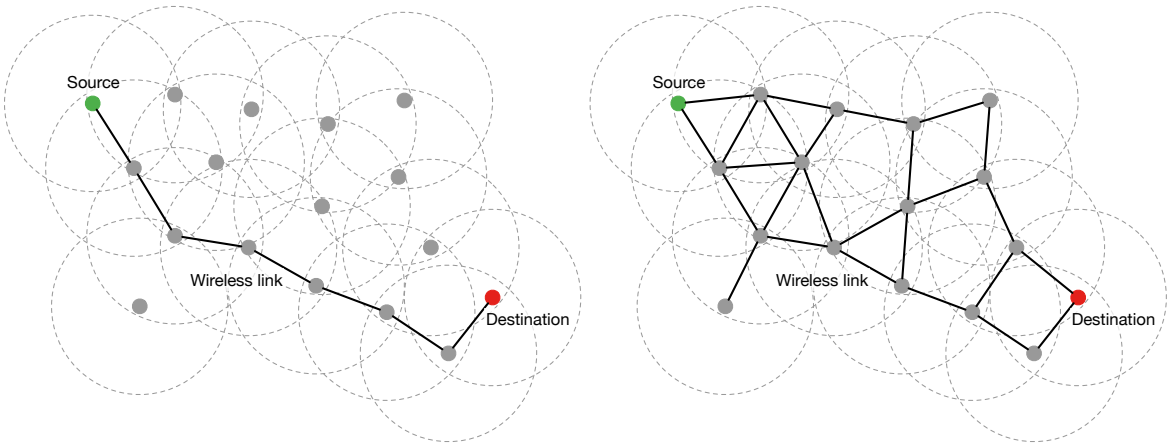
The rationale behind the use of synchronous transmissions is to depart from traditional single-link communication and instead let several nodes transmit and receive identical packets synchronously over multiple paths. Figure 1.1 shows schematically a packet propagation using single link transmissions (Figure 1.1a) and synchronous transmissions (Figure 1.1b) in comparison. As displayed in Figure 1.1a, in single-link-based protocols the nodes forward packets to only one node within their communication range. Using synchronous transmissions, depicted in Figure 1.1b, identical packets are flooded simultaneously to all nodes within one hop. Thus, several replicas of a packet exist. In case of packet loss on one link, the packet can still be disseminated until it is received by all nodes in the network. Letting the nodes transmit the replicated packets at exactly the same time instant – within a *temporal displacement* of sub-microseconds – increases the likelihood that a packet is correctly received despite interference.

Experiments with real nodes in optimal ambient conditions and under high noise interference have shown that synchronous transmissions boost the performance in low-power wireless networks [47, 52, 74, 130, 153]. While these results demonstrate the potential of synchronous transmissions, two important aspects remain unexplored:

- *Reliable operation in harsh environments:* Low-power wireless networks are often deployed in harsh environments, where the nodes are exposed, e. g., to high temperature variations. Related work has shown that temperature has a dramatic effect on the link quality and thus, on the communication reliability in single-link protocols [17, 178]. However, investigations on the temporal displacement of packets from synchronously transmitting nodes have not yet been performed. Therefore, it is unclear how well packets collide constructively in harsh environmental conditions.

- *Support of event-based data traffic:* Existing protocols building upon synchronous transmissions strongly focus on periodic data transfer, and thus, on periodically occurring events that need to be transmitted. In contrast to periodic data transfer is *event-based* communication, where an event, and thus, the data transmission occurs unpredictably. For a timely packet forwarding, the nodes have to check the channel regularly and long enough for incoming data packets. The aspect of reducing the channel check activity to save energy has yet only received little considerations in the context of synchronous transmissions. Another problem occurs in scenarios in which two nodes exchange data in a *one-to-one communication* manner. Realizing such scenario with existing solutions results in flooding the entire network, which consumes unnecessary energy.

In the following, we discuss both aspects in more detail.



(a) Single-link transmissions. *(b)* Synchronous transmissions.

Figure 1.1: Single-link vs. synchronous transmissions. In single-link transmissions, only one node forwards the packet until it reaches its destination. Using synchronous transmissions, a packet is received and retransmitted by all nodes within one hop, and thus, several replicas of a packet exists along different paths. In case of packet loss at one link, the packet can still be forwarded to its destination through other paths. Letting nodes transmit the replicated packet at the same time instant, so that their packets collide constructively, increases the likelihood that the packet is being received at the receiver(s).
The dots in the figures represent nodes and circles indicate the one-hop communication range of the nodes.

1.1.1 *Reliable operation in harsh environments*

Many low-power wireless networks are expected to be deployed in inaccessible or harsh environments where nodes are exposed to dust, vibrations, moister, and extreme heat/cold. Typical example applications range from industrial and factory automation [55, 63, 139] over smart grids [172] to critical infrastructure monitoring [10, 29]. Related work [17, 178] has shown that changes in temperature and humidity has strong influence on the packet reception reliability in traditional single-link networks. For example, *Boano et al.* [17] argue that an increasing temperature reduces a node's communication range due to a lower Signal-to-Noise Ratio (SNR). The latter is caused by the temperature sensitivity of the hardware equipment.

The impact of temperature and humidity on the reliability that has been explored in the context of single-link transmissions, indeed, also applies when using synchronous transmissions. Furthermore, in order to achieve a temporal displacement in the order of sub-microseconds, protocols based on synchronous transmissions heavily rely on the precise and consistent operation of the nodes' electronic equipment, and especially of

the nodes' clock. However, the clocks of commodity nodes show strong drifts over time and even larger drifts when temperature, humidity or voltage change, also affecting the temporal displacement. As a consequence, the probability that synchronously transmitted packets collide destructively instead of non-destructively increases significantly. However, the effect of clock drifts on the temporal displacement has not been quantified yet.

1.1.2 *Support for event-based data traffic*

Existing protocols based on synchronous transmissions let nodes periodically flood packets through the network. This periodic flooding perfectly matches to applications that require periodic data transfer. However, often *"[d]ata is measured and collected continuously but sent only when the data measured represents previously specified importance, e.g., an event. The time of occurrence of an event is unpredictable"* [139]. In other words, in various applications the data to transmit is generated at unpredictable time instants but must be forwarded immediately to a central server. For nodes to be able to timely detect the presence of communication in form of data packets, they must regularly switch their radios on and check the channel for incoming transmissions. The more often this channel check is performed, and the longer each check lasts, the higher is the energy consumption of the nodes. Thus, channel sampling induces a high energy overhead in event-based traffic. However, these frequent channel checks are needed to ensure a timely packet delivery. This trade-off has not yet been fully explored in synchronous transmission-based protocols.

Typical scenarios for such event-based data transfer are those in which data prediction [73, 134] or compression [66, 79] is used to reduce the amount of data to transmit. Another example is the use of standard Internet protocols like IPv6 in low-power wireless networks for the interoperability and interconnection with other networks [64, 172].

In Internet protocols, data can reach up to a few hundreds of bytes and is often exchanged one-to-one between only two nodes. Thus, flooding the entire network at each data exchange – as done by many protocols based on synchronous transmissions – causes unnecessary energy consumption. However, this issue has only received little consideration yet.

1.2 OBJECTIVES AND CONTRIBUTIONS

The main goal of this thesis is to contribute novel techniques and protocols based on synchronous transmissions that enable highly reliable, low-latency distribution of event-based data in low-power wireless networks in even harsh environments. More specifically, we provide the following three main contributions:

- *On-the-fly clock offset compensation.* In this thesis, we quantify the effects of clock offset caused by temperature on the temporal displacement of synchronously transmitting nodes. To counteract this issue, we propose Flock, a protocol primitive that compensates on-the-fly for clock frequency deviations among synchronously transmitting nodes.
- *Fast flooding of small amounts of data.* We reduce the length of channel checks while still allowing the reliable detection of packet transmissions for reducing the energy consumption of the nodes by introducing Whisper. Whisper targets two types of applications: First, it can be used to disseminate small amounts of data. Second, it can be integrated in another communication protocol running within the nodes to act as wake-up primitive, i. e., using Whisper as "wake-up call" before the actual data exchange if one or several nodes have an event to share.
- *Event-based one-to-one communication.* We enable standard Internet protocols to efficiently run on top of protocols based on synchronous transmissions. In particular, we target one-to-one communication scenarios. To this end, we introduce LaneFlood, a protocol that only includes nodes that are required for data exchange, while other nodes have turned their radio off to save energy.

1.2.1 *On-the-fly clock offset compensation*

The Microcontroller Units (MCUs) of common low-power wireless nodes rely on imprecise and unstable running Digitally Controlled Oscillators (DCOs). DCOs show strong frequency drifts over time. These drifts further intensify with changing environmental parameters like temperature and humidity, leading to frequency deviations among the nodes. In this thesis, we show and quantify that small clock offsets among synchronously transmitting nodes significantly decrease the probability in achieving constructive interference. We further show that this is even exacerbated when the nodes are exposed to different temperatures. To compensate for the different clock offsets across synchronously transmitting nodes, we propose Flock: On-the-Fly Clock Offset Compensation. Instead of only relying on the DCO, Flock exploits the fine-grained, highly accurate radio clock to compensate on-the-fly for DCO offsets. Most of this contribution described above has already appeared in the following publication:

- M. Brachmann, O. Landsiedel, and S. Santini. "Keep the Beat: On-The-Fly Clock Offset Compensation for Synchronous Transmissions in Low-Power Networks." In: Proceedings of the Conference on Local Computer Networks (IEEE LCN). 2017.

1.2.2 *Fast flooding of small amounts of data*

This thesis introduces Whisper, a reliable protocol tailored to the propagation of small amounts of data. Whisper exploits synchronous transmissions and a packet-in-packet technique, which makes network floods significantly shorter compared to the state-of-the-art. This reduces the duration of a flood and further enables robust sampling of packets.

Whisper targets the quick yet energy-efficient dissemination of small data portions and it can improve the energy-efficiency of other communication protocols. The latter can be achieved by using Whisper within the other protocol as network-wide wake-up primitive when one or multiple nodes have pending data to transmit in the next communication slot. Otherwise, when nodes have not received the "wake-up call", they keep the radio turned off to save energy. Parts of this contribution is in preparation to be published:

- M. Brachmann, O. Landsiedel, D. Göhringer, S. Santini, "Whisper: Fast Flooding for Low-Power Wireless Networks", In preparation for ACM Transactions on Sensor Networks, arXiv preprint, `http://arxiv.org/abs/1809.03699`. 2018.

1.2.3 *On-demand one-to-one communication*

We introduce LaneFlood to efficiently run standard Internet protocols like UDP/TCP and IP Version 6 (IPv6) in low-power wireless networks. Most Internet protocols only target data exchange between two nodes, a source node and a destination node. Therefore, LaneFlood only involves these two nodes along with forwarding nodes in the communication. This is in contrast to existing approaches that instead inherently flood the entire network. In fact, LaneFlood creates a path between source and destination. Nodes that belong to the path stay active, otherwise they turn the radio off. In this way, LaneFlood ensures energy-efficient one-to-one communication with only a subset of nodes in the network. When source and destination have finished the packet exchange or no other node has pending data, the nodes keep their radio off for some time to save energy. Parts of this contribution have been already published:

- M. Brachmann, O. Landsiedel, and S. Santini. "Concurrent Transmissions for Communication Protocols in the Internet of Things." In: Proceedings of the Conference on Local Computer Networks (IEEE LCN). 2016.

1.3 OUTLINE

The remainder of this thesis is structured as follows: Chapter 2 introduces background information about low-power wireless networks in general and synchronous transmissions in particular. Moreover, this chapter summarizes related work. Chapter 3 presents our analysis about synchronous transmissions in harsh environments and Flock, our strategy to counteract this issue. While the goal of Flock is to increase the probability of constructively colliding packets using synchronous transmissions, the following two chapters, Chapter 4 and Chapter 5, use synchronous transmissions as underlying communication service. In Chapter 4, we present Whisper for fast flooding of event-based small data and in Chapter 5, we demonstrate LaneFlood for efficiently running standard Internet protocols in low-power wireless networks. Finally, we conclude this thesis in Chapter 6 by summarizing the achieved results and discussing limitations and possible future directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we provide the reader with background information and related work that is relevant for this thesis and show delimitations to existing solutions. We start by briefly introducing the field of low-power wireless networks for readers that are unfamiliar with the topic in Section 2.1. In Section 2.2, we analyze and discuss protocols and standards that use single-link transmissions and target highly reliable, low-latency communication. Our developed protocols and techniques build upon the IEEE 802.15.4 standard on the physical transmission level, which we briefly describe in Section 2.3. We detail synchronous transmissions in IEEE 802.15.4 by explaining the concept and surveying state-of-the-art in Section 2.4. Lastly, in Section 2.5, we summarize this chapter and distinguish the contributions of this thesis from the related work.

2.1 LOW-POWER WIRELESS NETWORKS

Traditional low-power wireless networks, also known as Wireless Sensor Networks (WSNs), are clusters of loosely coupled, spatially dispersed, autonomous devices, dubbed *nodes*. Up to thousands of these nodes are installed to sense and record physical conditions of the environment. The nodes then pass the collected data cooperatively through the network to its destination. Early applications for low-power wireless networks include habitat and environment monitoring [13, 87, 108] and the surveillance of battle fields [89, 91, 150]. Further applications include next to monitoring also actuation and control allowing for, e. g., adaptive lighting in tunnels [28] or performance control in chemical plants [123].

One big challenge in low-power wireless networks is the power consumption. The nodes are often resource-constrained, which relates to a small memory (e. g., few kByte of Random Access Memory (RAM)) and processing unit (e. g., less than ten MHz) of battery-operated, cheap devices. However, the form factor of nodes highly depends on the application. For example, applications involving sensing, actuating and controlling may also involve nodes that have more resources available. Thus, low-power wireless networks may consist of heterogeneous nodes that have different storage space, processing power and even different strategies for their power supply. Nevertheless, protocols have to run and must therefore be designed for (1) the devices with the highest constraints and thus, the lowest amount of resources available, and (2) the required application lifetime, which may range from

a few days to several years [138]. During this time, the nodes may operate unattended, e. g., in harsh and extremely dangerous areas [9, 10], or for non-intrusive habit monitoring [126]. During these long-term operations, node failures caused by physical damage, breakdown of the electronics or simply battery-depletion are common. The operation of the low-power network, however, has to continue even in case of node malfunctions. Resilience is achieved by relying on wireless connectivity and the spontaneous formation of a distributed, infrastructure-less network. Thus, low-power wireless networks have to quickly adapt to topology changes. In addition, nodes may not be able to connect via direct links due to interference or the size of the area of interest, making *multi-hop* communication – i. e., the relaying of data by intermediate nodes – a necessary requirement. Because the radio consumes the most energy in low-power devices [14, 77], communication protocols aim to save energy by turning the radio off most of the time and switching it on shortly to transmit or forward data packets. Turning on and off the radio alternately is called *duty cycling*. Thus, communication protocols target a low duty cycle in order to increase the network lifetime.

2.1.1 *Node platforms and their architecture*

Nodes in low-power wireless networks mostly consist of three modules: a sensor module, a processing module and a wireless transceiver module. The composition of the three building blocks has led to the development of two widespread hardware architectures [7, 14, 187]:

1. *MCU and radio as two separate modules:* In this architecture, the MCU executes code and the radio module enables the wireless communication. This architecture allows for flexible selection of the radio and MCU chips and thus, enables an application-aware hardware design. However, *"application developers have full access to hardware, but at the same time need to take care of all the resources"* [14].

2. *System-on-Chips (SoCs):* A SoC integrates the MCU and the radio module into a single chip. *"[A]ll of the packet processing and applications processing is performed within the single chip"* [68].

The choice of architecture highly depends on the application scenario and its required peripherals, e. g., whether the application demands a customized hardware configuration [187]. Also, one architecture may consume more energy for a specific application than the other [7, 68]. Thus, both architectures can be found in low-power wireless networks. In this thesis, we focus on node platforms with *separate MCU and radio modules*. Challenges that we discuss and solve for this architecture may not occur for SoC-architectures. In particular, the node architecture is relevant in Chapter 3, where we discuss synchronous transmissions in harsh environments.

2.1.2 *The TelosB platform*

In this thesis, we mainly use the TelosB nodes [115] as reference platform for our implementations and evaluations. For this reason, we will introduce this platform in more detail below. The TelosB is the de-facto standard in low-power wireless testbed evaluations [52, 73, 117, 146]. It features a Texas Instruments (TI) MSP430f1611 16 bit-MCU, which can operate at maximum 8 MHz and the IEEE 802.15.4-compliant CC2420 radio transceiver from TI (previously Chipcon). The TelosB thus has a separate MCU and radio module, as discussed in Section 2.1.1. The TelosB is further equipped with 10 kByte of RAM and 48 kByte of ROM. The nodes can be powered and programmed over a USB interface. For autonomous operations, the TelosB can be equipped with two AA batteries.

The TelosB, is an open platform that was designed by Moteiv Corp., a spin-off from the Berkeley University of California. The published schematics have led to several by-forms of the TelosB from different companies, e. g., Tmote Sky (Moteiv Corp.), CrossbowTelos (Crossbow Technology, Inc.), MTM-CM5000 (Advanticsys), and TPR2420 (Memsic Inc.). The hardware design of all by-forms is identical, and they only differ in the manufacturer and the hardware revisions of the components they are equipped with. Different behaviors of the TelosB by-forms is thus due to revisions of single components. In this thesis, we use the Tmote Sky platform from Moteiv Corp. and the MTM-CM5000-MSP from Advanticsys. Figure 2.1 shows both platforms side-by-side with the Tmote Sky on the left side and the MTM-CM5000 on the ride side. If the TelosB by-form is not further specified for an experiment, the evaluation results are independent of the by-form.

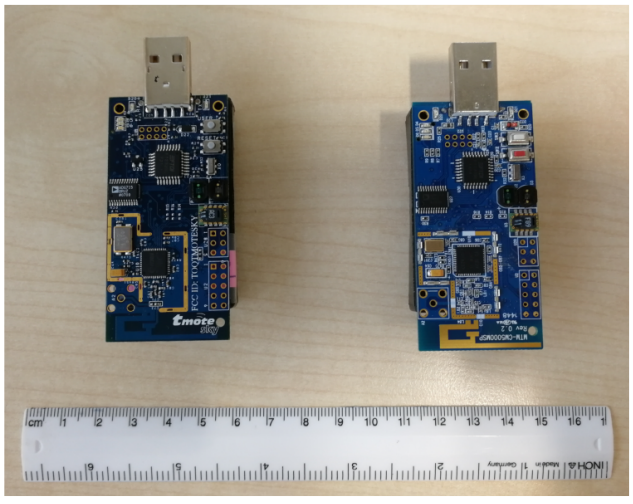


Figure 2.1: TelosB nodes. On the left side is a Tmote Sky from Moteiv Corp. and on the right side an MTM-CM5000 from Advanticsys.

2.1.3 *Testbeds for low-power wireless networks*

We use FlockLab [99] – a publicly available low-power wireless testbed – to evaluate our approaches. The FlockLab testbed is located in an office floor at the ETH Zurich. It allows to run software binaries for different hardware platforms like the TelosB (i. e., the Tmote Sky, according to a technical staff of FlockLab), TinyNode, and Opal (RF212) [99]. In this thesis, we only use the TelosB nodes to evaluate our approaches. FlockLab currently consists of 27 indoor nodes that form a multi-hop network. Figure 2.2 shows the floor plan of the office floor at the ETH with node distribution and the corresponding node identifiers.

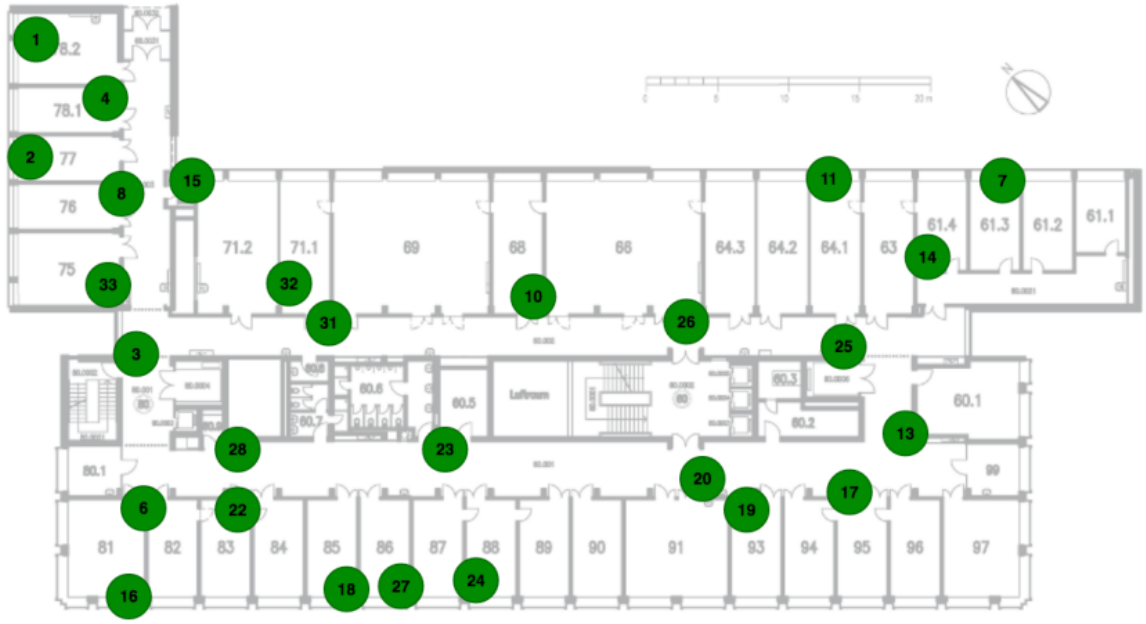


Figure 2.2: Node location in the FlockLab testbed. The FlockLab testbed has been used for evaluations throughout this thesis. It features the TelosB nodes in a multi-hop environment.

2.2 SINGLE-LINK COMMUNICATION PROTOCOLS AND STANDARDS

In the previous section, we gave an overview of the characteristics of low-power wireless networks and presented typical hardware features of the nodes. This section introduces communication protocols and existing standards for low-power wireless networks. In particular, we focus on such protocols and standards that aim for highly reliable packet delivery with low latency. In order to be able to classify our work as well as existing solutions, we first briefly introduce applications and their requirements regarding the timeliness of packet arrivals in Section 2.2.1. In our context, timeliness denotes the result of latency and

reliability. Industrial (wired) networks have had stringent requirements in reliability and the timeliness of data arrival since their deployment in the 1970s [169]. We therefore review standards and established protocols in the industrial context to derive insights that are relevant for this thesis. We start by surveying wired industrial networks in Section 2.2.2 in order to provide the necessary notions for wireless industrial networks, which we describe in Section 2.2.3. We also review open Internet standards in Section 2.2.4 before concluding this section with a subsumption of this thesis with respect to the presented standards and protocols.

2.2.1 *Requirements and applications*

The International Society of Automation (ISA) – one of the leading associations for setting global standards in automation – has classified automation and control systems regarding their importance of message timeliness into six classes. Table 2.1 summarizes the ISA classes, with class 0 involving the most time-critical data and class 5 without timeliness requirements. Most existing wireless solutions for automation and control systems realize monitoring systems (class 4 and class 5). As pointed out by [195]: *"Due to the stringent requirements for closed-loop control, it takes a long journey for the technology transit from wireless process monitoring and open-loop control to closed-loop control."* However, recent practical solutions involve open-loop control (class 3) for production monitoring and control in an oil refinery [123], and closed-loop control (class 2) for tunnel lightning [28]. Emergency actions (class 0) and regulatory control (class 1) are currently still realized by wired industrial solutions, due to their critical demands in reliability. In the following sections, we explore the industrial domain in more depth to derive insights that are relevant for this thesis. We start with wired industrial networks, which are the foundation of most wireless industrial networks.

2.2.2 *Wired industrial networks*

The family of wired communication protocols in industrial control systems is named *Fieldbus*. Fieldbus systems are defined in the International Electrotechnical Commission (IEC) 61158 standard as *"a digital, serial, multidrop, data bus for communication with industrial control and instrumentation devices such as – but not limited to – transducers, actuators and local controllers"* [55]. They are different from common Local Area Network (LAN) technologies like Ethernet in *"their robustness against harsh conditions and their ability to meet hard industrial requirements regarding real-time behavior and reliability"* [182]. A vast number of different fieldbus protocols exist, providing solutions for a variety of problems regarding application sector, requirements (see Section 2.2.1), end-user hardware and many more [169]. The most often deployed fieldbus systems are the Highway Addressable Remote Transducer (HART),

Table 2.1: ISA SP100 Application Classes. The table is taken verbatim from [31] and [179].

Category	*Class*	*Application*	*Role*	*Description*
Safety	Class 0	Emergency action	Safety-related actions, e. g., safety-interlock, emergency shutdown, and fire control	Always critical
Control	Class 1	Closed-loop, regulatory control	Motor and axis control, primary flow and pressure control	Often critical
	Class 2	Closed-loop, supervisory control	Low frequency cascade loops, multivariable controls, and optimizers with timeliness of communications measured in seconds to minutes	Usually non-critical
	Class 3	Open loop control	Actions where an operator, rather than a machine, "closes the loop" between input and output with timeliness in human scale, measured in seconds to minutes	Human in the loop
Monitoring	Class 4	Alerting	Event-based maintenance, low battery level indicator, and asset tracking.	Short-term operational consequences
	Class 5	Logging, Down-/Uploading	Sequence-of-events logs, reports of slowly changing information, preventive maintenance, and history collection	No intermediate operational consequence

Importance of message timeliness increases ↑

Foundation Fieldbus, Profibus, Controller Area Network (CAN) bus, and Modbus [55, 78, 169]. While all fieldbus protocols differ in their technical concept, they all implement reliable wired communication in a bus, star, ring, or tree topology [169].

Despite their ability to achieve the stringent requirements of emergency actions (class 0) and regulatory control (class 1) compared to wireless solutions, wired networks are inherently costly and inflexible. Nonetheless, the switch from wired to wireless networks is a long-lasting step. Thus, most wireless solutions have been built to complement wired standards, as we discuss in the following section.

2.2.3 *Wireless industrial networks*

In this and the following section, we review wireless standards and protocols to derive insights that are relevant for this thesis. We later summarize our findings in Section 2.2.5. While the following Section 2.2.4 deals with open standards, in this section we discuss protocols and standards that are designated for the deployment in industrial plants and

factories. Four important standards have been defined by various working groups for wireless multi-hop networks in industrial settings: WirelessHART, ISA 100.11a, Wireless Networks for Industrial Automation – Process Automation (WIA-PA), and ZigBee PRO. After presenting these standards, we briefly introduce the Wireless Interface for Sensors and Actuators (WISA), a commercial system from ABB, and the research project GINSENG.

The WirelessHART standard was released in 2007 by the HART Communication Foundation (HCF) [94] and approved 2009 by the IEC as international standard IEC 62591 [33]. It was developed to complement the HART fieldbus with the possibility of wireless communication [78]. The ISA, and more precisely, the ISA100 standard committee has defined and published the ISA 100.11a standard in 2009 [139, 195]. In 2012 ISA 100.11a became the international standard IEC 62734 [34]. WIA-PA was proposed 2007 by the Chinese Industrial Wireless Alliance (CIWA) [94] and became an international standard in 2009 named IEC 62601 [32]. In 2007 the ZigBee Alliance presented ZigBee PRO, which extends ZigBee (2003). While ZigBee targets home automation and consumer electronics, ZigBee PRO was designed for industrial automation [54, 78].

While WirelessHART only processes HART commands at its application layer, ISA 100.11a and WIA-PA also handle Fieldbus Foundation, Profibus, and Modbus commands [78, 139] (see Section 2.2.2). ZigBee PRO is, instead, not coupled to fieldbus commands. All four standards are based on the IEEE 802.15.4 standard. The IEEE 802.15.4 standard specifies the physical and Medium Access Control (MAC) layer for low-power wireless networks and is also the foundation of the protocols designed in this thesis. We therefore introduce the IEEE 802.15.4 standard in more detail in Section 2.3. WirelessHART and ISA 100.11a rely on the Time Synchronized Mesh Protocol (TSMP) [127], which uses Time Division Multiple Access (TDMA) to schedule the medium access and Frequency Division Multiple Access (FDMA) to allow for frequency hopping. Instead, WIA-PA implements TDMA and CSMA with Collision Avoidance (CSMA/CA) for accessing the channel and FDMA for frequency hopping [192]. ZigBee PRO uses CSMA/CA and employs a "frequency agility" concept. Frequency agility *"allows for an entire network to change its operative channel when faced with reduced link qualities caused by noise and/or interference"* [78]. In contrast to channel hopping, frequency agility is thus less tolerant to fluctuating channel conditions. In WirelessHART each node is a routing node that maintains its own neighbor table. Further, each WirelessHART network consists of exactly one network manager, which maintains and configures the network. For example, it has full knowledge of the network topology and all existing devices in the network. The network manager configures the routing path between any source and destination in the network, e. g., by requesting a node's neighbor table along with the battery status, ensuring an optimal routing path. ISA 100.11a, WIA-PA, and ZigBee PRO distinguish between routing and non-routing (e. g., sensors and actuators) nodes. ISA 100.11a further applies IPv6 for node addressing, which can also be used in ZigBee PRO. Thus, nodes in a ISA 100.11a or ZigBee network are directly accessible via

the Internet, allowing, e. g., to interconnect multiple production plants, as we discuss in Section 2.2.4 and Section 5.

WISA is a commercial system from ABB. It provides wireless communication as well as wireless power supply [145]. The latter is achieved using magnetic coupling. The wireless communication is based on IEEE 802.15.1 (the physical layer that is used in Bluetooth) at the physical layer and TDMA with frequency hopping at MAC layer. In WISA, up to 120 input/output devices can be connected to a base station, arranged in a star topology with a communication range of 5 to 10 m. The base station is expected to be connected to a controller over a wired fieldbus [145, 183].

The GINSENG system [123] is a research project that realizes performance control using IEEE 802.15.4-based low-power wireless networks. It offers a standalone solution including network protocols, system software, and the integration of industrial-suited middleware. GINSENG proposes GinMAC, a TDMA-based single-channel MAC protocol for low-power nodes that further includes queue management (GinQueue), topology control (GinTop), and performance monitoring (GinPerf).

2.2.4 *Standard Internet protocols for low-power wireless networks*

The solutions presented in Section 2.2.3 are designated protocols for the deployment in industrial plants and factories. A different direction are the so-called *IP-enabled* low-power wireless networks, also referred as *OpenWSN* [39, 86, 170]. As mentioned before, the ISA 100.11a and ZigBee standards have the advantage to be accessible via the Internet. However, instead of relying on proprietary or closed standards, IP-based low-power wireless networks use open standards, specified by the Internet Engineering Task Force (IETF). These networks are not dedicated for industry, but serve the purpose of the more general vision of the Internet of Things (IoT), where all devices are interconnected. By acquiring IPv6 addresses, each device, including low-power wireless nodes, can become either a client or a server. Clients and servers can interact via standard Internet protocols like User Datagram Protocol (UDP) or Transmission Control Protocol (TCP), the Hypertext Transfer Protocol (HTTP) or the Constrained Application Protocol (CoAP) – a lightweight alternative

Table 2.2: Envisioned protocol stack for Internet Protocol (IP)-enabled low-power wireless networks [39, 86].

OSI layer	Solution	Standard
Application	CoAP/HTTP	RFC 7252 [19]/RFC 2616 [53]
Transport	UDP/TCP	RFC 768 [129]/RFC 793 [27]
Network	IPv6 + 6LoWPAN + RPL	RFC 8200 [35] + RFC 6282 [65] + RFC 6550 [23]
Data Link	IEEE 802.15.4e TSCH	IEEE 802.15.4 [67]
Physical	IEEE 802.15.4	IEEE 802.15.4 [67]

of HTTP. Table 2.2 illustrates the envisioned protocols in the Open System Interconnection (OSI) model. On the physical and data link layer, IEEE 802.15.4 is expected. In 2012 Timeslotted Channel Hopping (TSCH) was introduced as an amendment to the MAC layer of IEEE 802.15.4. TSCH was derived from WirelessHART and ISA 100.11a and applies TDMA and channel hopping. Thus, with IEEE 802.15.4e-2012, the achieved reliability is comparable with WirelessHART and ISA 100.11a networks [39]. IPv6 over Low power Wireless Personal Area Network (6LoWPAN) is used as adaptation layer to perform header compression and packet fragmentation. Both is necessary since the maximum packet size in IEEE 802.15.4 is limited to 127 byte, while IPv6 allows packets of size 1280 byte. The IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is the state-of-the-art protocol in low-power wireless networks to guide packets between client and server.

2.2.5 *Subsumption of this thesis in the context of single-link protocols and standards*

All the solutions for wireless communication presented in Section 2.2.3 and Section 2.2.4 have in common that they rely on single-link transmissions with designated time-slots for each node for a quick and collision-free data distribution. Data distribution is achieved by these solutions through routing protocols. In contrast to these approaches, we use synchronous transmissions, where nodes synchronously propagate packets through the network. Instead of routing, we use flooding or let the nodes create a path between any two nodes in the network to flood data along the path. Using this approach, we show in this thesis that we enable synchronous transmissions to efficiently run standard Internet protocols as discussed in Section 2.2.4. Further, the presented approaches apply a form of frequency switching in case of strong interference. In this thesis, we do not explicitly apply frequency hopping. Effective solutions for frequency hopping when using synchronous transmissions already exist [36, 48, 49, 74, 112, 118, 120, 130] and can be integrated straightforward in the solutions presented in this thesis. In the following Section 2.3, we introduce the IEEE 802.15.4 standard, which builds the basis on the physical layer for our approaches. Afterwards, we explain synchronous transmissions for IEEE 802.15.4 networks in detail in Section 2.4.

2.3 THE IEEE 802.15.4 STANDARD

The IEEE 802.15.4 standard is the underlying communication standard for Low-Rate Wireless Personal Area Networks (LR-WPANs). LR-WPANs comprise ubiquitous low-cost, low data rate communication between devices with little or no underlying infrastructure. In particular, the IEEE 802.15.4 standard specifies the physical layer and the MAC sublayer for LR-WPANs. As mentioned in Section 2.2.3, IEEE 802.15.4 is the basis of, e. g., ZigBee, ISA 100.11a, and WirelessHART. IEEE 802.15.4 operates transnational at the Industrial, Sci-

entific and Medical (ISM) band at 2.4 GHz. It, thus, has to share the available frequencies for communication with other wireless technologies like IEEE 802.15.1 (Bluetooth) and IEEE 802.11 (Wifi) that have higher transmit powers. Several amendments for IEEE 802.15.4 exist that specify additional physical layer mechanisms (e. g., IEEE 802.15.4a improves localization), or address country-specific regulations and markets (e. g., IEEE 802.15.4c and IEEE 802.15.4d add new radio frequencies for China and Japan, respectively).

The CC2420 radio chip that is implemented on the TelosB nodes – our reference platform in this thesis as described in Section 2.1.2 – realizes the physical layer operations of the IEEE 802.15.4 standard. Similar to WirelessHART and ISA 100.11a, in this thesis, we design our own strategy for channel access, based on synchronous transmissions. We therefore neglect the MAC sublayer specification of IEEE 802.15.4. The protocols designed in this thesis heavily rely on the implemented physical layer of IEEE 802.15.4, and in particular on the Offset-Quadrature Phase Shift Keying (O-QPSK) modulation with half sine pulse shaping, and we therefore describe the important aspects in the following. In particular, we detail the structure of IEEE 802.15.4 packets in Section 2.3.1 and clarify the signal modulation and demodulation in Section 2.3.2.

2.3.1 *Packets in IEEE 802.15.4*

Figure 2.3 shows the structure of an IEEE 802.15.4 packet. A packet always starts with a synchronization header that consists of a preamble and a Start-of-Frame Delimiter (SFD). The preamble is 4 byte set to `0x00`. It is used by receivers to achieve symbol synchronization and to adjust for frequency offsets [72]. The SFD is a one byte value set to `0xa7` by default. It indicates the end of the preamble and the start of the packet data. The SFD field is followed by a 1 byte packet length field. The length field defines the number of bytes in the MPDU. The MPDU consists of the payload, which contains the actual data, and a 2 byte footer. The footer includes a Frame Check Sequence (FCS) that is calculated over the payload for bit error detection. During reception, a Cyclic Redundancy Check (CRC) using the FCS is performed. A receiver was not able to correctly decode a packet when the CRC fails.

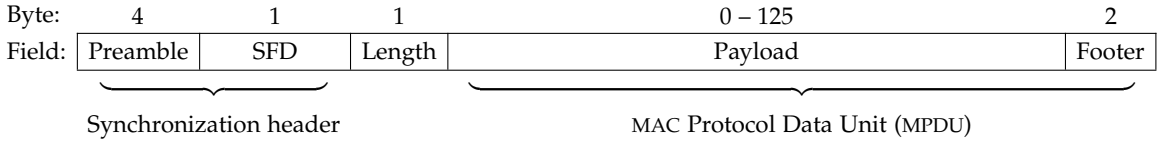


Figure 2.3: Structure of an IEEE 802.15.4 packet.

2.3.2 *Signal (de-)modulation and spreading in IEEE 802.15.4*

Before transmission, each byte of a packet is split into two symbols, thus one symbol corresponds to four bits. Each symbol is mapped to one out of 16 orthogonal chip sequences, called Pseudo-random Noise (PN) sequence. Each PN sequence consists of 32 chips. The mapping of bits to a PN sequence is known as Direct Sequence Spread Spectrum (DSSS). Four bits are, thus, spread to 32 chips. IEEE 802.15.4 uses DSSS in order to make the transmitted signal impervious to narrowband interference signals and to increase the SNR at the receiver. The PN-sequences are concatenated and modulated as half-sine onto the carrier, starting from the least significant chip. The CC2420 uses as modulation format O-QPSK with half sine chip shaping, which is equivalent to Minimum Shift Keying (MSK) modulation [72]. Consequently, each even-indexed chip is modulated onto the in-phase (I) carrier and each odd-indexed chip is modulated onto the quadrature-phase (Q) carrier with one-half period offset. The chips are transmitted at 2 MChips/s, thus the offset between each chip, which is the inverse chip rate, is 0.5 µs. The resulting data rate is 250 kbit/s at the 2.4 GHz band [67]. The modulated I and Q channels that are finally transmitted are called *signals*.

On receiving a signal, each half-sine from the I and Q channel is demodulated to a chip and 32 chips are grouped to one PN sequence. The radio performs soft decision on chip-level, i. e., the PN sequences may contain chips with non-binary values between 0 and 1 [72]. The de-spreading from PN sequences to symbols is performed by hard decision through mapping each PN sequence to the symbol with the highest correlation. The redundancy that is induced through DSSS increases the correct decoding of the received signal, even when a few chips have not been correctly decoded. The CRC after packet reception fails when the receiver has decoded too many chips in a PN sequence incorrectly, resulting in false symbol mappings and thus, falsely decoded bits.

2.4 SYNCHRONOUS TRANSMISSIONS IN LOW-POWER WIRELESS NETWORKS

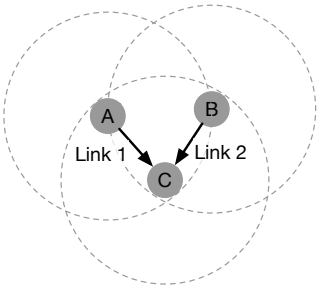
In the last section, we introduced the IEEE 802.15.4 standard and in particular the O-QPSK with half sine pulse shaping modulation[1], which builds the foundation on the physical layer for our synchronous transmission-based approaches presented in this thesis. Synchronous transmissions have shown to boost the performance of low-power wireless networks in terms of reliability, latency and energy-efficiency. We first provide the basic notions of synchronous transmissions in IEEE 802.15.4 in Section 2.4.1. Afterwards, we present a brief history of synchronous transmissions in Section 2.4.2. This thesis mainly builds upon

1 Henceforth, we use IEEE 802.15.4 as shortcut for IEEE 802.15.4 with O-QPSK with half sine pulse shaping modulation.

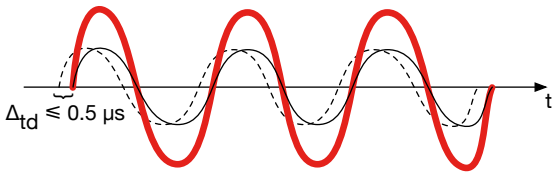
Glossy, the state-of-the-art protocol of synchronous transmissions. We therefore introduce Glossy and its basic principle in Section 2.4.3.

2.4.1 *Synchronous transmissions in IEEE 802.15.4*

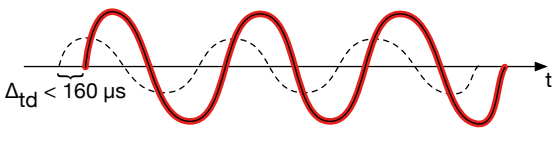
Using synchronous transmissions, two or more nodes transmit packets simultaneously to at least one common receiver in their one-hop range. At the receiver, two events can happen: either the receiver is able to correctly decode one packet, or the reception fails because packets collide and the CRC fails. A successfully received packet during synchronous transmissions is the result of *non-destructive interference*. Two physical phenomena exist that affirm non-destructive interference: *constructive interference* and the *capture effect*. Figure 2.4 illustrates both phenomena on the example of two synchronously transmitting nodes A and B, and a common receiver node C. In the following we explain these two phenomena in more detail in the context of IEEE 802.15.4. Note that the time values specified in the following only apply for O-QPSK with half sine pulse shaping modulation. The time values change with other modulation schemes.



(a) Nodes A and B transmit synchronously to node C.



(b) The signals of nodes A and B interfere constructively at node C.



(c) Node C receives the packet from node B due to the capture effect.

Figure 2.4: Non-destructive interference due to constructive interference and the capture effect. When two or more nodes transmit a packet synchronously, a receiver is able to successfully decode the incoming packet because the corresponding signals interfere non-destructively due to either constructive interference or the capture effect. Non-destructive interference, indeed, requires precise timing of the transmissions.

CONSTRUCTIVE INTERFERENCE. When two or more packets precisely overlap at a common receiver, their baseband signals may interfere constructively and superimpose. To achieve constructive interference, two conditions must be fulfilled:

1. The nodes must transmit identical packets, and thus, identical signals.
2. The transmissions of multiple senders must be precisely timed so that their signals are in phase with each other and the receiver is able to decode a sufficient amount of chips in a PN sequence.

Ferrari et al. [52] have theoretically and practically shown that a receiver is able to successfully decode a packet when the *temporal displacement* Δ_{Td} of two or more signals is 0.5 µs or less, which corresponds to the one-half chip period offset, described in Section 2.3.2. Henceforth, we refer this requirement for achieving constructive interference *the temporal displacement of* $\Delta_{Td \leqslant 0.5\,\mu s}$. Figure 2.4b illustrates the superimposition of the signals from node A and B at node C due to constructive interference. To achieve constructive interference, and thus, allow the arrival of signals at (exactly) the same time at a common receiver requires the precise timing of the packet transmissions. This is because propagation delay is negligible in low-power wireless networks due to the limited communication range of transceivers.

CAPTURE EFFECT Due to the capture effect, an IEEE 802.15.4-compliant receiver has a high probability to successfully intercept and decode a packet when the temporal displacement Δ of two incoming signals is greater than 0.5 µs. The capture effect, and in particular the *power capture effect*, occurs when *"the power of the signal of interest exceeds the sum of interference from colliding packets by a certain threshold"* [181]. In other words, a receiver has a high probability to correctly decode a packet when its signal is stronger than the sum of all interfering signals from other currently transmitting nodes. There are two conditions that must be fulfilled in order to make the capture effect happen:

1. The received signal must be at least 3 dBm stronger than the sum of the interfering signals [90].
2. The stronger signal must arrive before the radio has completely received the synchronization header of the weak signal [90, 188]. The IEEE 802.15.4 standard defines the length of the synchronization header to be 5 byte (see Section 2.3.1), which corresponds to 160 µs.

In Figure 2.4c, node C receives the packet of node B due to the capture effect. Note that signal link 2 (signal from node B) is the received signal by node C (highlighted in red). The capture effect transpires regardless whether the incoming signals are identical. Thus, it also occurs when two or more nodes transmit different packets. However, the capture effect is not scalable to numerous synchronous transmitters.

The probability to correctly decode a packet decreases with an increasing number of synchronously transmitting nodes [52, 90].

When during synchronous transmissions either constructive interference or the capture effect occur, the receiver is able to successfully decode an incoming packet with high probability. Thus, the signals have interfered non-destructively. However, signals may also interfere destructively. During *destructive interference*, the signals cancel each other out. In particular, when one signal is at its crest and the other signal is at its through, the amplitudes add up to their difference, which in this case is a resulting amplitude of zero. However, a complete cancellation of signals rarely happens. Typically, the signals only partially interfere destructively, which results in few falsely decoded symbols by the receiver and thus, to a failed CRC.

2.4.2 *A brief history of synchronous transmissions*

More and more protocols have been building upon constructive interference and the capture effect for the past few years. Figure 2.5 shows an excerpt of the published papers. The papers shown in the figure are grouped on the y-axis in four different categories: papers that examine synchronous transmissions, protocols that exploit the capture effect, protocols that exploit both constructive interference and the capture effect, and protocols that investigate security in synchronous transmission-based networks. In the following, we provide a brief history of the rise of synchronous transmissions in low-power wireless networks.

Until the early 2000s, it was assumed that packet collisions necessarily result in packet corruptions [82, 160, 167, 174]. Different collision avoidance schemes have been proposed like TDMA, Carrier-Sense Multiple Access (CSMA), or Request to Send / Clear to Send (RTS/CTS). However, CSMA and RTS/CTS approaches cannot prevent collisions in any case [11, 174]. In 2005, *Whitehouse at al.* [180] showed that the capture effect can help in detecting such packet collisions and also allows the recovery of the packet of the stronger signal. Between 2006 and 2008, as illustrated in Figure 2.5, *Son et al.* [155–158] performed a series of experiments, where they investigated the effects of multiple concurrently transmitting nodes on the packet reception reliability of a single sender-receiver pair. They found that the capture effect helps significantly in increasing the packet reception reliability. As a consequence, concurrently transmitting sender-receiver pairs increase the overall network throughput. Up to this point, the term *concurrent transmissions* refers to the simultaneous data transfer between nodes regardless of the time of the transmission start. Later (around 2011) until today, the term concurrent transmissions has been also used as a substitutional for synchronous transmissions, where senders simultaneously start transmitting packets.

Dutta et al. [43] have been the first that considered simultaneous transmission starts of concurrently transmitting nodes to achieve non-destructive interference in their Backcast

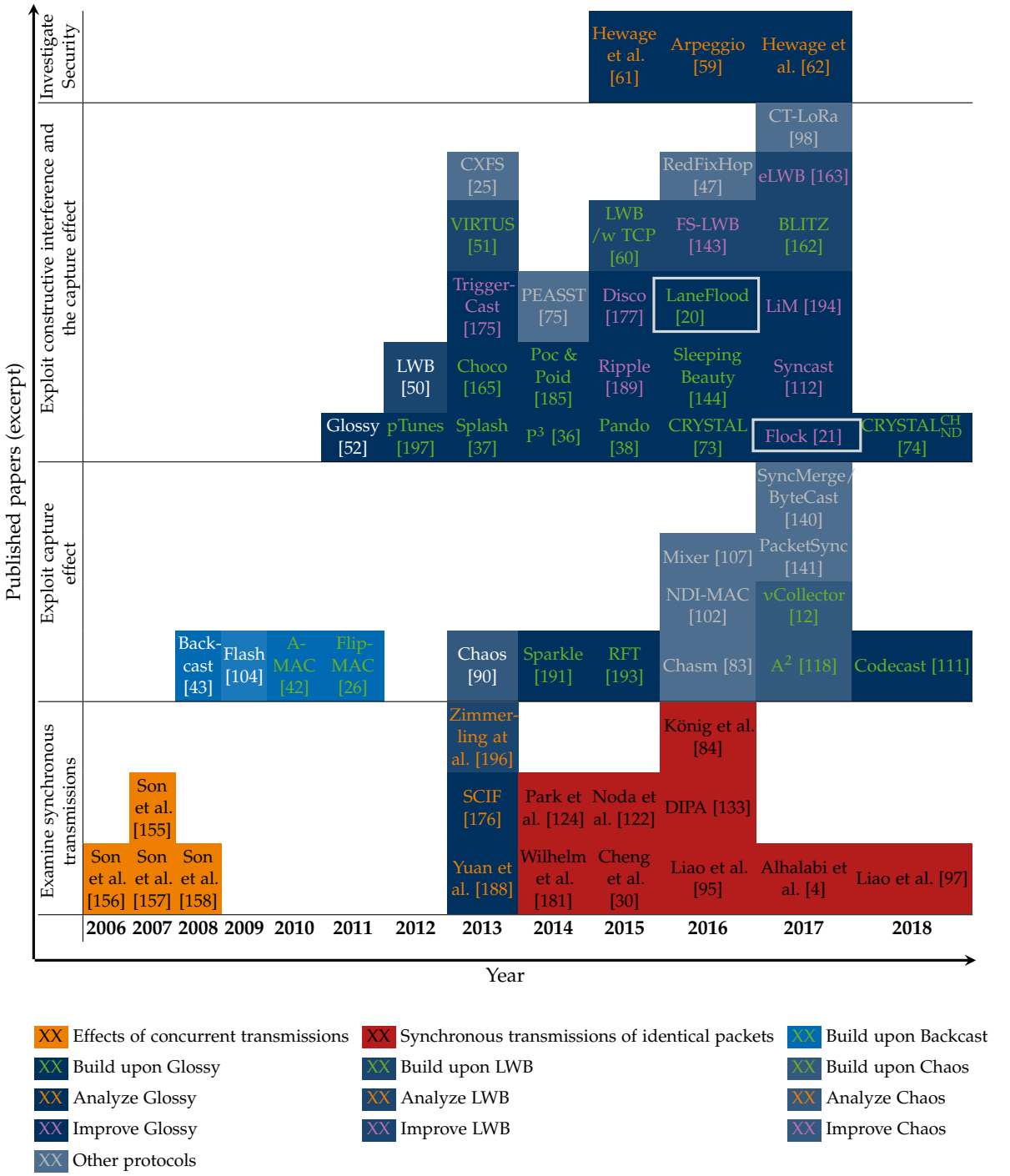


Figure 2.5: Brief history of the rise of synchronous transmissions. Protocols with white font color mark protocols that pioneered the research in synchronous transmissions, namely Backcast [43], Flash [104], Glossy [52], LWB [50], and Chaos [90]. A variety of novel protocols and approaches emerged based on the protocols named above. Framed protocols mark the approaches published by the author of this thesis.

primitive. In Backcast, a sender transmits a packet and all destination nodes confirm the reception by synchronously transmitting short hardware-generated acknowledgments. Thus, the sender can efficiently determine that at least one destination has received the packet. This principle has been applied to receiver-initiated protocols like A-MAC [42] and Flip-MAC [26]. While Backcast concurrently transmits short hardware-generated acknowledgments, Flash [104] successfully transmits longer data packets concurrently.

While the success of packet reception in the previous approaches heavily relies on the capture effect, *Ferrari et al.* [52] are the first with Glossy that exploit constructive interference when transmitting long data packets efficiently over a multi-hop network. Glossy has sparked a new research direction in wireless low-power communication protocols that timely transmit identical or different packets to exploit the capture effect or constructive interference. Two protocols that have further pushed synchronous transmissions are the Low-power Wireless Bus (LWB) [50] and Chaos [90]. LWB [50] uses Glossy to create a virtual wireless bus system based on consecutive Glossy floods. Chaos [90] integrates Glossy's principle but instead of sending identical data, the nodes transmit different data packets to efficiently support all-to-all data sharing. In recent years, a plethora of work have appeared that improve, analyze or build upon Glossy, LWB, and Chaos or examine constructive interference-based transmissions, as illustrated in Figure 2.5.

The successful packet reception in synchronous transmission-based approaches have not only been evaluated in dedicated simulations or testbed experiments, but have also been shown in the EWSN dependability competition [147]. The EWSN competition is co-located with the International Conference on Embedded Wireless Systems and Networks (EWSN) and aims to *"benchmark the dependability of state-of-the-art IoT protocols in environments rich with radio interference"* [147]. Competitors are ranked according to their overall performance regarding the following evaluation metrics: reliability, latency, and energy-efficiency. Figure 2.6 shows the protocols that have participated per year on the y-axis and the according ranks on the x-axis. Note that rank 4 is always given to two protocols and X indicates that the protocols have not been ranked. We clustered the protocols in synchronous transmission-based, TSCH-based, and other protocols. We already introduced the OpenWSN protocols stack and TSCH in Section 2.2.4. The TSCH-based protocols in the competition, however, implemented different strategies for TSCH without the layers above data link. The category "other protocols" includes all protocols that are neither synchronous transmission-based nor TSCH-based. As shown in the figure, approaches that utilize synchronous transmission always ranked the first three positions since the start of the competition in 2016. This success story impressively demonstrates the promising potential of synchronous transmissions in low-power wireless networks.

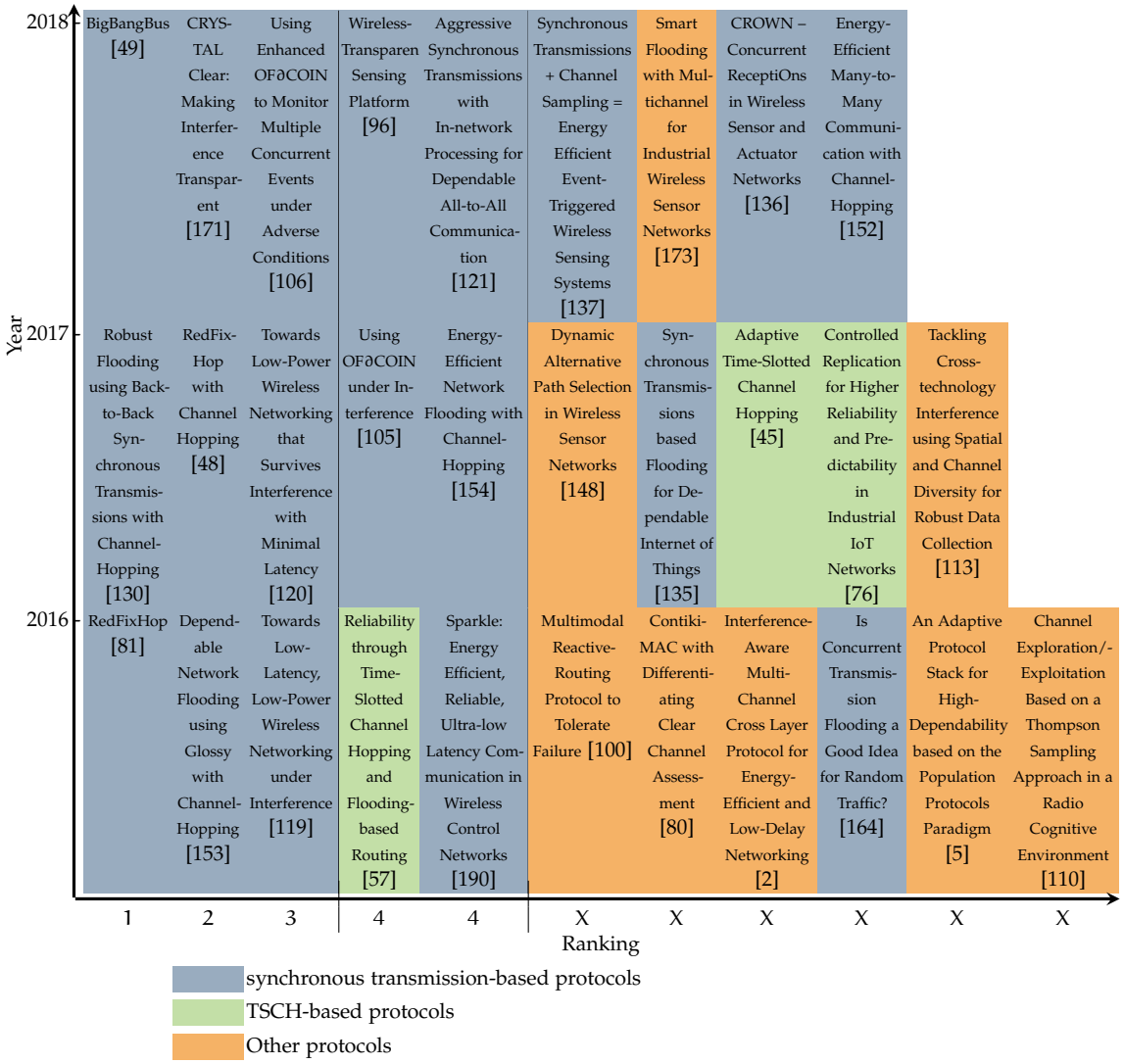
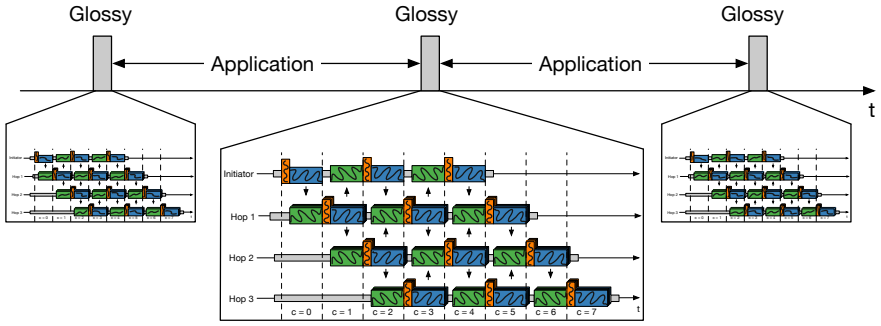


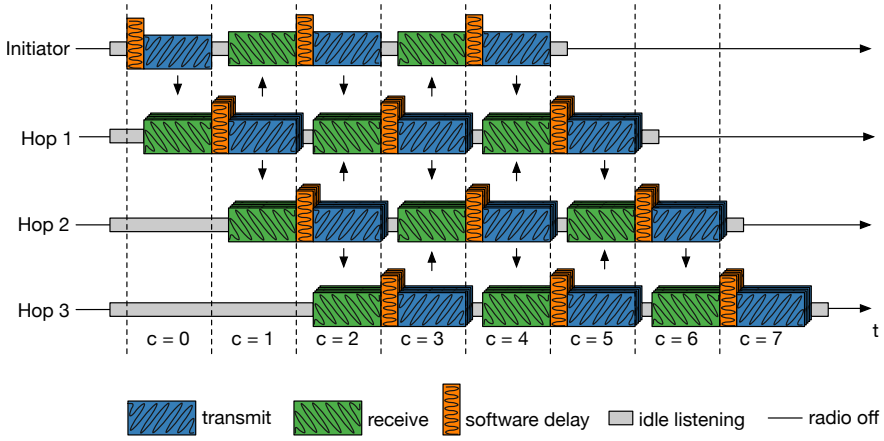
Figure 2.6: Protocols and their ranking at the EWSN dependability competition. Protocols based on synchronous transmissions have ranked the first three positions since the start of the competition in 2016.

2.4.3 *Synchronous transmissions with Glossy*

As illustrated in the previous section, Glossy has pioneered a new research direction. Also, the protocols presented in this thesis are based on Glossy. We therefore explain the operation of Glossy in more detail in the following.



(a) Application tasks are suspended during periodic Glossy floods.



(b) Glossy flood with $N_{tx} = 3$.

Figure 2.7: Glossy in action. Glossy is scheduled periodically. After suspending application tasks, Glossy follows a receive-and-relay scheme. Thereby, nodes synchronously transmit packets to their one-hop neighbors. After transmitting a packet N_{tx} times, the nodes turn their radio off. The figures are slightly modified versions of Figure 3 and Figure 6 from [52].

Due to its very strict timing requirements, synchronous transmissions and thus, non-destructive interference-based flooding has not been applied for data sharing until Glossy [52] was published in 2011 by *Ferrari et al*. Glossy provides quick data dissemination through flooding with implicit time synchronization of a network. As shown in Figure 2.7a, it decouples Glossy floods from the application. By suspending application tasks on the host system during network floods, Glossy achieves a synchronization accuracy of sub-

microseconds and quickly disseminates a packet through a 5-hop network within 2 ms and a reliability close to 100 %. The packet dissemination is triggered by a single, dedicated node, called *initiator*. Figure 2.7b illustrates a Glossy-flood on the example of a 3-hop network. The initiator transmits a packet and all nodes in the first hop receive and immediately relay this packet synchronously after a short processing time, called *software delay* T_{sw} throughout this thesis. Next, the second hop and the initiator receive the packet and also relay it. This receive-and-relay scheme continues until all nodes have transmitted a packet N_{tx} times. Afterwards, the nodes turn their radio off. The nodes have their radio turned on during the entire flood, which takes only a few milliseconds, resulting in a low packet distribution latency. A node has at least $2N_{tx} - 1$ opportunities to receive a packet at least once, resulting in a high reliability. Constructive interference is achieved when the software delay T_{sw} has the same duration in each node and thus, nodes within the same hop start their packet transmission at the same time instant. Each packet contains of a *relay counter* c that the nodes increase before each transmission. The relay counter c indicates, thus, how often a packet was already flooded in the network. Knowing c of the first packet a node has received, the duration of the software delay T_{sw}, and the packet length, the nodes synchronize to the initiator by computing the time the initiator has started the flood, the so-called *reference time*. The nodes turn their radio on for the next flood according to the reference time and the periodic flooding frequency. After the next flood, the nodes compare the expected with the actual start time of the flood and consequently are able to determine potential clock drifts. The nodes include the computed clock drift in their calculation for the beginning of the next Glossy-flood, resulting in a high accuracy in the simultaneous wake-ups of the nodes, and thus in a low radio duty cycle.

The description above only provides an overview and does not cover Glossy in a comprehensive manner. We provide more details in related chapters in this thesis.

2.5 SUMMARY AND SUBSUMPTION OF THIS THESIS

In this chapter, we provided the fundamentals relevant for this thesis and surveyed related work. Synchronous transmission-based protocols, and especially Glossy, LWB, and Chaos, have shown impressive performance regarding reliability, latency, and energy-efficiency. By building upon Glossy, this thesis tackles several shortcomings of the related work based on synchronous transmissions: We first show and mitigate clock drift variations among synchronously transmitting nodes that are caused, e. g., by harsh environmental conditions like strong temperature and humidity fluctuations in Section 3. Second, we address fast flooding of event-based data. As discussed in Section 2.4.3, Glossy runs periodically and follows a receive-and-relay scheme with a short software delay T_{sw} in between. In Section 4, we show that this scheme is energy-inefficient in event-based traffic scenarios with small data packets like configuration parameters or control information. We therefore

propose a novel packet-in-packet technique that exploits synchronous transmissions and in which nodes only initiate communication when they have an event to share. Last, Glossy periodically floods the entire network. In Section 5, we introduce a protocol that only involves required nodes in the communication. Nodes that are not required, or in case of no pending data traffic, the nodes turn their radio off.

In this thesis, we address high reliability, high energy-efficiency, and low latency. However, each solution in this thesis targets a different metric. Flock (Chapter 3) focuses on reliability, Whisper (Chapter 4) on reliability and energy-efficiency, and LaneFlood (Chapter 5) aims for all three metrics. Referring to table 2.1, this thesis targets applications in class 2 to class 5.

CHAPTER 3

ON-THE-FLY CLOCK OFFSET COMPENSATION

In this chapter, we address synchronous transmissions in harsh environments, in which low-power wireless networks are typically deployed. Within such environments, the nodes are exposed to strong changes in environmental parameters, e. g., temperature and humidity. It has been shown in the literature that these changes in environmental parameters have strong influence on the Received Signal Strength (RSS) and hence, on the packet reception reliability [16, 17, 168, 178].

In our first contribution, we show that also the temporal displacement Δ_{Td} of synchronous transmissions is affected by the changes in environmental parameters. In particular, it causes to exceed the temporal displacement Δ_{Td} that is required for achieving constructive interference. The reason for this temporal drift of incoming signals lies in the sensitivity of the node's clocks to temperature and humidity changes. More precisely, synchronous transmissions heavily rely on the precise and consistent operation of the nodes' clock to align the duration of the software delay T_{sw} among the nodes (see Section 2.4.3). The clock of the MCU in commodity nodes, however, shows strong drifts over time and stronger drifts of up to 20 % when the temperature changes [18, 52]. These drifts are the cause of varying software delays T_{sw} among synchronously transmitting nodes. As an example, Glossy makes the MCU execute a fixed number of clock cycles during T_{sw} to ensure that nodes start their transmission at the same time. Further, Glossy limits the number of MCU cycles to only less than a hundred to mitigate the impact of MCU clock drift that occurs over time. However, we show in this chapter that the execution of these few clock cycles can already result in a temporal displacement Δ_{Td} that exceeds the requirement for achieving constructive interference. We further quantify Δ_{Td}, when the nodes are exposed to temperature differences of up to 30 °C.

To address the problem clock drifts that cause *clock offsets* among synchronously transmitting nodes, we present Flock: On-the-Fly Clock Offset Compensation. Flock compensates for these clock offsets and thus, increases the probability that the signals align constructively. Flock, hence, makes Glossy, and protocols building upon it, more robust to operate in harsh environments. Furthermore, Flock relaxes an integral part of Glossy: Glossy was not designed to allow for additional operations during T_{sw}. More precisely, it strives to minimize the number of software instructions that are executed during T_{sw} to mitigate the

impact of clock frequency deviations. Thus, approaches that require intermediate operations, e. g., switching channels to increase resilience [48, 120, 130] or processing received packets [90], can apply Flock to compensate for clock frequency misalignments and thus relax Glossy's timing constraints during T_{sw}.

The remainder of this chapter is structured as follows: In Section 3.1 we detail how Glossy achieves $\Delta_{Td\leqslant 0.5\mu s}$ and show the effects of clock frequency deviations on Glossy in Section 3.2. We introduce Flock in Section 3.3 and evaluate its performance in simulations and on real nodes in Section 3.4. We describe related work in Section 3.5 and conclude this chapter in Section 3.6. We have published most of the contributions presented in this chapter in [21].

3.1 ENSURING SYNCHRONOUS TRANSMISSIONS WITH GLOSSY

We have shown in Section 2.4.3 that Glossy's network flood is based on an alternating sequence of receive and relay events that precisely overlap among the nodes[1]. Glossy's receive-and-relay scheme is driven by radio events only, allowing the alignment of synchronous transmission with the absence of explicit node synchronization, as described in Section 2.4.3. More precisely, the start and end of each packet reception as well as the start and end of each transmission triggers a pin in the radio, the so-called *Start-of-Frame Delimiter (SFD) pin*. Figure 3.1 illustrates the SFD pin activity. The SFD pin is set after successfully decoding a packet's SFD field on reception. Because the propagation delays are negligible in low-power wireless networks (in the order of nanoseconds), the SFD pin rises on all receiving nodes at almost the same time. The SFD pin stays active until the entire packet has been received. In other words, the interval in which the SFD pin is active indicates the *packet reception duration* T_{rx}. After the nodes have successfully received the packet, they (almost) immediately turn the radio from receive to transmit mode, which called *rx/tx turnaround*, and retransmit the packet, indicated with T_{tx} in Figure 3.1. Glossy ensures that the elapsed time between reception and the triggering of the rx/tx turnaround very short. More precisely, the falling edge of the SFD pin causes an interrupt on the MCU that indicates the complete reception of the incoming packet. Within the interrupt service routine, Glossy aims to quickly initiate the retransmission of the just received packet. The time it takes for the MCU to issue this retransmission is what we have introduced in Section 2.4.3 as the software delay T_{sw}. In order to ensure a temporal displacement of $\Delta_{Td\leqslant 0.5\mu s}$ among synchronously transmitting nodes – which is required to achieve constructive interference, as discussed in Section 2.4 – Glossy eliminates almost completely all possible contingencies that can affect the duration of T_{sw}. For example, it accounts for the interrupt delay. The interrupt delay is between 1 and 6 instructions in the MSP430f1611 –

1 This section is an extended version of Section II-A and Section II-B of our previous work published in [21].

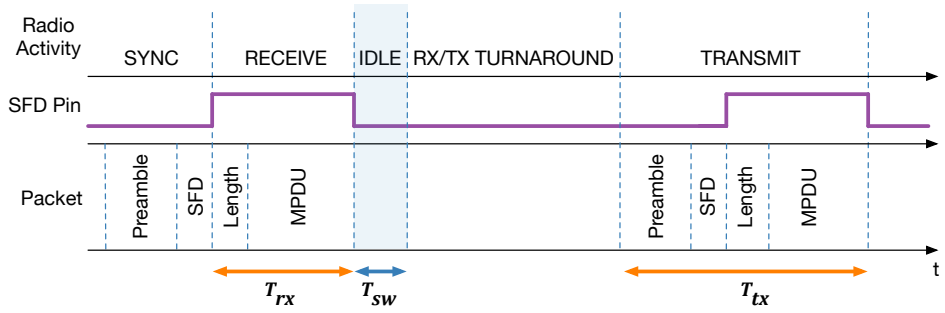


Figure 3.1: Radio activity during packet reception and retransmission. The SFD pin rises after successfully receiving/transmitting the packet's SFD field and falls after the last byte of a packet has been received/transmitted. The falling SFD pin after reception triggers in interrupt in which the MCU initiates the retransmission of the currently received packet and accounts for the interrupt delay. During this short period, software instructions are executed by the MCU, which is sourced by the DCO while the radio is idle. The packet reception duration T_{rx}, the software delay T_{sw} ,and the packet transmission duration T_{tx} are shown at the figure's bottom. This illustration is based on Figure 7 in [52].

the MCU of our reference platform, the TelosB – and occurs, because the MCU *"completes the execution of the current instruction before starting to serve the interrupt"* [52]. Glossy ensures that the MCU executes exactly the same number of instructions – corresponding to 97 clock cycles – on all nodes during T_{sw}.

However, besides a fixed *number of instructions* I that must elapse during T_{sw}, "T_{sw} *is still not constant: there is a variable delay in the transfer of digital signals between [the radio and the MCU clock]"* [52]. In other words, the falling SFD pin at the end of a packet reception is set at the rising edge of the radio's clock that runs with frequency f_r. However, the MCU detects the rising SFD pin at the rising edge of its own clock, i. e., a DCO that runs at frequency of f_{DCO}. Because these two clocks, i. e., the radio clock and the DCO, run at different frequencies, there is variable and unpredictable delay in T_{sw}. The ratio between this delay and the length of an MCU clock cycle is indicated in Glossy [52] as the factor k_p, *"the fraction of the DCO period* $1/f_{DCO}$ *required at the MCU to sample the SFD transition at the end of the packet reception. Given that the radio clock and the DCO run completely unsynchronized, the initial offset* k_p *is a continuous random variable uniformly distributed in the interval* $0 < k_p \leqslant 1$*"* [52]. By knowing the frequencies f_{DCO} and f_r, the number of instructions I and neglecting clock drift, the equation for computing T_{sw} is given by [52] with

$$T_{sw} = \frac{1}{f_r} \cdot \left\lceil (I + k_p) \cdot \frac{f_r}{f_{DCO}} \right\rceil. \tag{3.1}$$

The resulting value for T_{sw} *"is a discrete random variable with granularity* $1/f_r$*. The number of possible discrete values for* T_{sw} *and their distribution depend on the number of DCO ticks* I*"* [52].

With $f_r = 8\,\text{MHz}$ and $f_{DCO} = 4\,194\,304\,\text{Hz}$ the (theoretical) maximal distribution of T_{sw} on different nodes results in two possible values that are 0.125 µs [52] apart, which is sufficient to ensure a temporal displacement of $\Delta_{Td \leqslant 0.5\mu s}$. This, however, only applies when the DCOs of synchronously transmitting nodes run at the exactly same frequency. In practice, however, these clocks typically run on different nodes with different frequencies. In the following Section 3.2 and in Section 3.4, we show that these frequency deviations lead to significant differences in the duration of actual software delay T_{sw}.

3.2 IMPACT OF THE MCU CLOCK FREQUENCY ON THE SOFTWARE DELAY

The MCU in low-power nodes is often sourced by an unstable running DCO[2]. In fact, the DCO is expected to tick at a pre-specified, nominal frequency. Manufacturing issues, the node's by-form (see Section 2.1.2), or temperature, humidity, and voltage differences, however, cause the actual frequency to vary from node to node [18, 70]. Thus, the actual frequency of the DCO may vary significantly from the target value, even when the DCO is frequently calibrated by a stable 32 kHz crystal. For example, the MSP430f1611 – the MCU found on the TelosB platform – runs with a nominal frequency of 4 194 304 Hz [70]. Further, frequency deviations of the DCO due to temperature drifts are quantified with −0.38 %/°C [70]. Let us assume that two nodes are exposed to temperature differences of 20 °C. As a consequence, the DCO frequencies of these two nodes differ by 7.6 %. While the clock of one node can be assumed to run at 4 194 304 Hz (i. e., ~238 ns per tick), the other node would have a clock running at 3 875 537 Hz (i. e., ~258 ns per tick). While 97 ticks at the first frequency correspond to 23.13 µs, they instead result in 25.03 µs if the second frequency is considered. The *difference* Δ between these two delays, (i. e., Δ1.9 µs) is far higher than Δ0.5 µs, which is required to ensure constructive interference to occur.

3.3 FLOCK: ON-THE-FLY CLOCK OFFSET COMPENSATION

As discussed in the previous section, the actual duration of the software delay T_{sw} in Glossy can deviate across nodes, because the node's DCO frequency can drift unpredictably from the nominal frequency. The result is a clock offset among the nodes.

In this section, we address this problem and present our solution called Flock[3]. Flock compensates for this clock offset without introducing additional communication overhead in the network with a rather simple principle of operation. In short, while in Glossy the

2 This section is based on Section II-C of our previous work published in [21].

3 This section and all subsequent subsections are an extended version of Section III of our previous work published in [21].

MCU executes a fixed number of instructions during T_{sw} (97 clock ticks), Flock adjusts the number of MCU instructions individually for each node to account for clock frequency deviations of the DCO.

3.3.1 *Flock: How it works*

Flock needs a reliable estimate of either the actual DCO frequency or the DCO's frequency deviation from the nominal value to determine the number of software instructions that must be executed during T_{sw}. To obtain this estimate, Flock makes use of the radio clock as a time reference. The radio clock is highly accurate and runs with a higher resolution than the DCO in our reference platform, the TelosB. In particular, the maximum drift rate of the radio clock is at most ± 40 ppm according to the IEEE 802.15.4 standard [67]. The radio clock is responsible for the time instance at which the SFD pins are set during reception start and end. Thus, the rationale behind Flock is to use the time interval between the SFD pin transitions, indicated as T_{rx} in Figure 3.1, as reference to determine the frequency deviation of the DCO. T_{rx} can be assumed to be reliable, due to the high accuracy of the radio clock. Further, the duration of T_{rx} can be derived from the received packet, and more precisely, the packet length field (see Section 2.3.1).

In the following, we briefly provide a worst-case consideration to justify the use of T_{rx} as reference in Flock. The data rate for IEEE 802.15.4 at 2.4 GHz using O-QPSK modulation is 250 kbit/s. Thus, the transmission of a 9 byte Glossy message, consisting of a 1 byte length field, and an 8 byte MPDU including a 1 byte Glossy header, a 4 byte payload, a 1 byte relay counter c and the 2 byte FCS, takes 288 µs. Let us assume that the radio clock has a maximum drift of ± 40 ppm. The resulting error on T_{rx} is thus ± 11.52 ns ($288\,\mu s \cdot \pm 40 \cdot 10^{-6}$). Consequently, we can assume that T_{rx} is equal for all nodes, except for a few nanoseconds. This is still a factor of 23 below the required 0.5 µs for achieving constructive interference. In addition, ± 40 ppm is the maximum acceptable drift rate for the radio clock over its entire lifetime and operating conditions, including fluctuating voltages and temperatures. Thus, the actual drift rate is much smaller than ± 40 ppm, as we can confirm through our experiments in Section 3.4.2. We also observe that the frequency of the DCO remains stable in the interval $T_{rx} + T_{sw}$, which eliminates another potential error source.

In the following, we indicate with E_{rx} the number of clock cycles that elapse during T_{rx} under the assumption that the DCO runs at the nominal frequency f_{DCO}. Accordingly, with E^*_{rx} we indicate the number of DCO clock cycles when the DCO runs at its actual frequency, indicated as f^*_{DCO}. We further indicate with I the number of clock cycles that must elapse during T_{sw} when the DCO frequency is f_{DCO}. By knowing E_{rx}, E^*_{rx}, and I, we can compute

the number of clock cycles I^* that must elapse during T_{sw} when the DCO frequency is f^*_{DCO} with:

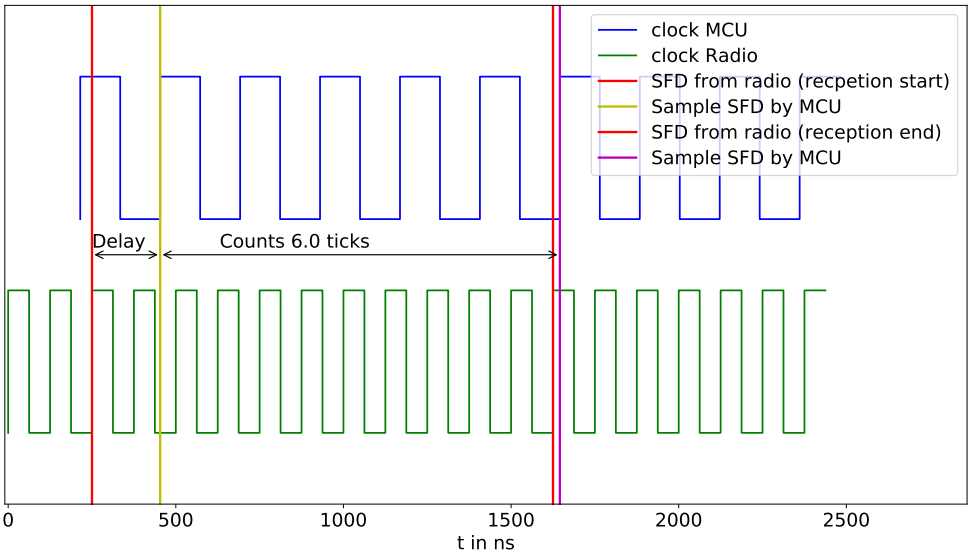
$$I^* = \left\lfloor \frac{E^*_{rx}}{E_{rx}} \cdot I \right\rceil. \tag{3.2}$$

In Equation 3.2, we denote with $\lfloor\rceil$ the "nearest integer function", i. e., the right side of the equation must be rounded to the closest integer. Using the equation above, Flock computes the anticipated value of I^* for a given I. In the following we detail how Flock derives the values for E^*_{rx} and E_{rx} of Equation 3.2.

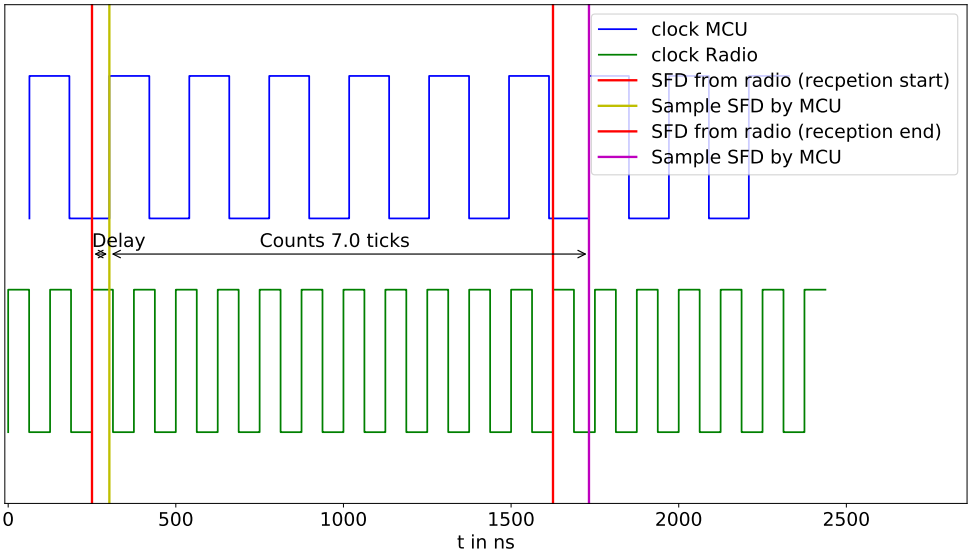
3.3.2 *Counting* E^*_{rx} *and computing* E_{rx}

In order to determine the value of E^*_{rx}, we let the MCU count the number of DCO clock ticks that elapse during T_{rx}. More precisely, we exploit the capability of the TelosB to timestamp the transitions of the SFD pin. Thus, we capture a timestamp when the SFD pin rises at the beginning of a packet reception, and another timestamp at the falling SFD pin at the end of a packet reception. Accordingly, E^*_{rx} is the result of the difference of these two timestamps. Since the MCU timestamps are given in clock ticks, the corresponding value for E^*_{rx} is an integer value.

In the following, we provide a more detailed description of how Flock counts the MCU clock ticks to derive E^*_{rx}. Figure 3.2 illustrates on two examples (Figure 3.2b and Figure 3.2a) how E^*_{rx} is determined. In particular, Figure 3.2 shows the radio clock (colored in green) and the corresponding SFD pin transitions (red lines) as well as the MCU counting the clock ticks (colored in blue) that elapse during the SFD transitions. Just as it occurs in Glossy when determining T_{sw} (see Section 3.1), the radio clock runs at frequency f_r and sets the SFD pin at its rising edge (red line) while the MCU detects the rising SFD pin at the rising edge (yellow line) of its own clock , the DCO, running at its own speed. Thus, a variable and unpredictable delay originates, illustrated in Figure 3.2 with "Delay". The ratio of this delay and the length of an MCU clock cycle is indicated with k_p, as discussed in Section 3.1. The value of k_p causes the MCU to count different clock cycles for the same duration of T_{rx}. Moreover, the MCU detects the transition of the SFD pin on its next clock tick, thus, it actually counts one additional clock cycle. Figure 3.2 illustrates both facts: k_p causing the MCU to count different values for the same T_{rx}, and the MCU counting one clock cycle in addition. For example, in Figure 3.2b, there are five clock cycles between the SFD transitions. However, the MCU detects the falling SFD pin at the end of the reception at the sixth clock tick. In Figure 3.2a, k_p causes the MCU to count seven ticks in total.



(a) MCU counts 6 ticks.



(b) MCU counts 7 ticks.

Figure 3.2: Counting E^*_{rx}. There is a variable and unpredictable delay between the time instant the radio clock sets the SFD pin at its rising edge and time instance the MCU detects the active SFD pin at the rising edge of its own clock, the DCO. The length of this delay varies and causes the MCU to count different values for E^*_{rx} that occur with different probabilities.

Assuming a nominal frequency f_{DCO}, we can use the procedure described above to compute the value for E_{rx}.

$$E_{rx} = \lceil (T_{rx} \cdot f_{DCO}) + k_p \rceil + 1. \tag{3.3}$$

In rare cases, when the product of $T_{rx} \cdot f_{DCO}$ results in an integer number, there is only one possible value for E^*_{rx}. Otherwise, E_{rx} assumes one of two possible values (e. g., 6 ticks or 7 ticks as shown on the example in Figure 3.2) that occur with different probabilities. For computing I^* in Equation 3.2, Flock selects the value that occurs with *higher probability* E^{high}_{rx}. We compute E^{high}_{rx} by first determining the occurrence probability of the *bigger value* E^{max}_{rx} of the two possible values (e. g., 7 ticks in Figure 3.2). Because k_p is a random variable that is uniformly distributed in the interval $0 < k_p \leqslant 1$, the probability of E^{max}_{rx} can be computed as follows:

$$P(E^{max}_{rx}) = (T_{rx} \cdot f_{DCO}) - \lfloor (T_{rx} \cdot f_{DCO}) \rfloor. \tag{3.4}$$

Lastly, we can determine E^{high}_{rx} in dependence of the probability of E^{max}_{rx} with:

$$E^{high}_{rx} = \begin{cases} \lceil (T_{rx} \cdot f_{DCO}) \rceil + 1 & \text{if: } P(E^{max}_{rx}) < 0.5 \\ \lceil (T_{rx} \cdot f_{DCO}) \rceil + 2 & \text{if: } P(E^{max}_{rx}) \geqslant 0.5\ . \end{cases} \tag{3.5}$$

3.3.3 *Theoretical analysis on the distribution of* T_{sw}

Flock enables protocols building upon synchronous transmissions to operate more robustly in harsh environments and further allows such protocols to include intermediate operations during T_{sw}, e. g., for processing of incoming packets or channel switching. In this section, we theoretically analyze the distribution of T_{sw} by also considering the number of instructions during T_{sw} as well as the duration of T_{rx}, and thus, the packet length.

The resulting software delay T_{sw} in Flock after combining Equation 3.1 and Equation 3.2 can be computed with

$$T_{sw} = \frac{1}{f_r} \cdot \left\lceil \left(\left\lfloor \frac{E^*_{rx}}{E_{rx}} \cdot I \right\rfloor + k_p \right) \cdot \frac{f_r}{f^*_{DCO}} \right\rceil. \tag{3.6}$$

We know from the previous Section 3.3.2 that E_{rx} may result in one or two values, even in the absence of clock offset. With E^{high}_{rx} we select for Equation 3.2 the value for E_{rx} that occurs with the highest probability. There is, however, still a chance of $1 - P(E^{high}_{rx})$ that

the value with lower probability occurs for E_{rx}. Let us assume that the bigger value for E_{rx} is the one that occurs with higher probability, thus, $E_{rx}^{max} = E_{rx}^{high}$. Consequently, we set $E_{rx} = E_{rx}^{max}$ in Equation 3.2. Let us further assume that a perfectly stable running DCO with $f_{DCO}^{*} = f_{DCO}$ is available and that during T_{rx} the MCU counts E_{rx}^{min}, thus, the smaller value of E_{rx}. Because the two values E_{rx}^{max} and E_{rx}^{min} differ by one DCO tick, E_{rx}^{*} becomes $E_{rx} - 1$. After transforming Equation 3.2, it becomes $I^{*} \cdot E_{rx} = I \cdot (E_{rx} - 1)$. This equality holds if $I^{*} < I$. More generally, it applies:

$$I^{*} = I \pm \frac{I}{E_{rx}} \qquad \text{if: } f_{DCO}^{*} = f_{DCO} \tag{3.7}$$

Thus, the difference between I and I^{*} is $\frac{I}{E_{rx}}$ DCO ticks, also when neglecting clock drift. There are now three observations that we can make:

1. Glossy sets $I = 97$ to achieve *"the theoretical lower bound of only two possible values for* T_{sw}*"* [52] that are $1/f_r$ apart. We obtain from Equation 3.6 and Equation 3.7 that Flock achieves a theoretical lower bound of two possible values for T_{sw} only when the product $T_{rx} \cdot f_{DCO}$ results in an integer value. Otherwise, the theoretical low bound of Flock is three possible values for T_{sw} with granularity $1/f_r$.

2. From Equation 3.7 we observe that with a fixed E_{rx} and increasing I – i. e., having a fixed packet length while introducing additional instructions during T_{sw} – the deviation between I^{*} and I increases. The consequence is a higher range of values for T_{sw}. In other words, a large processing delay in Flock may also exceed a temporal displacement of $\Delta_{Td \leqslant 0.5\mu s}$ even in the absence of clock offset.

3. We further read from Equation 3.7 that with a fixed I and increasing E_{rx} – i. e., a fixed number of instructions during T_{sw} and an increasing size of transmitted packets – the deviation between I^{*} and I decreases. As a result, also the number of possible values for T_{sw} decreases. This implies that to be able to support intermediate operations during T_{sw}, also T_{rx} and thus, the packet length must be increased in Flock.

In the next section, we evaluate Flock in various scenarios. Among others, we confirm the observations that we made within the current section – Flock supports a large amount of instructions I when adjusting the packet length – even in the presence of clock offset variations.

3.4 EVALUATION OF FLOCK

This section shows the ability of Flock to successfully compensate for clock offsets among synchronously transmitting nodes[4]. In Section 3.4.1, we first simulate the distribution of T_{sw} with and without Flock for different packet sizes and numbers of instructions using Python. Afterwards, we report in Section 3.4.2 our evaluation results when running Glossy with and without Flock on real nodes under different temperature conditions. Finally, we compare in Section 3.4.3 the performance of synchronous transmissions with and without Flock with respect to reliability, latency, and energy-consumption in a controlled lab environment.

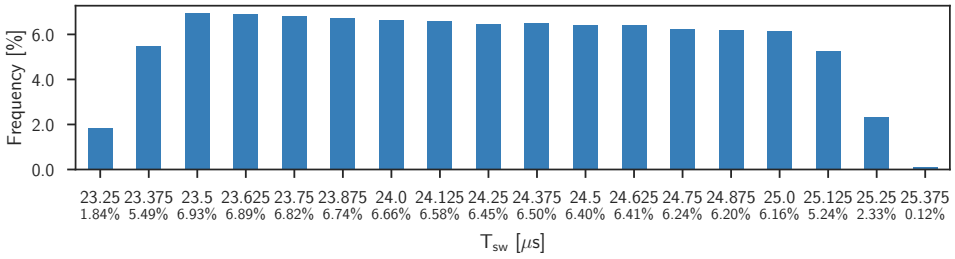
3.4.1 *Performance of Flock in simulations*

In the first set of experiments, we investigate the distribution of T_{sw} with and without Flock in different settings through simulations in Python. In particular, we are interested in the impact of the packet length and the number of instructions I on the distribution of T_{sw} when the DCO frequency of nodes deviate from the nominal value. This set of experiments confirms that even the small number of instructions I used in Glossy may cause to exceed the temporal displacement of $\Delta_{Td\leqslant 0.5\,\mu s}$ when DCO frequency deviations occur. Afterwards, we show that Flock compensates for frequency deviations of the DCO even when a high number of instruction cycles I – allowing, e. g., for processing operations – is used by changing the transmitted packet length. This is conform with the observations from Section 3.3.3.

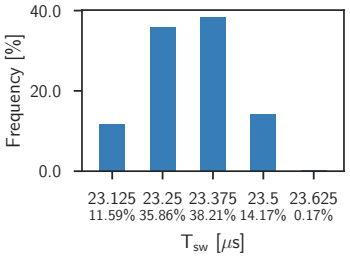
EXPERIMENTS We start our simulation with $I = 97$, i. e., the number of instruction executed by the MCU during T_{sw} in Glossy. To simulate a temperature difference between 0 °C and 20 °C, we randomly vary f^{*}_{DCO} between 4 194 394 Hz and 3 875 537 Hz. This corresponds to a frequency deviation between 0 % and 7.6 % from the nominal value, as discussed in Section 3.2. We repeat each simulation 1,000,000 times. For computing T_{sw}, we use Equation 3.1 for Glossy without Flock and Equation 3.6 with Flock.

RESULTS Figure 3.3a shows the distribution of T_{sw} for Glossy without Flock. In the figure, T_{sw} spreads from 23.25 µs to 25.375 µs, resulting in a difference of Δ2.215 µs with 18 values that are 125 ns apart. The latter is because of the radio clock running at 8 MHz, which results in a resolution of 125 ns (see Section 3.1). This result confirms that even the small number of instructions in Glossy causes a distribution of T_{sw} that may exceed the temporal displacement of $\Delta_{Td\leqslant 0.5\,\mu s}$. Next, we repeat the simulation with Flock and

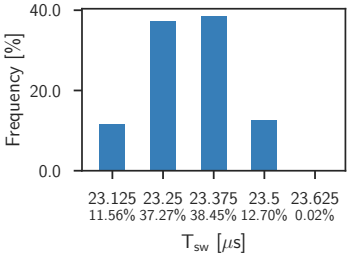
4 This section and the subsequent subsections are an extended version of Section IV of our previous work published in [21].



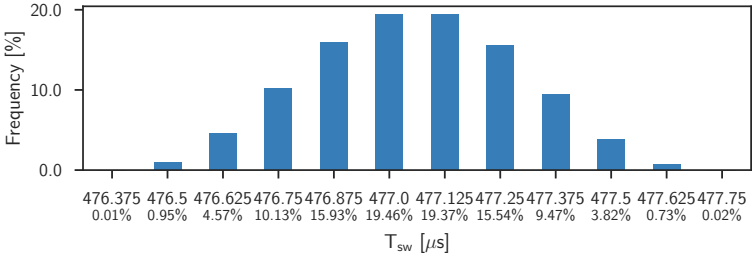
(a) Glossy: I = 97.



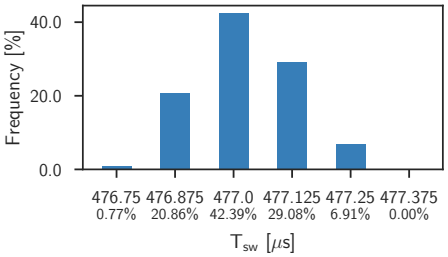
(b) Flock: I = 97, packet length = 9 byte.



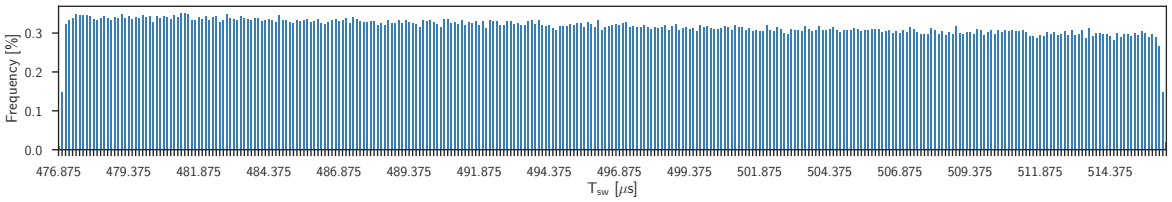
(c) Flock: I = 97, packet length = 128 byte.



(d) Flock: I = 2000, packet length = 9 byte.



(e) Flock: I = 2000, packet length = 128 byte.



(f) Glossy: I = 2000.

Figure 3.3: Simulated distribution of the software delay T_{sw}. Flock efficiently compensates DCO frequency deviations among synchronously transmitting nodes. Adjusting the packet length helps in reducing the amount of possible values for T_{sw}.

a packet length of 9 byte. Figure 3.3b shows the distribution of T_{sw} in this scenario. T_{sw} spreads only over 5 values with a resulting difference of $\Delta 0.625\,\mu s$. More precisely, T_{sw} distributes within $\Delta 0.5\,\mu s$ in 99.83 % of the simulation runs. Thus, this confirms that Flock can achieve constructive interference by efficiently mitigating clock frequency deviations among nodes.

In the following, we simulate the impact of the packet length and the number of instructions I on the distribution of T_{sw} in Flock. First, we leave I unchanged and increase the packet length to 128 byte. The resulting distribution of T_{sw} is shown in Figure 3.3c. In this scenario, the packet length has only marginal effect when comparing the distribution with Figure 3.3b. In particular, the frequentness of T_{sw} being within $\Delta 0.5\,\mu s$ increases to 99.98 %. This is because a packet length of 9 byte is sufficient to compensate clock frequency deviations for 97 clock ticks. Next, we change I to 2000 ticks, i. e., the time in Chaos [90] to compute a "maximum" function across 139 nodes, and the packet length to 9 byte. The simulation result is shown in Figure 3.3d. T_{sw} spreads over 12 values with a difference of $\Delta 1.375\,\mu s$. Thus, compared with Figure 3.3b, Figure 3.3d shows a larger distribution of T_{sw} ($\Delta 0.625\,\mu s$ vs. $\Delta 1.375\,\mu s$). This is expected and already discussed in Section 3.3.3 (Observation 2). In particular, from Equation 3.7 with $I = 2000$ and $E_{rx} = 1209$, i. e., the number of clock ticks that the MCU should count during T_{rx}, we can derive that the difference between I and I^* is around 2 DCO ticks. These 2 additional DCO ticks, consequently result in a higher distribution of T_{sw}. To verify observation 3 in Section 3.3.3, i. e., the packet length must be increased to let Flock support intermediate operations, we change the packet length to 128 byte. Figure 3.3e shows the result. The distribution of T_{sw} decreases to $\Delta 0.625\,\mu s$ with larger packets. Further, constructive interference can be achieved in 99.24 % of the simulation runs. Figure 3.3f shows T_{sw} when running Glossy with $I = 2000$ for reference. T_{sw} can take values over an interval of $\Delta 39.5\,\mu s$ Thus, even with small packets, the distribution of T_{sw} using Flock is smaller than without Flock.

3.4.2 *Quantifying the effects of temperature on the software delay*

In the previous section, we have evaluated through simulations the effect of Flock on the distribution of T_{sw}. In this section, we present the results of experiments performed in a controlled environment with real nodes. We find in this set of experiments that Flock enables to achieves constructive interference in 98 % of the cases, even when the nodes are exposed to temperature differences of 30 °C. More precisely, Flock reduces the distribution of T_{sw} from $\Delta 2.124\,\mu s$ to $\Delta 0.75\,\mu s$. Thus, Flock increases the probability of synchronously transmitted packets to precisely overlap.

EXPERIMENTS We embedded Flock in Glossy's publicly available source code[5] and run it on TelosB nodes[6]. To account for different frequency offsets, we implement a look-up table that uses E^*_{rx} as input. We pre-calculate for different values of E^*_{rx} the corresponding number of instructions I^*. Thus, the output of the look-up table is a number of No Operations (NOPs) to add during T_{sw}, which corresponds to I^*. The consequence is that Flock within Glossy requires additional instructions. Thus, in our implementation the total number of instructions is $I = 112$.

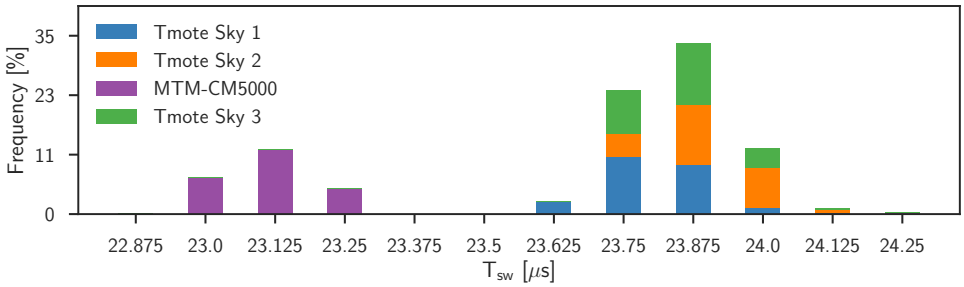
We first measure T_{rx} for 9 byte packets – the default packet size in Glossy – using an oscilloscope. We find that T_{rx} is 288.6 µs instead of 288.0 µs as computed in Section 3.3.1. As stated in [72], the CC2420 – the radio chip used in the TelosB – adds a processing delay during packet reception. We assume that the discrepancy between measurements and theory is caused by this delay. We have not observed such delay on sending nodes. Nonetheless, this delay has no effect on the design of Flock. However, it has to be considered when computing the value for E_{rx}.

We measure the distribution T_{sw} on four nodes acting as receivers by connecting their SFD pins to an oscilloscope. Three receivers are Tmote Sky nodes and one is an MTM-CM5000 node. As discussed in Section 2.1.2, both platforms, Tmote Sky and MTM-CM5000, are by-forms of the TelosB platform that only differ in their manufacturer and the hardware revisions of the components they are equipped with. We enforce different frequency offsets of the DCO by repeating the experiments under three different temperature conditions: room temperature (22 °C), cold (10 °C) and hot (40 °C). These temperature ranges are within the operating conditions of the nodes that are specified with −40 °C and 85 °C [115]. We generate the cold and hot conditions with a refrigerator that can be switched to either cooling or heating. After carefully placing the four receivers into the refrigerator we wait a few minutes before starting the experiments, to allow the nodes to acclimatize. The initiator – the node that starts the communication and disseminates the packets – operates at room temperature. We set the transmission power to 0 dBm, i. e., the maximum transmit power for TelosB nodes, and make sure that the reception reliability is close to 100 %. We further ensure that the DCO is calibrated with the stable 32 kHz clock before each Glossy-flood to ensure a fair evaluation. We first run Glossy at room temperature without Flock. After collecting over 2,000 samples, we repeat the experiment, but expose the four receivers to a cold condition and then to a hot condition. Finally, we run the experiment again, but with Flock enabled.

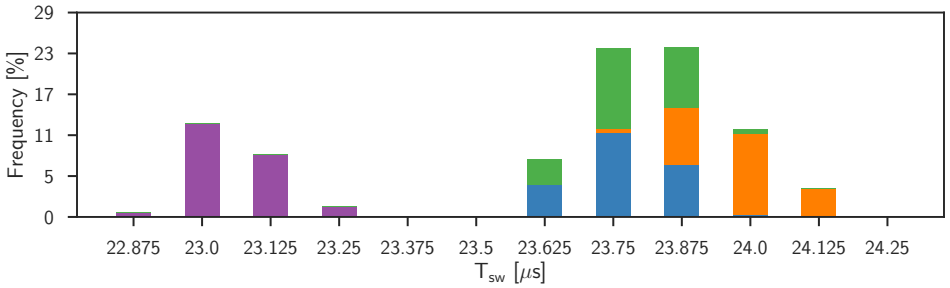
RESULTS Figure 3.4 shows the resulting distribution of T_{sw} in the different temperature conditions for Glossy and Figure 3.5 shows the results for Flock. When comparing Figure 3.4a (i. e., Glossy at room temperature) and 3.4b (i. e., Glossy exposed to the cold

5 http://sourceforge.net/p/contikiprojects/code/HEAD/tree/ethz.ch/glossy/

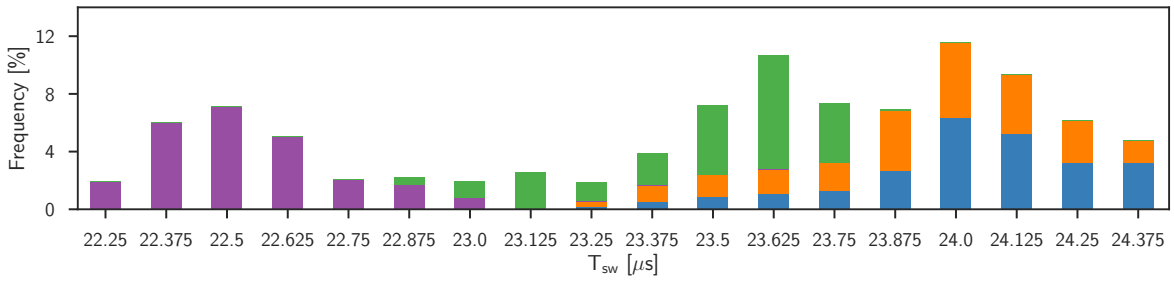
6 The source code of Flock is publicly available at https://github.com/martinabr/flock.



(a) Glossy: 22°C.



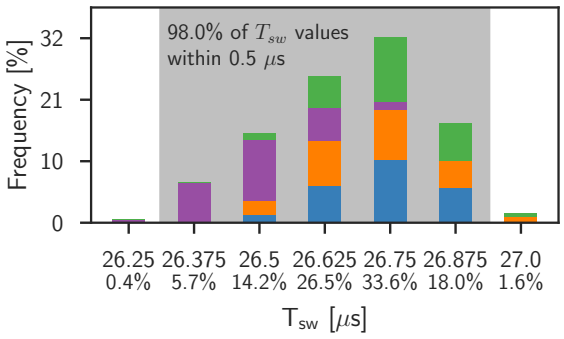
(b) Glossy: 10°C.



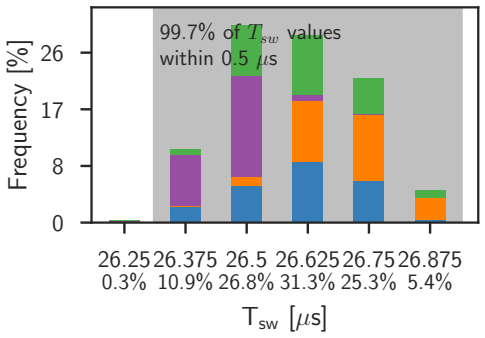
(c) Glossy: 40°C.

Figure 3.4: Experimentally retrieved distribution of the software delay T_{sw} in Glossy. Temperature as well as the nodes' by-form heavily affect the distribution of T_{sw}. In particular, T_{sw} spreads over Δ2.125 µs for over 2,000 samples at 40 °C.

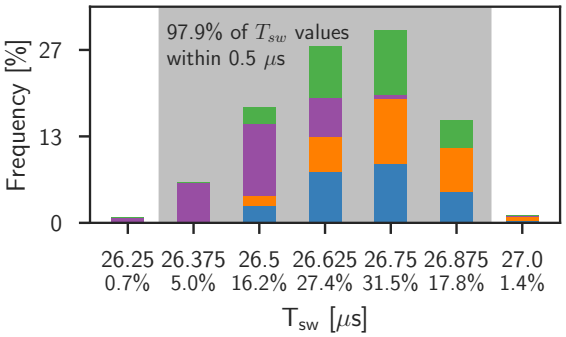
condition), the distributions of T_{sw} for the Tmote Sky nodes are close together while the distribution of T_{sw} for the MTM-CM5000 strongly differs from the distribution of the Tmote Sky nodes. We believe that the cause for these differences is (1) the nodes by-form, and (2) the age of the nodes. In both temperature conditions the resulting distribution for T_{sw} spreads over Δ1.375 µs. In the hot environment the distribution of T_{sw} expands to Δ2.125 µs, distributed over 18 values with a distance of 125 ns. It has been already observed by *Boano et al.* that *"the impact of temperature on the radio chip is considerable, and the higher the temperature is, the lower are the signal strength and the link quality"* [16]. Obviously, this



(a) Flock: 22°C.



(b) Flock: 10°C.



(c) Flock: 40°C.

Figure 3.5: Experimentally retrieved distribution of the software delay T_{sw} *in Flock.* Flock is able to reduce the spreading of T_{sw} to only $\Delta 0.75$ µs for the same temperature conditions. However, the implementation of Flock requires a few more instructions I compared to Glossy, so that Flock has a slightly higher software delay.

also applies for other components that the nodes are equipped with, especially the DCO. Constructive interference is not guaranteed in all three temperature environments.

Figure 3.5 depicts the distribution of T_{sw} with Flock. When the nodes are exposed to room temperature conditions (Figure 3.5a) and warm conditions (Figure 3.5c) T_{sw} spreads over $\Delta 0.75\,\mu s$. In the cold condition (Figure 3.5b) T_{sw} spreads over $\Delta 0.625\,\mu s$. In all three temperature conditions the distribution of T_{sw} is within 0.5 µs in 98 % of the cases, the required temporal displacement to achieve constructive interference. The gray area in the plots marks the range of values for T_{sw} for which constructive interference can be achieved. In particular, this range spreads from 26.375 µs to 26.875 µs in all temperature conditions. Thus, we can conclude that constructive interference can also be achieved when the nodes are located in mixed temperature environments.

As mentioned before, Flock requires a few software instructions for itself. The average value for T_{sw} increases from 23.56 µs for Glossy with Flock disabled to 26.63 µs with Flock enabled. Thus, the average time the radio is active should be higher in using Flock compared to "standard" Glossy. However, we show in the following experiments that enabling Flock allows still to achieve a lower radio-on time during a Glossy flood.

3.4.3 *The performance of Flock in a controlled environment*

In this set of experiments, we investigate the effects of clock offset and the impact of Flock on the performance of synchronous transmissions with respect to reliability, latency, and radio-on time as measure for the energy-consumption. We find that when the DCO of the nodes deviate by 1.5 %, Flock enables the reliability of synchronous transmissions to increase by up to 3.7 % and also reduces latency as well as the time the radio is active during a packet flood.

EXPERIMENTS We use the code from the previous experiment and deploy it on three Tmote Sky nodes – two senders and one receiver. The setup of the nodes is shown in Figure 3.6. We ensure that we only measure the effects of clock offset by eliminating external factors like multi-path propagation that may also influence the temporal displacement of signals and thus, the performance of synchronous transmissions. For this purpose, we connect the nodes at the SubMiniature Version A (SMA) connection sleeves of the antennas via coaxial cables and a tee coaxial adapter as shown in Figure 3.6. We assign sender 1 and the receiver a nominal MCU frequency of 4 194 304 Hz. To simulate a frequency offset of 1.5 % we set the nominal frequency of sender 2 to 4 131 389 Hz. We set the transmit power of the nodes to −25 dBm and ensure that both receivers intercept packets with approximately the same power level (in our case −66 dBm). We perform this set of experiments at room temperature and run first "standard" Glossy and afterwards Glossy with Flock. We repeat each experiment three times, with 2,100 packet floods for each experiment. Afterwards we repeat the experiment with a transmit power of −29 dBm. N_{tx}, i. e., the number of packet

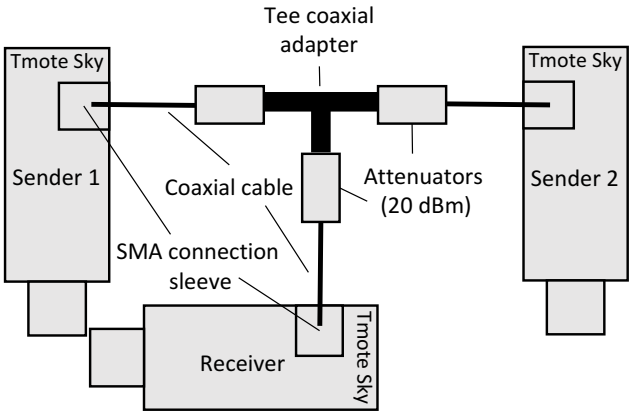


Figure 3.6: Experiment setup for Flock in a controlled environment. We reduce multi-path propagation and other channel effects by connecting the nodes via coaxial cables at their SMA connection sleeves.

retransmissions, is set to 3 in all experiments. Triggered by the receiver, the two senders synchronously transmit a packet.

At the receiver, we measure three key metrics: reliability, latency, and the radio-on time. The *reliability* is the ratio of missed and total packets. The *latency* indicates the time between the transmission of a packet and its first reception. The *radio-on time* is the time the radio is active during a packet flood. The receiver further collects information about dropped packets (including retransmitted packets) due to bit-flips caused by destructive interference: The parameter *wrong length* is the ratio of the number of packets that are dropped due to a false length and the total number of dropped packets. A length is considered false when it is either smaller than 2 byte, which is the size of the footer (see Section 2.3.1), or greater than the maximum packet length of 127 byte, specified by the IEEE 802.15.4 standard [67]. Glossy distinguishes itself from other coexisting protocols by a header field with value `0xa0`. The parameter *wrong header* indicates the ratio of packets that are dropped due to a value that differs from the expected one and the total number of dropped packets. Packet drops due to failed CRCs are indicated with the *wrong FCS* parameter.

RESULTS Table 3.1 summarizes the results for these of experiments. The results confirm that Flock improves the overall performance of synchronous transmissions. In particular, Flock increases reliability and reduces latency and radio-on time for both transmit powers. As mentioned in Section 3.4.2, the implementation of Flock requires a few instructions during T_{sw}. The total number of instructions during T_{sw}, thus, increases from $I = 97$ to $I = 112$. However, Flock is still able to achieve a lower radio-on time and a lower latency compared to Glossy. We believe that without Flock the first packets of a Glossy flood collided destructively due to the temporal displacement exceeding $\Delta_{T\,d\leqslant 0.5\,\mu s}$ and thus, the packets were corrupted and dropped by the nodes. The first packet is, hence, received later,

and thus, the latency increases. The nodes still transmit a packet $N_{tx} = 3$ times, which results in an increased radio on time.

Table 3.1: Experimental results of Flock in a controlled environment. When the nodes' DCO clocks deviate by 1.5 %, Flock is able to increase Glossy's reliability by 3.7 % at a transmit power of −29 dBm.

Scenario	*Tx power [dBm]*	*Reliability [%]*	*Latency [ms]*	*Radio-on time [ms]*	*Wrong length [%]*	*Wrong header [%]*	*Wrong FCS [%]*
Synchronous transmissions w/ Flock	-25	100.0	2.163	4.586	13.66	6.05	80.28
	-29	98.74	2.464	6.986	13.21	5.98	80.81
Synchronous transmissions w/o Flock	-25	99.80	2.226	5.197	17.0	0.0	83.0
	-29	95.18	2.834	10.184	11.88	5.85	82.27

3.5 RELATED WORK

Several authors have investigated the effects of temperature and humidity on the reception reliability in single-link networks[7]. For example, *Boano et al.* [15, 16] showed experimentally that the reliability decreases by 30 % when the nodes are exposed to temperature differences of 30 °C. *Wennerström et al.* [178] confirmed these results and also showed that the reliability decreases especially at high temperatures. These findings are consistent with our observations in Section 3.4.2. However, Flock aims to compensate clock offset that is caused e. g., by temperature differences among nodes.

Clock offset or clock drift compensation has been extensively studied in the context of clock synchronization. For example, RBS [44], TPSN [56], FTSP [109], RITS [142], RATS [88], and PulseSynch [92] are protocols that aim to time-synchronize the nodes in a low-power wireless network. The mentioned protocols achieve a synchronization accuracy in the order of microseconds. In TPSN and RBS, e. g., the average synchronization error between two nodes in a single hop network is 16.9 µs and 29.1 µs, respectively [56]. RITS, RATS and PulsSynch have an average error of 5.3 µs, 8.0 µs, and 2.06 µs, respectively. FTSP achieves an average error of only 1.48 µs [109] in a one-hop scenario and 0.5 µs per hop in a six-hop network using MAC layer time-stamping. However, in the latter scenario, FTSP has an average convergence time of 10 minutes.

Constructive interference requires the temporal displacement of identical signals in the order of sub-microseconds, and more specific, within 0.5 µs [52] or even less [84]. Glossy, e. g., achieves $\Delta_{Td \leqslant 0.5\,\mu s}$ with high probability, by minimizing the number of MCU instructions and letting each node execute the same amount of instructions, i. e., 97 MCU ticks, between the end of a packet reception and the retransmission request. Due to clock fre-

7 This section is an extended version of Section V of our previous work published in [21].

quency deviations among the nodes, the actual time to execute these 97 clock cycles may differ from node to node. Flock provides a practical solution to compensate these clock frequency deviations on-the-fly without introducing additional message overhead. Furthermore, it allows intermediate instructions to be executed, and thus, relaxes an integral part of Glossy.

Recently, *König et al.* [84] presented an approach to reduce the temporal displacement in Glossy. In particular, they account for the propagation delay, the clock synchronization error, and the transmission timing error, i. e., *"the delay the node starts the transmission after a given desired local time"* [84]. In their approach, the authors achieve constructive interference in over 30 % of the cases while Flock obtains constructive interference in over 98 %.

A proposed technique to limit the effects of DCO deviations is to use the Virtual High Resolution Time (VHT) [146]. VHT utilizes the fact that low-power platforms are typically equipped with more than one clock. For example, the TelosB node, which we use in this thesis as reference platform, features a 4.1 MHz and a 32 kHz clock. While the first one is a highly unstable, high-power clock that has a high resolution, the latter one is a highly accurate clock that draws little power consumption and offers a low resolution. VHT combines both clocks to enable low-power and accurate high-resolution time-stamping. Several synchronous transmission-based approaches utilize VHT [84, 120, 130], but also denote limitations [120, 130]. In particular, the accuracy obtained using VHT is mostly insufficient to ensure a temporal displacement of $\Delta_{Td \leqslant 0.5\mu s}$.

3.6 SUMMARY

In this chapter we discussed Flock. Flock aims to compensate clock frequency deviations among synchronously transmitting nodes that are caused, e. g., by manufacturing issues, the by-form, or the internal temperature of the nodes. It adapts on-the-fly the number of clock cycles that must elapse during the software delay T_{sw} that is induced in Glossy to compensate for interrupt variations and to trigger the retransmission of a packet. This allows Glossy and Glossy-like protocols to operate in harsh environments and to include additional operations like processing of packets or channel switching while still maintaining constructive interference, and hence, a high probability in receiving a packet. Our evaluation showed that Flock efficiently adapts for frequency deviations of the DCO and increased Glossy's performance in terms of reliability, latency, and radio-on time.

The goal of the work presented in this chapter was to increase the probability of constructively colliding packets, even in harsh environments. Therefore, Flock can be used as low-layer primitive in combination with other protocols based on Glossy-like synchronous transmissions. For example, in the work presented in the following chapter, we utilize Flock in event-based scenarios and more precisely, in the quick yet energy-efficient distribution of small amounts of data.

CHAPTER 4

FAST FLOODING OF SMALL AMOUNTS OF DATA

In the previous chapter, we addressed the effects of frequency deviations among synchronously transmitting nodes on the temporal displacement Δ_{Td}, and thus, on the performance of protocols based on synchronous transmissions. In the following two chapters, we describe how one can use synchronous transmissions as a low-level communication service to support event-based data traffic. In particular, in this chapter, we discuss the quick and energy-efficient distribution of small data packets.

Event-based data traffic occurs frequently in applications for low-power wireless networks. For example, changing the set-point of the Heating, Ventilation and Air Conditioning (HVAC) system in a smart building triggers sporadically events. These events can occur unpredictably at any time and must be forwarded immediately by the nodes to the building management system for further processing. Thus, the nodes have to be awake to be able to forward such events with low latency. However, events can be quite rare. For example, Istomin et al. [73] report that using data prediction in an indoor temperature scenario and dividing time in 30-second slots, an event occurs only in 80 % of the slots. There is thus a discrepancy between latency and energy-efficiency. In order to achieve a low latency, the nodes have to frequently turn their radios on and sample the channel for incoming packets. The more often this sampling is performed – and the longer it lasts – the lower is the latency but the higher is the nodes' energy consumption. And since events are rare, nodes often consume unnecessary energy during idle listening, i. e., listening for potential packets.

In this chapter, we present Whisper to address the challenge described above: Whisper is a network primitive that builds upon synchronous transmissions and provides fast and energy-efficient communication in low-power wireless multi-hop networks. Whisper makes network floods significantly shorter by integrating the flood into a single packet transmission. This (a) reduces the duration of a flood and thereby allows Whisper to reduce radio-on time and energy consumption and (b) enables robust sampling, because there is no "gap" between two packets in Whisper. Whisper can be used as "standalone" dissemination protocol and as service for other protocols, e. g., as an efficient wake-up primitive to reduce the protocols idle listening overhead. The latter is the case in the event-based transmission of large data packets. In such scenarios, the nodes listen to potential packets for a predefined interval before turning the radio off. Using Glossy-like protocols, the length of this interval depends on the packet size, the number of hops in the network,

and N_{tx}. For example, it would take Glossy almost 67 ms to disseminate a 127 byte packet in a 6-hop network with $N_{tx} = 3$. Thus, the nodes have to keep the radio turned on for the same amount of time when no packet has been disseminated. Using Whisper, the nodes keep their radio on for at most a Whisper slot, which is less than 3 ms. Only if they have received a packlet during the Whisper slot, they await the actual data packet afterwards. Otherwise, they keep their radio turned off. Thus, using Whisper, nodes only communicate when they have an event to share. In sum, in case of no data traffic, the nodes are awake using Whisper for only 3 ms instead of 67 ms.

The remainder of this chapter is structured as follows: Section 4.1 introduces the rationale of Whisper. Next, Section 4.2 discusses the design choices of Whisper in detail. We evaluate Whisper and compare its performance to the state-of-the-art in Section 4.3. Section 4.4 presents the related work and discusses key differences of Whisper. Finally, we conclude this chapter in Section 4.5.

4.1 WHISPER: HOW IT WORKS

At its core, Whisper is a communication primitive that allows to flood small amounts of data into a multi-hop network[1]. It can be used, e. g., to disseminate a configuration parameter or to signal all nodes in a network that they must stay awake for helping forwarding incoming possibly large data packets. In the following, we describe the three cornerstones of Whisper's design: packlets, direction-aware channel sampling, and synchronous transmissions.

SIGNALING PACKETS AND PACKLETS In Whisper, a node that needs to send data – hereafter referred to as the *sender* – transmits a *signaling packet* that looks as depicted in Figure 4.1. It consists of several *packlets*, whereas a packlet is a piece of message payload that has the structure of an actual IEEE 802.15.4 packet, including preamble, SFD and footer. Whisper's signaling packet, thus, mimics a train of short, identical packets being sent continuously by the radio, as illustrated in Figure 4.2a.

1 This section is based on Section II of our previous work published in [22].

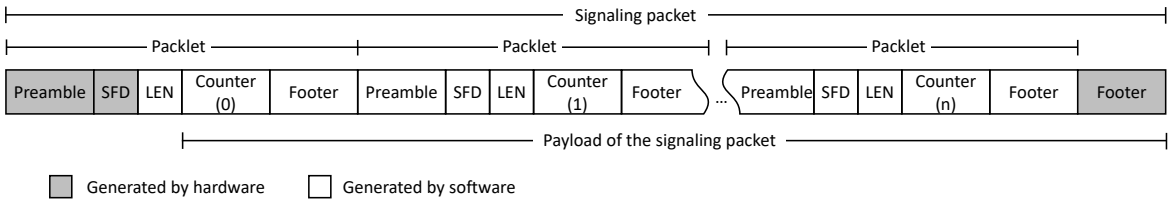
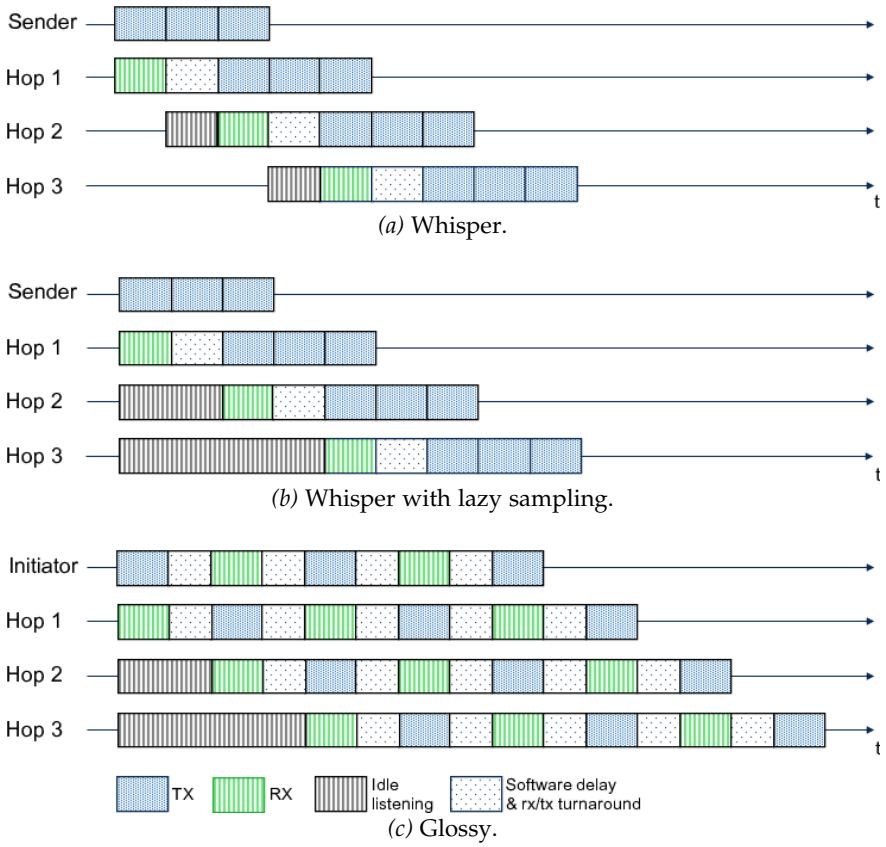


Figure 4.1: Structure of a signaling packet that is used as a "wake-up call". Note that the gray-shadowed footer at the end of the packet is created only when the radio is used in *buffered mode*.



(a) Whisper.

(b) Whisper with lazy sampling.

(c) Glossy.

Figure 4.2: Whisper vs. Whisper with lazy sampling vs. Glossy. Whisper transmits a train of short packlets continuously sent by the radio. In contrast, Glossy's transmit-and-relay scheme leaves "gaps" between two transmissions, caused by the reception of packets, the software delay T_{sw}, and the turnaround from receive to transmit mode (and vice versa) of the radio.

This is a core difference between Whisper and, e. g., Glossy. In Glossy, the initiator – i. e., the node that starts the packet dissemination, as described in Section 2.4.3 – continuously switches between sending and receiving mode and thus, leaves "gaps" between two transmissions, as depicted in Figure 4.2c. The absences of such gaps in Whisper significantly reduces the overall duration of a packet flood and thus, it reduces the time the nodes have to sample the channel for potential packets. In Section 4.2.1 and Section 4.2.7, we provide further details about the design of Whisper's signaling packet and packlets.

SAMPLING STRATEGY For nodes to be able to detect the presence of a signaling packet, they must, indeed, regularly switch their radios on and check the channel for incoming transmissions. The more often this channel check is performed, and the longer

each check lasts, the higher is the duty cycle of the nodes, and thus, their energy consumption. Whisper, and in particular the use of packlets, opens up the possibility to design thrifty sampling strategies. Efficient channel sampling is instrumental to reduce the overall radio-on time, and thus, the duty cycle of nodes.

A straightforward sampling strategy – to which we refer to as *lazy sampling* (see Figure 4.2b) – consists in making all nodes switch their radios on at the beginning of a communication slot. This strategy is used in Glossy and other approaches such as LWB [50] or Crystal [73] and can be used in Whisper, too. When adopting lazy sampling, nodes must wait for an incoming transmission long enough so that the signaling packet from a sender can propagate through the entire network. This can, however, take several milliseconds in a network of several hops and represents a high cost in terms of energy consumption, especially if no packet is transmitted.

Whisper uses an alternative sampling strategy that we call *direction-aware sampling*. This strategy exploits two key observations: (i) in many practical scenarios the network topology is usually fixed or changes slowly, and (ii) data traffic flows in one direction only – e. g., from an initiator to all other nodes in a network in a data dissemination scenario or from a node in the network to a central sink node in case of event-driven data collection scenarios. Thus, nodes can estimate their distance in hops from the sender or destination and switch on their radios only when a signaling packet is likely to "pass-by", as shown in Figure 4.2a. We provide further details on Whisper's sampling strategies in Section 4.2.5 and 4.2.6.

SYNCHRONOUS TRANSMISSIONS To ensure a fast and reliable propagation of the signaling packet, Whisper exploits synchronous transmissions. When a neighbor of the sender turns its radio on, it needs to intercept only one of the packlets to detect the existence of a signaling packet. If no packlet is detected, the node switches its radio off to save energy. If a packlet is instead successfully received, the node keeps its radio on and helps propagating the signaling packet. It does so by joining the ongoing synchronous transmission with its own signaling packet, which is again a single packet made of multiple packlets.

To this end, a node that starts sending a signaling packet must ensure that its own packlets overlap with the packlets that are already being transmitted by other nodes. This means that packlets sent by nearby nodes must fulfill two conditions: (i) they must be identical, and (ii) they must be sent within a temporal displacement of $\Delta_{Td \leqslant 0.5\mu s}$, as schematically illustrated in Figure 4.2a. We discuss in Section 4.2.2 and Section 4.2.3 how Whisper manages to fulfill both of these conditions.

4.2 WHISPER: A CLOSER LOOK

After having briefly presented the main features of Whisper in the previous section, we now provide a more detailed description.[2]. Hereafter, we consider Whisper as wake-up primitive as service for other communication protocols that are deployed in the nodes.[3] In particular, Whisper facilitates the concurrent transmission of several signaling packets, i. e., several senders notify the nodes in a network about incoming bulk data packets. To this end, the nodes need to be synchronized to a common reference. Thus, Whisper assumes that there is an initiator – as it also exists in Glossy – that infrequently synchronizes the network. The synchronization can be provided by the communication protocol that exists besides Whisper, e. g., by using Glossy, or by using implicit time synchronization through Whisper-floods. Indeed, only the initiator is allowed to start a packet transmission in the synchronization slot.

In the following, we first discuss the design of the signaling packet in Section 4.2.1 and present a back-of-the-envelope calculation to show why Whisper has superior performance in comparison to Glossy. We then show how Whisper exploits synchronous transmissions. We detail how Whisper makes the packlets that need to be transmitted identical with ongoing transmissions in Section 4.2.2 as well as how it makes them precisely overlap in Section 4.2.3. In Section 4.2.4 we present a concept for time synchronization with Whisper. Afterwards, we present the two sampling strategies considered in this thesis: lazy sampling in Section 4.2.5 and direction-aware sampling in Section 4.2.6. We present in Section 4.2.7 a fully IEEE 802.15.4-compliant version of Whisper, and discuss potential improvements to make Whisper robust against external interference in Section 4.2.8. Lastly, we discuss the portability of Whisper to other existing radio chips and node platforms.

4.2.1 *The signaling packet*

The design of the signaling packet as shown in Figure 4.1 allows Whisper to achieve a fundamental goal: create a train of packets sent back-to-back without any gaps between consecutive transmissions. We argue that this is an essential stepping stone to (a) simplify timing of synchronous transmissions, (b) enable sampling strategies allowing nodes to sleep efficiently, and (c) reduce the duration of a network-wide flood.

Whisper uses the payload of the signaling packet to simulate several packets being sent back-to-back. This is achieved by filling the payload with a sequence of *packlets*. As

2 This section and the following subsections are an extended version of Section III of our previous work published in [22]. More precisely, we extended this thesis with the following subsections: Section 4.2.4 – Time synchronization, Section 4.2.6 – Computing p_{min} and p_{min}, Section 4.2.6 – Direction-aware sampling in collection scenarios, and Section 4.2.9 – The portability of Whisper.

3 The operation of Whisper for data dissemination is equivalent to the provided description, besides that the size of the packlets increases with the payload length.

illustrated in Figure 4.1, a packlet in Whisper consists by default of five fields: a preamble, a 1-byte SFD, 1-byte length field, a 1-byte payload, and a 2-byte footer with FCS. Nonetheless, the format of the packlet in Whisper is user-configurable and can be flexibly extended to contain, e. g., more data. When a receiver starts listening for incoming packets, it only needs to intercept the preamble and SFD of one of the packlets to detect an ongoing transmission. The multi-header technique presented in [93] inspired our packlet-based design of the signaling packet. The authors concatenated several synchronization headers to mitigate interference, and thus, to increase the chance of detecting an incoming packet. Whisper, instead, uses the same technique to transmit IEEE 802.15.4-compliant packets.

While the IEEE 802.15.4-compliant length of the preamble is 4 byte, in some radios, e. g., the CC2420 [72] of our reference platform TelosB, both the preamble length and the SFD are configurable parameters. To reduce the total length of a signaling packet and thus, decrease the radio-on time of the nodes, Whisper's default implementation sets the length of the preamble to 2 byte. This assumes that Whisper can exploit low-level features of the transceiver and makes it non-IEEE 802.15.4-compliant. Nonetheless, Whisper can operate with a preamble of arbitrary length and can thus, if required, also be used with a preamble of 4 byte. While a longer preamble affects performance, we show in Section 4.3 that Whisper outperforms Glossy also with preamble length of 4 byte.

Given the description above, a packlet in Whisper is by default 7 byte long[4]. Since IEEE 802.15.4 radios transmit at 2.4 GHz at a rate of 250 kbit/s, the transmission of a single packlet lasts 224 µs. The sender sends N_{tx} packlets and its total transmission time with $N_{tx} = 3$ is thus 672 µs. The radio of other nodes in the network is instead active for the duration of $N_{tx} + 2$ packlets. This is because, as illustrated in Figure 4.2a, one packlet is received, during the second packlet the radio switches from receive to transmit mode and then N_{tx} packlets are sent. Thus, when $N_{tx} = 3$, the radio of the other nodes in the network is active for 1.120 ms.

In contrast, nodes running Glossy must keep their radio on for at least 2.304 ms during a flood. Figure 4.2c shows that nodes in Glossy receive or transmit a packet 6 times. Assuming that also Glossy sends packets with a 1-byte payload – and, thus, that a Glossy packet is as long as a packlet– each node actively transmits or receives for 1.344 ms. Glossy must, however, also continuously switch between receive and transmit mode, as illustrated in Figure 4.2c. This rx/tx turnaround of the radio takes 192 µs [67] and nodes must turn the radio from receive to transmit mode 5 times, which adds almost 1 ms (960 µs) of additional radio-on time. The radio-on time of Glossy is thus roughly twice as long as that of Whisper (2.304 ms vs. 1.120 ms) – even though we did not account for Glossy's software delay, which should be added to the rx/tx turnaround time (cf. [52] at Section 5.3). We also did not

4 While this structure seems to induce a high control overhead (6 byte) for a payload of 1 byte, we actually consider a packlet including synchronization header, length and footer field as unity to allow for efficient signaling of pending bulk data. More precisely, this allows nodes to recognize that a channel is busy while being able to distinguish packet exchange from other interference.

consider – neither in the calculation above nor in Figure 4.2 – the guard times that are present in both Whisper and Glossy. A guard interval is usually short[5] and appears only once at the beginning of the idle listening phase. It, thus, has only little influence on the computation presented above.

This back-of-the-envelope calculation shows that the superior performance of Whisper in comparison with Glossy, as we have shown Section 4.3, is mainly due to the fact that Whisper eliminates the gaps between consecutive transmissions. This advantage persists even if Whisper is used with lazy sampling as in this case, the time spent in idle listening is roughly the same as for Glossy.

In the example discussed above, we assume Glossy packets with a payload of 1 byte. The payload of standard Glossy packets is, however, 4 byte: a 1 byte Glossy header, a 2 byte sequence number, and a 1 byte relay counter [52]. We reduce the payload size to 1 byte (we keep only the field *relay_counter*), to avoid penalizing Glossy due to its larger payload size. This also allows us to show that shortening the payload size in Glossy is not sufficient to make it more efficient than Whisper as wake-up primitive.

Even though Whisper targets the dissemination of small amounts of data portions, it can also be used to transmit large quantities of data having the same benefits over Glossy as described above. This is because Glossy always receives the full packet before transmitting it, while Whisper only receives a packlet once. At first view, Whisper's packlet size seems limited since all N_{tx} packlets have to fit within 127 byte – the maximum packet size in IEEE 802.15.4 as discussed in Section 2.3.1. However, we show in the following Section 4.2.2 how to enable Whisper to send signaling packets of more than 127 byte.

4.2.2 *Sending identical packlets*

Sending identical packets is a necessary condition for constructive interference to occur, or more precisely, for packets not to interfere destructively, when synchronous transmissions are used. In Whisper, this translates in ensuring that all packlets sent at the same time are identical.

The only value that changes across different packlets within a signaling packet is the *packlet counter* p, which is the 1-byte payload of each packlet. As illustrated in Figure 4.1, the first packlet has counter $p = 0$, the second $p = 1$, and so on. When nodes start sending their own signaling packet, they must properly set the value of the counter p in their packlets. In particular, as also shown in Figure 4.3, Whisper makes a node that receives a packlet with counter $p = i$ set the counter of its first packlet to $p = i + 2$. This is because while the $p = i + 1$th packlet is being transmitted, the radio switches from receive to transmit mode,

5 The reference implementation of Glossy we use in the evaluation has a guard time of roughly 130 µs (measured experimentally). In [73], *Istomin et al.* showed that a guard time of 150 µs is sufficient to compensate for clock drifts that accumulate over 5 minutes.

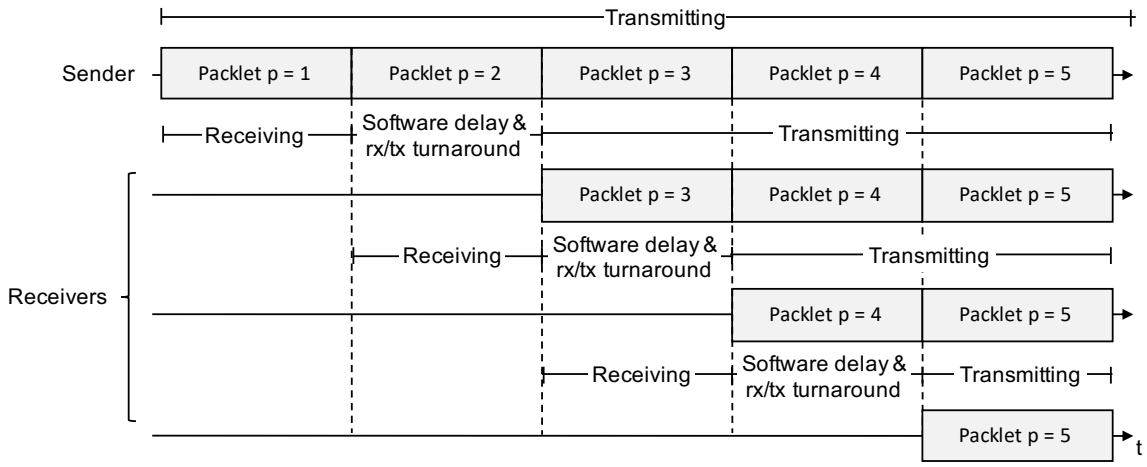


Figure 4.3: Operation of Whisper. Nodes receive a packlet, process it while turning their radio to transmit mode and afterwards transmit their signaling packet.

called rx/tx turnaround. In this time frame, the node, thus, "misses" a packlet and has to wait with its transmission for the next packlet being transmitted.

Besides ensuring that the value of the packlet counter p is identical for all concurrently transmitted packlets, Whisper must also properly set the *length* field of the packlets. This field specifies the length in bytes of the payload and the footer. For packlets that have a 1-byte payload, the length field must thus be set to 3 byte. This can be easily done for all packlets but the first. As illustrated in Figure 4.1, the packlet with counter $p = 0$ "borrows" the preamble, SFD, and length field of the signaling packet. The first byte of the signaling packet, however, must specify how many bytes the radio must send before automatically ceasing to transmit. If Whisper would use this mode of operation (called *buffered mode* in the CC2420 [72]), the (first) length field in the signaling packet would indicate the total length of the signaling packet in bytes – which is different than 3 byte. This would cause the length field of each signaling packet to collide with the length field of concurrently sent packlets. As a result, destructive interference would occur and cause packet drops.

To avoid this problem, we exploit an alternative transmit mode available on certain IEEE 802.15.4 radio transceivers (including the CC2420 [72], the radio chip of our reference platform, the TelosB): the *TXFIFO looping mode*. In this mode, the radio ignores the length field and just continuously reads data from the radio buffer and sends it. Once the content of the buffer has been sent, the radio wraps around and starts to read and send the data from the beginning. This continues indefinitely until a timeout explicitly stops the transmission. Since the value of the length field is ignored when the radio operates in TXFIFO looping mode, Whisper can set the first length field to the length of a packlet

instead of to the length of the signaling packet, thus completely overcoming the problem described above. Further, the length of the signaling packet that is transmitted is not limited to 127 byte. While the radio transmits the data in the buffer at a rate of 250 kbit/s (using MSK at 2.4 GHz as discussed in Section 2.3.2), the MCU can write new data into the buffer. The data is written via Serial Peripheral Interface (SPI) at a rate equal to half the MCU frequency, thus 2.097 MHz. As a consequence, the size of the transmitted data packet is unlimited. However, the receivers still operate in buffered mode, thus, practically the length of a packlet is limited to 127 byte.

While this mode of operation may not be available on all IEEE 802.15.4 transceivers, which limits the portability of Whisper, we believe that it is important to explore novel design ideas notwithstanding current technological limits and protocol standards. We further plan to explore an alternative approach to avoid the TXFIFO looping mode: using byte-wise transmission power control as in [140] to send the length field using the smallest possible transmit power. In our performance evaluation of Whisper in Section 4.3 we nonetheless explicitly consider a fully IEEE 802.15.4-compliant version of the protocol (called Whisper (compliant)), which we describe in Section 4.2.7.

4.2.3 *Sending packlets synchronously*

In the previous subsection, we mentioned that after receiving a packlet (with counter p_i), a node must wait for an entire $T_{packlet}$ before sending its first packlet (with counter $p_i + 2$). This is because while packlet $p_i + 1$ is on the air, the node must perform the rx/tx turnaround of the radio, which lasts $T_{turn} = 192$ µs for IEEE 802.15.4 radios [67]. This leaves a wait time $T_{wait} = T_{packlet} - T_{turn} - T_d$, where T_d is the time that elapses between the rising SFD edge of a sender during transmission and the corresponding rising SFD edge of a receiver during reception.

The existence of T_d is due to the fact that the reception of a packlet lasts slightly longer than its transmission. This *data delay* is a common phenomenon in wireless radios, and each transceiver has a specific latency of the receive and transmit paths, which is reported in the data sheets. If not compensated for, the existence of T_d would make nodes start sending the next packlet before receivers have completed the reception of the previous one. Including propagation delay, T_d is reported to be $3 < T_d \leqslant 3.6$ µs [72, 84, 188] and is thus non-negligible. In Whisper, we set $T_d = 3$ µs.

The existence of this fixed wait time is a further difference between Whisper and Glossy. Indeed, Glossy aims at re-sending a packet as quickly as possible after receiving it (i. e., immediately after the rx/tx turnaround). This is because the longer nodes wait to re-transmit a packet, the stronger MCU clock instabilities become relevant and can thus cause transmissions of different nodes to misalign, as demonstrated in the previous chapter. The software delay T_{sw} in Glossy is roughly 23.13 µs. In Whisper, if we assume a payload of

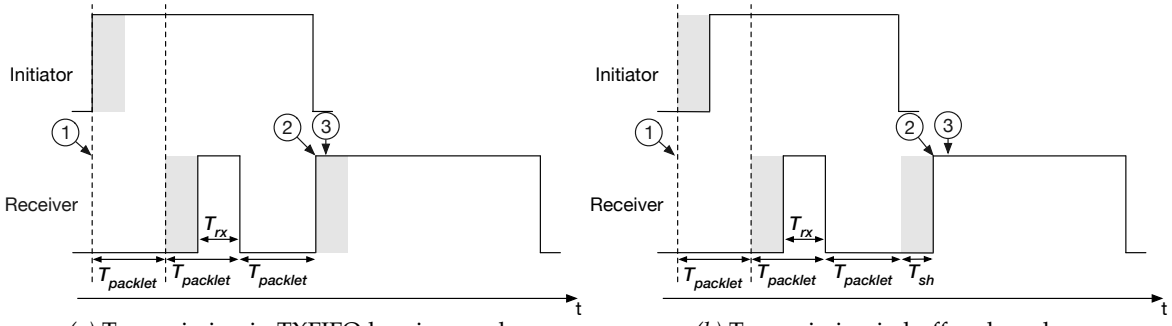
1 byte and a preamble of 2 byte – which corresponds to the shortest possible packlet – then $T_{packlet} = 224$ µs and, thus, $T_{wait} = 29$ µs. Although the values of T_{wait} and of Glossy's software delay are relatively close to each other, the latter does not depend on the length of the packet whereas it does in the case of Whisper. The wait time is thus more critical for Whisper than for Glossy.

To cope with this issue, we employ Flock, the clock drift compensation mechanism we presented in Chapter 3. Flock compensates for instabilities in the DCO that drives the MCU of many low-power hardware platforms. It, thus, makes Whisper able to ensure that packlet transmissions align within the 0.5 µs window notwithstanding the existence of T_{wait} and even if T_{wait} is significantly longer than 29 µs.

4.2.4 *Time synchronization*

Whisper provides implicit time synchronization, similar to Glossy (see Section 2.4.3). For this, a network running Whisper requires a fixed initiator that (infrequently) disseminates signaling packets to which the nodes synchronize. Because packlets are sent back-to-back without gaps in between, the nodes can simply compute the time at which the initiator has started transmitting the signaling packet by knowing the packlet counter p and the length of a packlet. The time the initiator has started the transmission of the signaling packet is then used by the nodes as a common *reference time*.

Just as Glossy, Whisper exploits the Virtual High Resolution Time (VHT) approach introduced by *Schmid et al.* [146] to compute the reference time. The rationale behind VHT is to use two independent time sources for high-resolution, low-power time stamping of events: a high resolution clock that, however, is often prone to frequency deviations and a stable low resolution clock. Since the length of a packlet is fixed and known a priori, the nodes only need to know the packlet counter p and either the arrival time of the packlet or the time the nodes start transmitting their signaling packet to compute the reference time. Both the arrival time of the packlet and transmit time of the signaling packet are time stamped in Whisper using the DCO that runs with $f_{DCO} = 4\,194\,304$ Hz. Figure 4.4 illustrates the procedure of Whisper taking the relevant time stamps that are required for computing the reference time. As shown in the figure, we use in our implementation the transmit time of the signaling packet, indicated with ②, for computing the reference time, shown as ①. This is because the time constraints are more relaxed during this phase of Whisper. Thus, we can safely capture time stamps from the high resolution and the low-resolution clock – which are required for VHT. Capturing the two timestamps for VHT are indicated with ③ in Figure 4.4. As shown in Figure 4.4b and discussed in detail in Section 3.1, the SFD pin goes active after the successful transmission of the packet's SFD field in buffered mode, i. e., the default operation mode of the radio. Thus, the nodes must include also the duration of transmitting the synchronization header T_{sh} in their computation of the reference time.



(a) Transmission in TXFIFO looping mode *(b)* Transmission in buffered mode

Figure 4.4: SFD activity when transmitting in TXFIFO looping and buffered mode. In TXFIFO looping mode, the SFD pin rises at the beginning of the preamble transmission, while in buffered mode the SFD pin rises after the successful transmission of the SFD field in the packet's synchronization header (depicted in gray). Thus, during buffered mode, the nodes also have to include the transmission duration of their synchronization header T_{sh} when computing the *reference time* ①. The reference time is computed using the packlet counter p, the time to transmit a packlet $T_{packlet}$, the time instant at which the DCO time stamps the transmission of the signaling packet ②, and one time capture with both the high-resolution timer (i. e., the DCO) and the low-resolution timer (i. e., the external 32 kHz crystal) ③.
In receive mode, Whisper uses the buffered mode only, thus, the SFD pin rises after having successfully received the SFD field, as already discussed in detail in Section 3.1.

However, we found that during the TXFIFO looping mode, the SFD pin is already set at the beginning of the preamble transmission. This has to be considered when computing the reference time, as shown below:

$$
\begin{aligned}
T^{l}_{tx\_to\_cap} &= ③^{l} - ②^{l} \\
T^{h}_{tx\_to\_cap} &= \frac{1 + T^{l}_{tx\_to\_cap}}{\frac{f_{DCO}}{f_{32\,kHz}}} \\
T^{h}_{ref\_to\_tx} &= \begin{cases} \left((p+2)\cdot T_{packlet}\right)\cdot f_{32\,kHz} & \text{if: TXFIFO looping mode} \\ \left((p+2) * T_{packlet} + T_{sh}\right)\cdot f_{32\,kHz} & \text{if: buffered mode} \end{cases} \\
①^{h} &= ③^{h} - \left(T^{h}_{tx\_to\_cap} + T^{h}_{ref\_to\_tx}\right). \qquad (4.1)
\end{aligned}
$$

We indicate in Equation 4.1 with raised "l" and "h" time values with low and high resolution, respectively. Note that the translation from $T^{l}_{tx\_to\_cap}$ to $T^{h}_{tx\_to\_cap}$ in the second line is taken from the Glossy source code[6].

6 https://sourceforge.net/p/contikiprojects/code/HEAD/tree/ethz.ch/glossy/

4.2.5 *Lazy sampling*

A straightforward way to maximize the probability that a node intercepts a signaling packet consists in making the nodes keep their radio in idle listening for the entire duration of a *Whisper slot* T_{slot}. This is the time interval during which a Whisper flood is executed and during which – like in Glossy and other protocols based on synchronous transmissions – all other application tasks executing the hosting platform are suspended. The length of the slot is a protocol parameter and should be set depending on the expected network diameter d_{net}. In particular, it holds:

$$T_{slot} = (2d_{net} + N_{tx}) \cdot T_{packlet}\,, \tag{4.2}$$

where $T_{packlet}$ is the time needed to send a packlet. The first addend in Equation 4.2 accounts for the fact that Whisper progresses at a "speed" of 2 packlets per hop, as illustrated in Figure 4.3. The second addend considers that at the last hop, after the first packlet has been transmitted, a node must still transmit $N_{tx} - 1$ packlets. If Whisper is used in a network of 6 hops and with $N_{tx} = 3$ and $T_{packlet} = 224\mu s$ (1 byte payload), a slot length of 3.36 ms would be sufficient. In practical settings, however, it is recommendable to use a slightly larger value to account for synchronization drifts and other issues. In the experiments presented in Section 4.3, for instance, we use $T_{slot} = 5$ ms.

The advantage of the lazy sampling strategy is that it is independent of the network topology and communication scheme. However, a significant drawback is that it causes all nodes in the network to stay in idle listening for an entire Whisper slot, even if no signaling packet is sent. To reduce this idle listening time, and thus, the overall radio-on time, Whisper exploits a different strategy, which we call *direction-aware sampling*. We detail this direction-aware sampling strategy in the following section.

4.2.6 *Direction-aware sampling*

The main idea behind the direction-aware strategy is to let the nodes switch their radio on only shortly before the flood is expected to "pass by". In low-power networks, traffic often flows in one direction only, e. g., from an initiator towards all other nodes in the network in data dissemination scenarios [37, 38, 52] or from all nodes to a sink in data collection [73]. If the direction of the traffic is known – hence the name *direction-aware* sampling – Whisper can exploit this information to run an efficient sampling strategy.

In a dissemination scenario like the one considered in Glossy, for instance, traffic always flows from a fixed initiator to all other nodes. If Whisper is used in this scenario, the counter p of the packlet received by a forwarding node depends on the distance in hops between the node and the initiator. If the topology of the network can be assumed to be

static or vary slowly, this distance, and thus, the counter p, can also be assumed to be constant or to vary only a little across consecutive floods. Whisper exploits this situation and lets each node keep in memory two values: p_{min} and p_{max}. Both values are estimates of the counters of the "earliest" and "latest" packlet that a node expects to receive and are used by the nodes to compute when to turn the radio on and off, respectively. In particular, the nodes compute the start of the sampling interval with:

$$t_{start} = t^*_{start} - T_{guard} + \max(0, p_{min} - 1) \cdot T_{packlet} \tag{4.3}$$

In Equation 4.3, t^*_{start} indicates the time at which the initiator is expected to start its transmission, whereas T_{guard} is the guard time that protects against possible synchronization drifts. The term $\max(0, p_{min} - 1)$ indicates that the nodes wake up to the packlet before the one they actually expect. This allows the nodes to learn the arrival of earlier packlets when channel conditions change. The duration of the sampling interval can be calculated with

$$T_{sampling} = (\lfloor p_{max} \rfloor + (N_{tx} + 1)) \cdot T_{packlet} - t_{start}. \tag{4.4}$$

In the following, we detail how Whisper determines the values for p_{min} and p_{max}.

Computing p_{min} *and* p_{max}

When a node has not yet received its first packlet, it sets $p_{min} = 0$ and $T_{sampling} = T_{slot}$. After the first packlet with counter p is received, the nodes set $p_{min} = p_{max} = p$. Afterwards, the mechanisms described in the following are used to update p_{min} and p_{max} after each Whisper slot. The value of p_{min} is set to the lowest value of p ever received. Thus, the nodes wake up as early as possible to avoid missing a packlet. The computation of p_{max} is slightly more elaborate. The value p_{max}, on one hand, must be set such that the nodes have their radio on for as long as necessary to be able to capture packlets, even when the earlier hops are exposed to interference and thus, miss packlets. Underestimating p_{max} would, thus, cause a node to switch off its radio too early, which in the worst case could stop the propagation of the flood. On the other hand, the value p_{max} must be set such that the nodes have their radio on for as short as possible to save energy. Both requirements are fulfilled by computing p_{max} as follows:

$$p_{max} = (p_{max} + p)/2 \qquad \text{if: } p \geqslant p_{max} - 2. \tag{4.5}$$

The term $p_{max} - 2$ in the condition of Equation 4.5 accounts for the progression speed of 2 in Whisper. Thus, Equation 4.5 allows the filtering of outliers that could result in a high

sampling duration, and thus, high energy consumption, while also being able to react to changing channel conditions.

The strategies chosen to set both p_{min} and p_{max} are very conservative and can definitely be improved in future work. The design of Whisper actually opens up opportunities for designing further smart sampling strategies beyond the two – lazy and direction-aware – discussed in this thesis. Further, the discussion above assumes that Whisper is used in a data dissemination scenario with a fixed initiator. In the following, we explain how the direction-aware sampling strategy can be applied to collection scenarios.

Direction-aware sampling in collection scenarios

Whisper can also be used in data collection scenarios. Our direction-aware sampling strategy described above exploits the fact that dissemination results in network tree with the initiator as root. Each node learns its distance from the initiator in terms of hops based on p. Running Whisper in collection scenarios only requires the nodes to start listening for incoming packlets in the tree's reverse order, i. e., from the leaves of the tree to the root, with the sink as root node. To learn their position in the tree, a short initialization phase is required, where the sink disseminates signaling packets. Figure 4.5 illustrates the initialization phase on the left side and the actual collection on the right side with $N_{tx} = 3$. During the initialization phase, the nodes calculate their position in the reverse tree. For calculating their position in the reverse tree, the nodes need to know the last packlet being disseminated during dissemination in a Whisper slot, which can be computed – by knowing the network diameter d_{net} – with:

$$p_{slot}^{dissemination} = 2d_{net} + N_{tx} - 1. \tag{4.6}$$

As illustrated in Figure 4.5, $p_{slot}^{dissemination} = 14$ in a network with 6 hops and $N_{tx} = 3$. During dissemination, and thus, in the initialization phase, the initiator immediately starts transmitting with packlet $p = 0$ after waking up, as shown on the left side of Figure 4.5. However, during collection, any node with pending data can be sender and start transmitting. Thus, during collection, a node is either sender or forwarding node. Letting a sender transmit at $p = 0$, as the sink during dissemination, implies that it would expect a packlet at $p = -2$ when it is a forwarder (indicated in Figure 4.5 with "potential RX"). We therefore have to add an offset of 2 when calculating the last packlet being disseminated. Thus, as shown on the right side of Figure 4.5, the sender (during the initialization phase it was the 6th hop) transmits at $p = 2$ and potentially receives a packet at $p = 0$. The resulting equation is

$$p_{slot}^{collection} = p_{slot}^{dissemination} + 2. \tag{4.7}$$

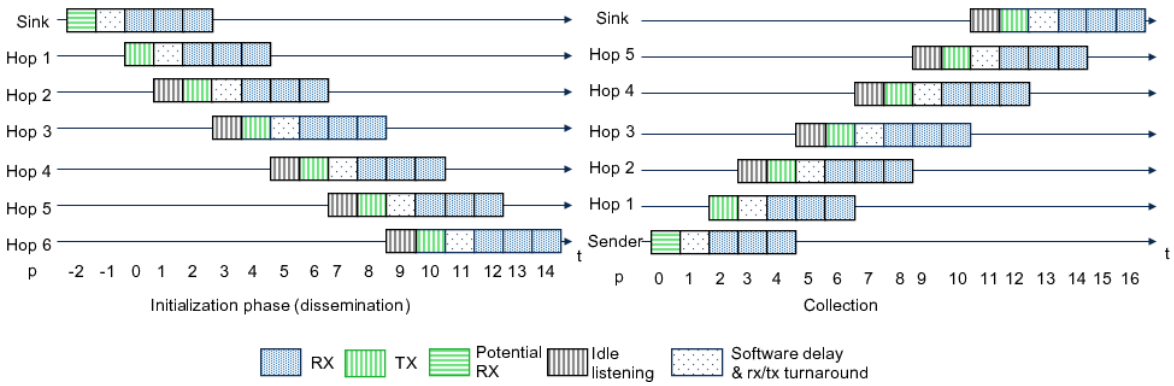


Figure 4.5: Packlets transmitted and received per hop during initialization phase and collection with $N_{tx} = 3$. On the left side, the sink disseminates a packlet through the network in the initialization phase. On the right side, one node in the 6th hop called sender, triggers during collection a packet propagation to the sink. The sender would expect to receive a message at $p = 0$. So it starts transmitting at $p = 2$.

Knowing $p_{slot}^{collection}$ the nodes can now calculate the reverse packlet counter $p_{reverse}$ during the initialization phase based on the incoming packlet p with

$$p_{reverse} = p_{slot}^{collection} - (p + N_{tx} + 3). \tag{4.8}$$

The sink node assigns itself $p^{reverse} = p_{slot}^{collection} - (N_{tx} + 1)$.

After the initialization phase, the nodes calculate p_{min} and p_{max} in the same way as for dissemination. The same is true for calculating the duration of the sampling interval $T_{sampling}$ and the start of the sampling interval t_{start}. A sender, however, has to start the transmission with $p_{min} + 2$. That is because p_{min} indicates the packlet the sender expects to receive, but it has to transmit the packlet the successive hop expects. The senders can compute when to start transmitting with

$$t_{start}^{sender} = t_{start}^{*} + ((p_{min} + 2) \cdot T_{packlet}). \tag{4.9}$$

4.2.7 *Whisper (compliant)*

The standard version of Whisper described above exploits low-level mechanisms of the radio transceiver. For the sake of completeness, we consider in our evaluation also an IEEE-802.15.4-compliant version of Whisper, called *Whisper (compliant)*. This version uses a 4-bytes preamble and the radio in buffered mode, which has the following two consequences. First, the first length field of the signaling packet must be set to the actual length of the

payload and will thus collide with the length field of concurrently sent packlets. This causes the first packlet of each signaling packet to be dropped and thus slows down the progression of the flood. In particular, Whisper (compliant) needs three instead of two packlets per hop to progress. Second, the radio hardware will set the footer of the signaling packet, i. e., the gray-shadowed footer in Figure 4.1. To avoid this footer to collide with the footer of synchronously sent packlets, Whisper (compliant) makes all nodes stop sending at the same time to align the footers of all signaling packets.

4.2.8 *Resilience against external interferences*

As other approaches based on synchronous transmissions, Whisper is sensitive to external interference. Common devices such as microwave ovens or Wi-Fi access points can disturb communication and significantly reduce the reliability of the protocol. Several approaches already presented in the literature show that introducing frequency diversity, and in particular channel hopping, is an effective countermeasure against external interference [74, 130, 153]. These techniques, especially sending each flood on a different frequency [74], are straightforward to integrate in Whisper.

4.2.9 *The portability of Whisper*

Whisper exploits low-level mechanisms that are, admittedly, not available for all radio transceivers. However, the design of Whisper is still not limited to the CC2420 radio chip and the TelosB platform. Many platforms are still equipped with this radio chip. For instance, the XM1000 platform [1] uses the CC2420. Other radio chips also provide the option to change the preamble length and provide the TXFIFO looping mode (or a similar option). These include, for example, the CC2520 [69], which is used by the WiSMote [6] platform and the CC1101 [71] that is equipped on the MSP430-CCRF [103]. In particular, the CC1101 provides three packet length modes that can be reprogrammed during receive and transmit: fixed mode, variable mode, and infinite mode. Using these modes, the CC1101 supports packet lengths that are longer than 127 byte. Whisper can use them to avoid the transmission of the length field of the signaling packet.

For radios that do not support the TXFIFO looping mode or similar options, we still provide an IEEE 802.15.4-compliant variant of Whisper, as described in Section 4.2.7. In addition, as mentioned in Section 4.2.2, the byte-wise transmission power control approach from [141], where the length field is transmitted with the smallest transmit power, is a promising solution to mark the signaling packet length.

4.3 EVALUATION

In this section[7], we evaluate Whisper in extensive testbed experiments. We start by presenting our evaluation setup in Section 4.3.1. We then compare Whisper and Glossy in Section 4.3.2 and find that *nodes using Whisper achieve a significantly smaller radio-on time, both with and without data traffic, compared to Glossy*. Next we present in Section 4.3.3 the evaluation results for the case that multiple senders disseminate a signaling packet concurrently. In this scenario we find that *contention causes less packet drops and thus, a higher reliability in Whisper compared to Glossy*. Afterwards, we evaluate in Section 4.3.4 the impact of Whisper's low-level mechanisms: preamble length, number of transmissions N_{tx}, and collisions due to different length fields. We find that the *preamble length has a neglectable impact on the reliability compared to the savings in the radio-on time*. Further, *the number of retransmissions has a stronger impact on Glossy than on Whisper*, and, *the colliding packet length fields have strong impact on the progression speed of a Whisper-flood*. Lastly, we use Whisper within Crystal – a recently presented collection protocol – in Section 4.3.5 and show that *Whisper significantly reduces Crystal's duty cycle while also impacting reliability*.

4.3.1 *Evaluation setup*

In this section, we present our evaluation setup, including a short description of our implementation, the used Whisper and Glossy versions, and the metrics and scenarios that we consider during this evaluation.

We implemented Whisper for the Contiki operating system[8]. We embedded the code base of Flock[9] into Whisper and reused parts of the publicly available implementation of Glossy[10] in our code.

To illustrate the performance of Whisper in detail, we implemented different versions of the protocol. Whisper is the full-fledged protocol that includes direction-aware sampling (see Section 4.2.6) and exploits the TXFIFO looping mode (see Section 4.2.2). In Whisper, we further use a 2-byte preamble as mentioned in Section 4.2.1 and set $N_{tx} = 3$.

We also explore the performance of Whisper in a series of other configurations, e. g., with lazy sampling instead of direction-aware sampling, with a 4-byte instead of 2-byte preamble, as well as with different values for N_{tx}. In the plots, we indicate after the name of Whisper the specific change with respect to the default implementation, i. e., *Whisper (lazy)* indicates a version of Whisper that uses lazy sampling but keeps the TXFIFO looping mode,

7 This section and the following subsections are an extended version of Section IV of our previous work published in [22]. More precisely, we extended this thesis with the following subsection: Section 4.3.5 – Crystal and Whisper.

8 `http://www.contiki-os.org/`

9 `http://github.com/martinabr/flock`

10 `http://sourceforge.net/p/contikiprojects/code/HEAD/tree/ethz.ch/glossy/`

the 2-byte preamble and $N_{tx} = 3$. Lastly, we also consider the fully IEEE 802.15.4-compliant version of Whisper described in Section 4.2.7. Whisper (compliant) uses a 4-byte preamble, lazy sampling, $N_{tx} = 14$ and does not exploit the TXFIFO looping mode.

As for Glossy, we use its publicly available code base[10]. As discussed in Section 4.2.1, we set the payload of Glossy packets to 1 byte to avoid an unfair penalization of Glossy due to its larger packet size. We provide experimental results obtained by running Glossy with both a 2-byte and a 4-byte preamble.

We run experiments in different dissemination scenarios, as summarized in Table 4.1. For dissemination, we test both with only one sender and with different senders. We also consider the case in which different senders transmit concurrently and differentiate between concurrent senders positioned close-by each other or roughly evenly distributed across the network.

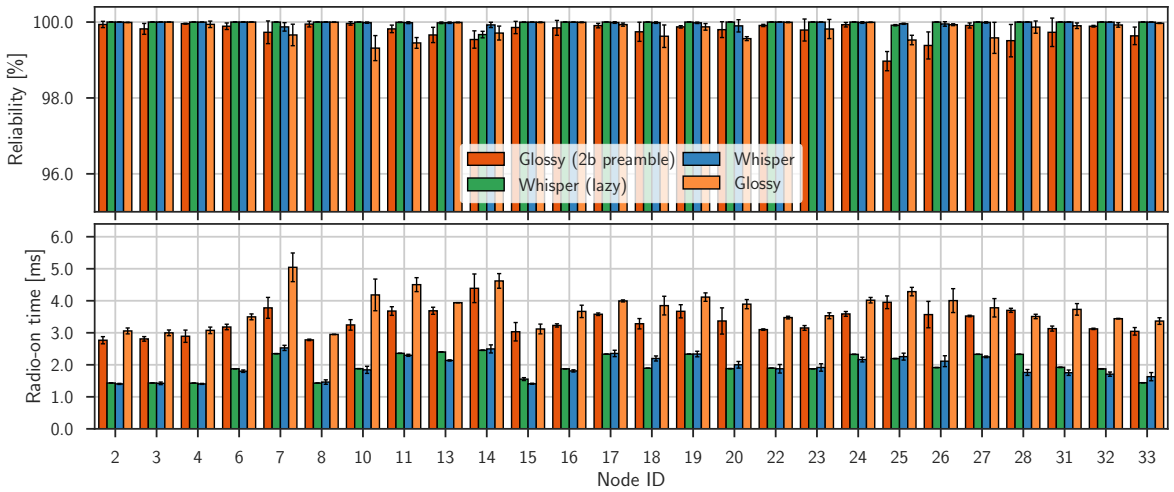
We run our experiments in the FlockLab testbed (see Section 2.1.3) and thereby focus on two key performance metrics: reliability and radio-on time. We compute the *per-node reliability* as the ratio of the total number of signaling packets successfully received by a node and the total number of signaling packets sent during an experiment. We then derive the *network reliability* as the average of the reliability of all nodes in the network. The *radio on-time* is the time the radio is turned on and active (including idle listening) during a Whisper (or Glossy) slot. As for the case of reliability, the radio-on time of the network is computed as the average of the radio-on time of each node.

4.3.2 *Whisper vs. Glossy*

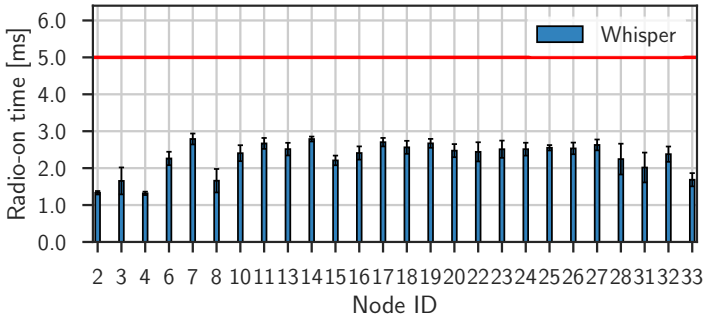
We first compare the performance of Whisper, Whisper (lazy) as well as Glossy and Glossy (2b preamble) in a dissemination scenario with a single, fixed sender (diss. fixed). This scenario corresponds to, e. g., a controller that needs to signal the nodes to stay awake for an unscheduled software update. We find that nodes using Whisper achieve, compared to Glossy, a similar or higher reliability and a significantly smaller radio-on time, both with and without data traffic.

Table 4.1: Summary of evaluation scenarios and configuration parameters.

Scenario	*Label*	*Sender [node id]*
Dissemination with fixed sender	diss. fixed	1
Dissemination with different senders	diss. diff.	10, 22, 11, 16, 23, 19, 20, 31, 26, 7
Dissemination with concurrent, close-by senders	diss. close	4, 2, 8, 1
Dissemination with concurrent, far-away senders	diss. far	16, 19, 7, 1



(a) Performance during data dissemination. In comparison with Glossy, both Whisper and Whisper (lazy) reduce the radio-on time by a factor of two while achieving a reliability near 100 %. Nodes have learned their distance to the source (node 1) and efficiently turn their radio on before the flood "passes-by".



(b) Whisper achieves a low radio-on time even when no signaling packet is disseminated. In contrast, Whisper (lazy) and Glossy use a fixed timeout mechanism that is set to 5 ms (marked as red line) to turn the radio off in case no packlet has been received. Learning the distance to the source (node 1) allows for energy-efficient channel checks.

Figure 4.6: Performance of Whisper, Whisper (lazy), Glossy, and Glossy (2b preamble) at 0 dBm in FlockLab in a data dissemination scenario with only a single, fixed sender (diss. fixed). Whisper outperforms Glossy in terms of energy-efficiency during data dissemination as well as when no signaling packet are sent.

EXPERIMENTS We run Whisper, Whisper (lazy), "standard" Glossy and Glossy (2b preamble) in the following configuration. We select the node with identifier 1 as the sender. As shown in Figure 2.2, the node is located on the outer edge of the FlockLab testbed, which allows us to obtain a large network diameter. To vary the topology and in particular the number of hops between the sender and the farthest receivers, we use two different transmit powers: −10 dBm and 0 dBm. This results in a network diameter of 3 to 4 and 5 to 6 hops, respectively. Each experiment consists of 10,000 floods, and we repeat

each experiment 3 times. For Whisper, we further measure the radio-on time over 5,000 slots during which the sender sends no signaling packet. The collected per-node data is averaged over the three independent runs and the standard deviation is plotted as error bars in the figures.

RESULTS Figure 4.6a details – for the case in which the sender disseminates a signaling packet in each slot – the per-node reliability in the upper plot and the per-node radio-on time in the lower plot. While the reliability of Whisper and Whisper (lazy) is comparable or slightly higher than that of Glossy and Glossy (2b preamble), the radio-on time is significantly lower – roughly half that of Glossy in most cases – for Whisper and Whisper (lazy). Figure 4.6a also shows that Whisper and Whisper (lazy) achieve a similar radio-on time. This is due to the small network diameter at 0 dBm in FlockLab. Table 4.2 shows a higher difference in the radio-on time between the two protocols at −10 dBm.

Figure 4.6b shows the per-node radio-on time of Whisper when no signaling packet is sent. The bold (red) line at 5 ms corresponds to the radio-on time of approaches like Whisper (lazy) or Glossy that – in case of the absence of communication – keep nodes in idle listening for the entire slot. Whisper can save radio-on time in this case thanks to the use of direction-aware sampling, which makes nodes switch their radio off at the latest when the expected reception time of the packlet with counter $p = p_{max} + N_{tx} + 1$ has elapsed. This characteristic of Whisper is particularly relevant when nodes must frequently switch on their radios to limit delays in relaying data traffic – yet often no packet is flooded, like in the data prediction scenario of Crystal [73].

4.3.3 *Concurrent dissemination of signaling packets*

To consider the case in which different nodes must signal the presence of data – possibly even concurrently – we evaluate the performance of Whisper, Whisper (lazy) and Glossy in the three scenarios *diss. diff.*, *diss. close*, and *diss. far* (see Table 4.1). We find that contention for the same slot causes less packet collisions – and thus results in higher reliability of – in Whisper and Whisper (lazy) compared to Glossy.

EXPERIMENTS We run Whisper, Whisper (lazy) and Glossy consecutively with transmission power −10 dBm and 0 dBm. In the *diss. diff.* scenario, each sender consecutively transmits 1,000 signaling packets before handing over to the next sender. We execute 10,000 floods in each experiment (i. e., for each protocol), and we run each experiment 3 times.

Table 4.2: Summary of evaluation results. Whisper and Whisper (lazy) outperform Glossy in terms of reliability and radio-on time in various scenarios. Nodes using Whisper (lazy) and Glossy use a timeout mechanism to turn the radio off in case they have not intercepted a packlet/packet within a given time. In this evaluation the timeout is set to 5 ms.

Protocol	*Scenario*	*Tx power* [dBm]	*Reliability* [%]	*Radio-on /w signaling* [ms]	*Radio-on w/o signaling* [ms]
Whisper	diss. fixed	-10	99.980	2.055	2.546
		0	99.980	1.936	2.474
Whisper (lazy)	diss. fixed	-10	99.817	2.477	5.0
		0	99.983	1.962	5.0
	diss. diff.	-10	99.932	2.175	5.0
		0	99.986	1.865	5.0
	diss. close	-10	99.887	2.438	5.0
		0	99.952	2.129	5.0
	diss. far	-10	99.786	1.626	5.0
		0	99.965	1.540	5.0
Glossy	diss. fixed	-10	99.738	4.253	5.0
		0	99.828	3.756	5.0
Glossy (2b preamble)	diss. fixed	-10	99.616	3.914	5.0
		0	99.767	3.356	5.0
	diss. diff.	-10	98.369	3.805	5.0
		0	98.963	3.351	5.0
	diss. close	-10	99.350	4.071	5.0
		0	99.024	3.932	5.0
	diss. far	-10	98.881	3.680	5.0
		0	98.559	3.721	5.0

RESULTS Figure 4.7 shows the network reliability (upper plot, left), radio-on time (lower plot, right) and the percentage of dropped packlets/packets per Whisper/Glossy slot (lower plot, left). In all the considered scenarios, Whisper and Whisper (lazy) achieve a higher reliability and a lower radio-on time than Glossy.

The difference in performance is more evident in scenarios with concurrent senders, i. e., *diss. close* and *diss. far*. The reason is that interference due to concurrent floods has a stronger impact in Glossy than in Whisper. More precisely, floods from different senders overlap with a slightly different temporal displacement caused by (i) senders not being synchronized within sub-microseconds and (ii) as stated in [112] *"a combination of software, hardware, and signal propagation delays"* caused by an increasing number of concurrent transmitters. While (i) affects both Whisper and Glossy to an equal extent, (ii) intensifies for each gap between consecutive transmissions, resulting in a stronger impact on Glossy than on Whisper. The consequence is that nodes using Glossy drop more packets on av-

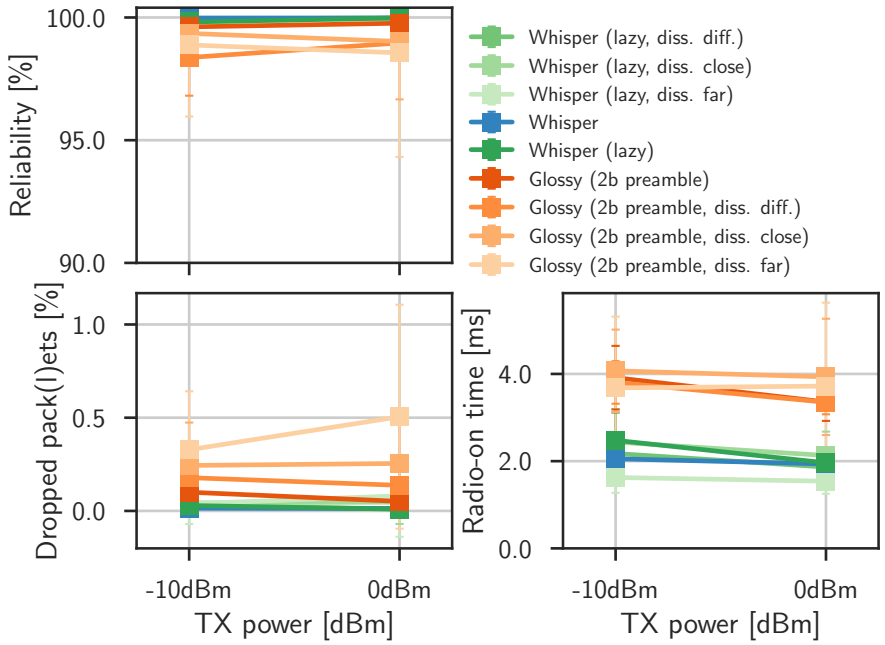


Figure 4.7: Comparison in dissemination scenarios. Whisper and Whisper (lazy) achieve in all scenarios a two-fold lower radio-on time compared to Glossy. At the same time, Whisper and Whisper (lazy) achieve a higher reliability.

erage, e. g., we measure 0.5 % in *diss. far* (lower plot, left) at 0 dBm resulting in 1 % lower reliability compared to Whisper (lazy), in which nodes only dropped on average 0.03 % of the packets.

4.3.4 *Impact of low-level mechanisms*

We now investigate the impact of the individual low-level mechanisms used in Whisper. We thereby consider a dissemination scenario with a single, fixed initiator (*diss. fixed*).

4.3.4.1 *Impact of preamble length*

We start by analyzing the effect of the preamble length on the performance of Whisper (lazy) and Glossy. We find that a 2 byte preamble significantly reduces the radio-on time for both protocols while causing a negligible loss in terms of reliability.

EXPERIMENTS We run Whisper (lazy) and Glossy in the diss. fixed scenario using $N_{tx} = 3$, preamble length of both 2 byte and 4 byte, and transmit powers −10 dBm and

0 dBm. We execute 10,000 Whisper/Glossy network floods for each protocol and collect data from 3 independent runs.

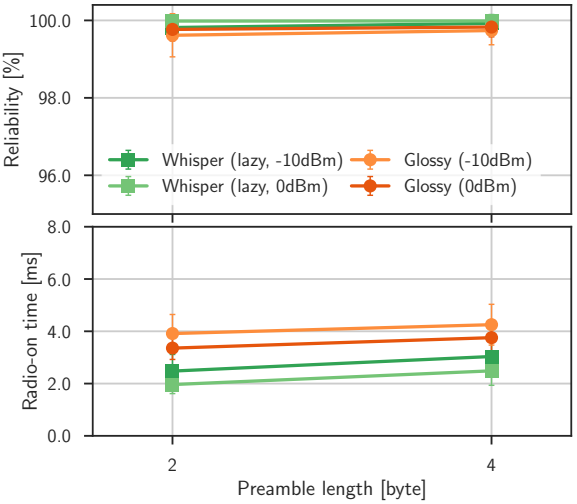
RESULTS Figure 4.8a shows the network reliability in the upper plot and the achieved radio-on time in the lower plot. One can observe a slight increase in reliability with the 4 byte preamble compared to the 2 byte preamble. Comparing the gray-shadowed results corresponding to $N_{tx} = 3$ in Table 4.4a, the network reliability with a 2 byte preamble drops about 0.1 % for all protocols and transmit powers, which corresponds to the loss of 10 packets out of 10,000. The radio-on time, however, increases with the longer preamble by 10 % and 20 % for Whisper (lazy) and Glossy, respectively, as shown in Figure 4.8a in the lower plot. To a very small decrease in reliability (0.1 %) thus corresponds a significant improvement in terms of radio-on time (10 %). This can be explained considering that the preamble and SFD byte are used by receivers to achieve symbol synchronization and to adjust for frequency offsets [72]. The length of the preamble, however, only affects transmissions. The receiver starts intercepting a packet as soon as it has found a single preamble byte followed by the SFD. Transmitting a longer preamble is useful to increase the signal-to-noise ratio, and thus, to help the receiver in detecting the preamble and SFD bytes. An increase of the preamble length from 2 to 4 bytes leads, however, to almost negligible improvements, as illustrated in the figure.

4.3.4.2 *Impact of the number of transmissions* N_{tx}

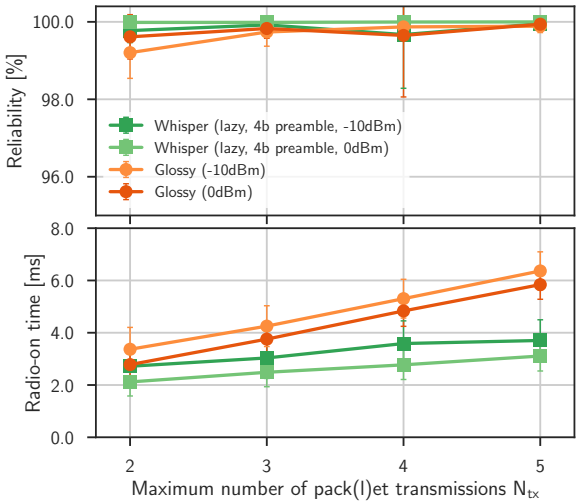
We now discuss how different values of N_{tx} affect the performances of both Whisper and Glossy. We find that both protocols achieve a similar reliability. However, Whisper (lazy, 4b preamble) has a smaller radio-on time than Glossy and with every N_{tx}, the effect on the radio-on time increases stronger in Glossy compared to Whisper (lazy, 4b preamble).

EXPERIMENTS We run Whisper (lazy, 4b preamble) and set $N_{tx} = \{2, 3, 4, 5\}$. We use transmit powers −10 dBm and 0 dBm and execute 10'000 Whisper/Glossy network floods for each protocol and collect data from 3 independent runs. We further run "standard" Glossy with a preamble length of 4 byte in the same configuration.

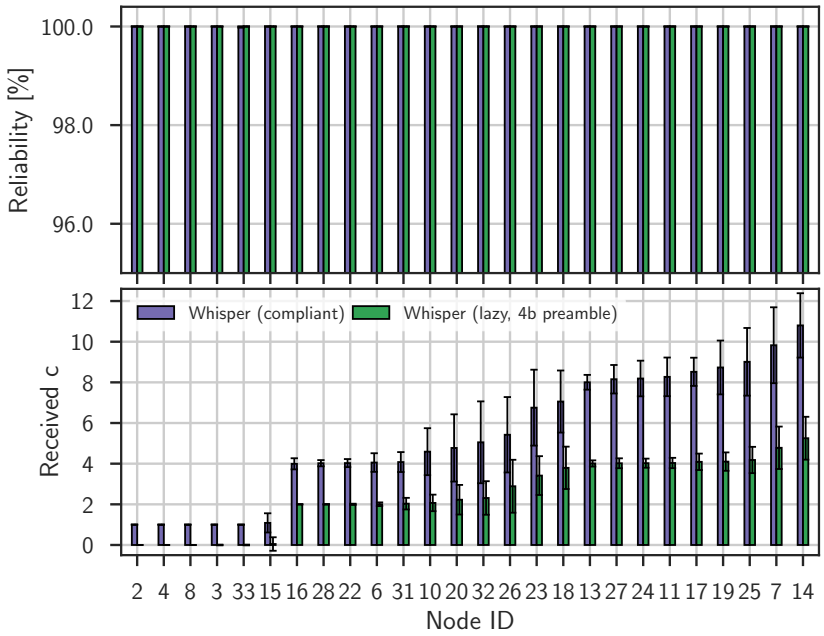
RESULTS Figure 4.8b shows the network reliability in the upper plot and the radio-on time in the lower plot for varying values of N_{tx}. The upper plot of Figure 4.8b shows that for different values of N_{tx}, Whisper (lazy, 4b preamble) and Glossy achieve a comparable reliability. Apart from minor fluctuations, the reliability increases as N_{tx} increases, as expected. The lower plot in Figure 4.8b shows that Whisper outperforms Glossy in terms



(a) Impact of preamble length. A 4 byte preamble increases the overall reliability by 0.1 % while increasing the radio-on time by 20 % and 10 % for Whisper and Glossy, respectively, compared to a 2 byte preamble.



(b) Impact of number of packlet/packet transmissions. Glossy's radio-on time increases stronger with N_{tx} compared to Whisper's (lazy, 4b preamble). The reason is the additional packet reception as well as the RX/TX turnaround.



(c) Impact of length field. Colliding length fields in Whisper (compliant) have a great impact on the progression speed of the flood.

Figure 4.8: Impact of low-level mechanisms.

Table 4.3: Summary results of low-level mechanisms.

(a) Reliability.

Protocol	*Tx power [dBm]*	*4b preamble* $N_{tx}=2$ *[%]*	$=3$ *[%]*	$=4$ *[%]*	$=5$ *[%]*	*2b preamble* $N_{tx}=3$ *[%]*
Whisper (lazy)	-10	99.774	99.921	99.672	99.953	99.817
Whisper (lazy)	0	99.985	99.986	99.996	99.998	99.983
Glossy	-10	99.204	99.738	99.870	99.888	99.616
Glossy	0	99.613	99.828	99.649	99.939	99.767

(b) Radio-on time.

Protocol	*Tx power [dBm]*	*4b preamble* $N_{tx}=2$ *[ms]*	$=3$ *[ms]*	$=4$ *[ms]*	$=5$ *[ms]*	*2b preamble* $N_{tx}=3$ *[ms]*
Whisper (lazy)	-10	2.718	3.036	3.587	3.705	2.477
Whisper (lazy)	0	2.118	2.487	2.772	3.109	1.962
Glossy	-10	3.367	4.253	5.303	6.363	3.914
Glossy	0	2.781	3.756	4.834	5.841	3.356

of radio-on time even with lazy sampling and a 4 byte preamble. More precisely, Table 4.4b shows that in Whisper (lazy, 4b preamble) increasing N_{tx} by 1 causes an increase of the radio-on time of roughly 288 µs – which corresponds to $T_{packlet}$ for a packlet with a 4 byte preamble and 1 byte payload. In Glossy the radio on-time increases for each N_{tx} by the duration of one received and one transmitted packet á 288 µs, the rx/tx turnaround time with 192 µs and 23.13 µs for the software delay. As a consequence, the increase in radio-on time with increasing N_{tx} is more prominent in Glossy than in Whisper.

4.3.4.3 *Impact of collisions due to different length fields*

Lastly, we compare Whisper (compliant) with Whisper (lazy, 4b preamble, 14 packlets). We find that the collisions due to different length fields in Whisper (compliant) have a significant, negative influence on the speed at which a flood can progress.

EXPERIMENTS We run Whisper (compliant) with 14 packlets, which results in a signaling packet of 122 byte. We further run Whisper (lazy, 4b preamble, 14 packlets) and make all nodes stop transmitting their signaling packets simultaneously. The signaling packet in Whisper (compliant) and in Whisper (lazy, 4b preamble, 14 packlets) differs only for the length field of the first packlet. This is the length of the signaling packet in Whisper (compliant) and the length of a packlet in the latter.

RESULTS Figure 4.8c shows the per-node reliability in the upper plot and the received counter p in the lower plot. We find that each node achieves a reliability of 100 % for both protocols. This is consistent with the results discussed in Section 4.3.4.2, where we found that the reliability increases with each additionally transmitted packlet. This is also the case when nodes simultaneously stop sending instead of ceasing after N_{tx} transmissions.

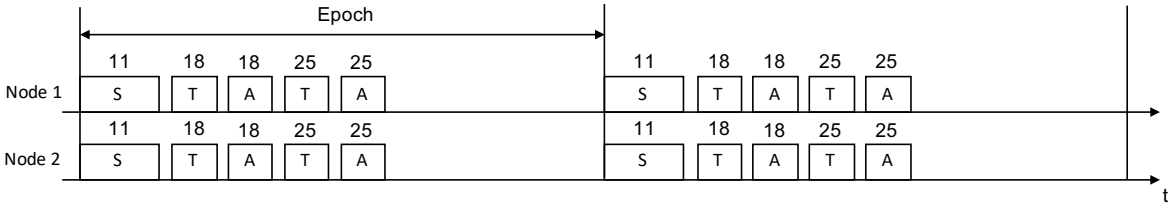
The lower plot of Figure 4.8c reveals that nodes using Whisper (compliant) receive higher counter values compared to Whisper (lazy, 4b preamble, 14 packlets). This is what causes a slower progression of the flood and is not unexpected given that in Whisper (compliant) (i) nodes drop packlets whose length field is not set correctly, and (ii) the packlets are exposed to collisions due to the different length fields. More precisely, the packlet with $p = 0$ is dropped by the nodes in the first hop (nodes with identifiers 2 to 15 in FlockLab), because the length field is not set to the length of a packlet but to the length of the signaling packet. The nodes in the first hop receive packlet $p = 1$ and consequently miss $p = 2$ due to the rx/tx turnaround. They transmit packlet $p = 3$, which collides with the packlet of the sender. Thus, nodes on the second hop (identifiers 16 to 31) successfully receive packlet $p = 4$. This procedure continues until the last hop (node with identifier 14) receives packlet $p = 11$. In comparison, the same node receives $p = 5$ with Whisper (lazy, 4b preamble, 14 packlets). This shows that the flood progresses faster with Whisper (lazy, 4b preamble, 14 packlets) and thus requires less packlets to be sent in total, which reduces the radio-on time.

4.3.5 *Crystal and Whisper*

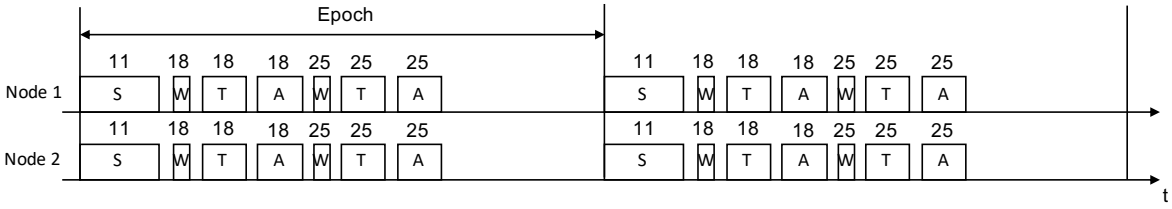
In the following, we integrate Whisper in Crystal [73, 74], a recently proposed data collection protocol based on Glossy that targets data prediction scenarios, and show the performance of Crystal with and without Whisper. We find that Whisper significantly reduces the duty cycle of Crystal, but also impacts its reliability.

CRYSTAL AND CRYSTAL WITH WHISPER In the following, we briefly describe the operation of Crystal and afterwards how Whisper is incorporated in Crystal.

Crystal uses Glossy-floods to let nodes transmit data to a sink node. When the sink node has received a data packet, it acknowledges its reception. Otherwise, it disseminates a negative-acknowledgment. The transmit (T) and acknowledgment (A) slots alternate until the sink has disseminated a negative-acknowledgment twice. After the second negative-acknowledgment, the nodes turn their radio off. Crystal runs periodically and organizes its operations in epochs. Figure 4.9a shows an example of a Crystal epoch. Each epoch starts with a synchronization (S) slot in which the nodes re-synchronize to the sink node.



(a) Crystal.



(b) Crystal /w Whisper.

Figure 4.9: A Crystal epoch with and without Whisper. The (W-), T- and A-slot alternate until the sink node has disseminated a negative-acknowledgement twice during the A-slot. The numbers above the slots denote the channel used for communication. The illustrations are based on Figure 5 in [74].

After the synchronization slot follow the T- and A-slots as described above. To increase resilience against interference, Crystal integrates a channel hopping mechanism. The numbers above the communication slots in Figure 4.9a indicate the channel at which the nodes communicate. In particular, each TA-pair uses the same channel.

The sink node always disseminates packets in the S- and A-slot. However, there is infrequent communication in the T-slot. For example in Crystal's temperature prediction scenario, there is no data dissemination in over 80 % of the T-slots. However, as discussed above, the duration of the T-slot must be sufficiently long to support the packet size that is required by the application. As a consequence, the nodes consume unnecessary energy. To reduce the idle listening time when no data communication is needed, and thus, the energy consumption, Whisper can be integrated into Crystal. Figure 4.9b shows Crystal with Whisper. Whisper (W) runs before each T-slot. A node that has data to share, transmits a signaling packet in the Whisper slot. Nodes that intercept a packlet turn their radio on in the T-slot, otherwise they keep the radio turned off until the following A-slot. In this setting, Whisper relies on the synchronization and channel hopping mechanisms from Crystal. More precisely, Crystal initiates the start of the Whisper slot and also provides the communication channel, which is the same as used for the TA-slots. Since Crystal is a collection protocol, we use Whisper (i. e., Whisper (coll.)) in its collection configuration as described Section 4.2.6. We use Crystal's bootstrapping period – which lasts 10 epochs – as initialization phase for Whisper to let the nodes learn their position in the network tree.

Table 4.5: Crystal's configuration parameters.

Tx power [dBm]	N_S *[#]*	N_T *[#]*	N_A *[#]*	W_S *[ms]*	W_T *[ms]*	W_A *[ms]*
0	3	2	3	10	9	7
-10	4	3	4	14	14	12

EXPERIMENTS In this experiment, we measure the *duty cycle* and the achieved *reliability*. In particular, the duty cycle is the averaged per-node duty cycle that is the ratio of the sum of the radio-on time for all slots during an epoch and the duration of the epoch. The reliability indicates how many nodes that have transmitted an update in the current epoch also received an acknowledgment from the sink. Thus, the measured reliability includes the packet reception rates of the S-slot, T-slot, the A-slot, and when used, also the Whisper (W)-slot.

We first run Crystal with an epoch of 1.5 s and a data payload of 20 byte. We use the default number of retransmissions N for the different slots and transmit powers, given in [73] and summarized in Table 4.5. As also shown in the table, we further used the default duration W for the S- and A-slots. However, since we increase the payload, we adjusted the duration for the T-slot. We run Crystal in FlockLab, with node 1 as sink, and with updates $u = 0$, $u = 2$, and $u = 5$. An update $u = X$ implies that among the 27 nodes, there are X events (or data packets) to share during a specific epoch. Crystal with $u = 0$ indicates that no message is transmitted in T-slot, and thus, in this case we measure the idle listening time of Crystal. More precisely, it indicates the minimal power consumption, and we, thus, use it as base line for other update configurations. After running Crystal 3 times for 60 minutes, we repeat the experiments with Crystal using Whisper (coll.).

RESULTS Figure 4.10 shows the duty cycle in the upper plot and the reliability in the lower plot. We find that Whisper reduces the duty cycle by a factor of two for all updates u. Considering only the energy consumption during communication (i. e., 20 mA) and assuming a battery with a capacity of 2000 mAh, Crystal with Whisper increases the nodes' lifetime by a factor of 2.3 compared to "standard" Crystal (278 days vs. 119 days of lifetime) in this experiment[11]. The factor of how much Whisper increases the network

11 Crystal ($u = 0$) at 0 dBm transmission power has an average duty cycle of 3.46 %. Accordingly, the battery lasts for 120 days (2000 mAh/(3.46 % · 20 mA · 24 h)). Crystal w/ Whisper (coll.) (u=0) at the same transmission power has a duty cycle of 1.18 %, and thus, the battery lasts for 353 days (2000 mAh/(1.18 % · 20 mA · 24 h)). As a result, in case of no data transmission (i. e., $u = 0$) Whisper used in Crystal increases the network lifetime by a factor of 2.9 compared to "standard" Crystal. Considering $u = 5$, i. e., among the 27 nodes in FlockLab, there are 5 data packets to transmit during an epoch, the duty cycle of Crystal ($u = 5$) is 3.49 % and the duty cycle of Crystal w/ Whisper (coll.) ($u = 5$) is 1.5 %. Using the computation described above, Crystal with Whisper has a 2.3 times higher network lifetime compared to Crystal without Whisper.

lifetime depends on the size of the packets sent in the T-slot as well as the number of retransmissions. A large packet requires a long duration of the T-slot compared to short packets. The longer the duration of the T-slot, the higher is the gain in terms of energy-efficiency that Whisper is able to achieve. However, one can also observe that Crystal with Whisper has a lower reliability compared to "standard" Crystal (e. g., 95.7 % vs. 98.4 % at −10 dBm with $u = 5$). Indeed, when the sink misses a packlet in a Whisper slot, it keeps its radio turned off during the T-slot and thus, also misses the data packet. Thus, Whisper requires high reliability when used as wake-up service. New sampling strategies or a higher value for N_{tx}, as shown in Section 4.3.4.2, can help to increase Whisper's reliability.

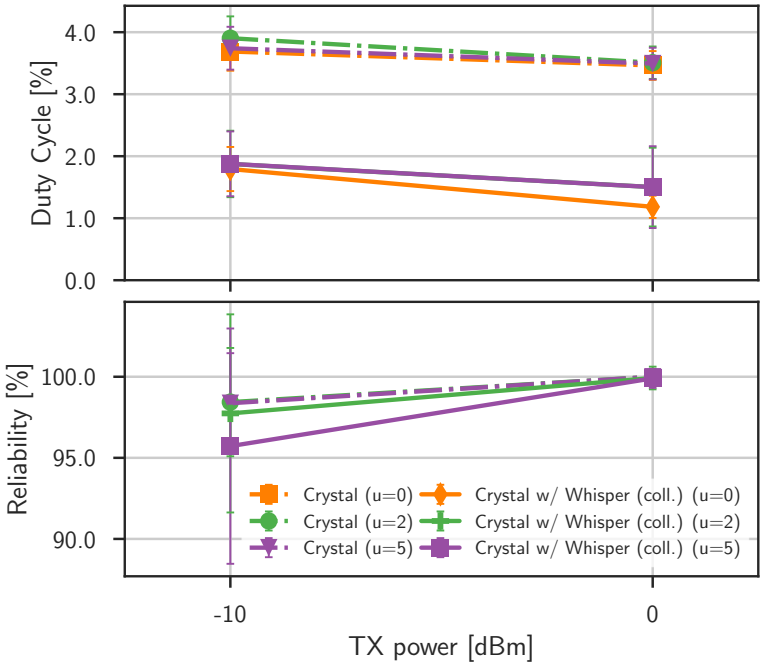


Figure 4.10: Impact of Whisper on Crystal. Whisper halves the duty cycle of Crystal. However, a missed packlet in Whisper also impacts the reliability of Crystal.

4.4 RELATED WORK

The overall architecture of Whisper builds on the concepts introduced by Glossy[12]. The novel design elements that we introduce – in particular packlets and direction-aware sampling – make Whisper significantly more efficient than Glossy. Whisper's superior performance is obtained by completely eliminating gaps between consecutive, synchronous transmissions.

12 This section is based on Section V of our previous work published in [22].

Lu et al. [104] already introduced in Flash the concept of consecutive packet transmissions after a single reception for rapid flooding. Later, *Lim et al.* [130] applied this concept in Glossy. In both approaches, consecutive packets are, however, not sent back-to-back as in Whisper but have gaps between them. This is because they are transmitted as individual packets and, thus, the radio must still perform an rx/tx turnaround even between consecutive transmissions. This increases the overall transmit time and strongly limits the use of sampling strategies as introduced in Whisper. Furthermore, due to the instability of the DCO, the alignment of synchronously transmitted packets decreases quickly with the number of packets. In Whisper, instead, the use of Flock – presented in Chapter 3 – and the concept of packlets guarantee that transmissions are precisely aligned. However, the competitive advantage of Whisper with respect to these approaches regarding the duration of a packet flood decreases with increasing payload size. This is because the time spent by the radio in rx/tx turnaround becomes negligible with respect to the time spent receiving or transmitting.

Approaches that exploit scheduled Glossy floods to provide high-level protocols – e. g., [50], or Crystal [73, 74] – could replace Glossy with Whisper to achieve a more efficient operation. Other, more complex protocols like Splash [37] or Pando [38] could also benefit from integrating Whisper's design in their architecture.

Several authors proposed protocols to use some form of frequency diversity to make synchronous transmissions more robust against interference. These include full-fledged protocols like Splash [37] or Pando [38] but also improved versions of Glossy like the one proposed by *Sommer and Pignolet* [153]. While we have not yet implemented the use of multiple channels within Whisper, this is part of our future work. However, we have shown on the example of Whisper in Crystal that Whisper already achieves high reliability when flooding on different channels.

Other approaches related to Whisper are those that provide – or can be used to implement – a network-wide wake-up service. Some basic techniques like Low-Power Listening [128] or Backcast [43] have been successfully used in Medium Access Control protocols to schedule nodes' rendezvous [24, 42, 114, 128]. They are however contention-based approaches and are inherently perform worse – both in terms of reliability and latency – than approaches based on synchronous transmissions.

Lastly, protocols that build upon wake-up radios, e. g., Zippy [161], ALBA-WUR [159], or the approach proposed in [101] rely on nodes that are equipped with special hardware. Thus, such approaches are orthogonal to Whisper.

4.5 SUMMARY

In this chapter, we presented a novel protocol to support the quick yet energy-efficient distribution of event-based data portions with Whisper. Whisper is a communication

primitive that builds upon synchronous transmission-based flooding with a packet-in-packet approach allowing to easily create energy-efficient sampling strategies. We present two sampling strategies: lazy sampling and direction-aware sampling. Our evaluation in the FlockLab testbed showed that Whisper can disseminate data twice as fast as Glossy with no loss in reliability.

CHAPTER 5

EVENT-BASED ONE-TO-ONE COMMUNICATION

In the previous chapter, we presented a quick yet energy-efficient approach to flood small amounts of data into a low-power wireless network. In particular, we considered small data like configuration parameters, and "wake-up calls" announcing bigger data traffic to reduce the nodes' overall energy consumption. In the latter case, the announced data can consist of bulk data that is transferred possibly using standard Internet protocols.

Using standardized Internet protocols like CoAP [19], HTTP, TCP, UDP, and IPv6 in low-power wireless networks opens new possibilities and use cases. For example, it enables the interoperability between devices of different vendors and link-layer technologies like IEEE 802.15.4, IEEE 802.15.1 (Bluetooth) and IEEE 802.11 (Wifi) [60] as schematically illustrated in Figure 5.1. In the figure, a border router acts as gateway between the low-power wireless network and the Internet, and thus, enables the date exchange over different link-layer technologies. This allows for interconnecting different production plants and locating the control plant of *"mines and onshore and offshore oil and gas field[s]"* [172] to *"near population centers where people prefer to work and live"* [172]. Standard Internet protocols often support the communication between two devices in a one-to-one manner, e. g., a CoAP client located in the Internet requests a batch of data (e. g., (multimedia) sensor data [3] or traffic monitoring information [131, 132]) from a specific node in the network.

The data that is exchanged using standard Internet protocols can be quite large. For example, a CoAP request may have several tens of byte [85]. Using synchronous transmission based protocols like Glossy for the data exchange implies flooding the low-power wireless network with large packets that are addressed to a single destination. The consequence is that nodes which (1) do not actively help forwarding the message to the destination, and (2) are not the destination consume unnecessary energy.

In this chapter[1], we present a novel approach to enable efficient one-to-one communication using standard Internet protocols based on synchronous transmission. In particular, we introduce LaneFlood. LaneFlood allows quick yet energy efficient one-to-one communication. As discussed in Section 2.2.4, Internet protocols in low-power wireless networks are typically deployed on top of the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL). RPL provides one-to-many and one-to-one routing. However, RPL has to maintain routes by selecting parent nodes, announce routing metrics, discover neighbors

1 This section is an extended version of Section I and Section III-E of our previous work published in [20].

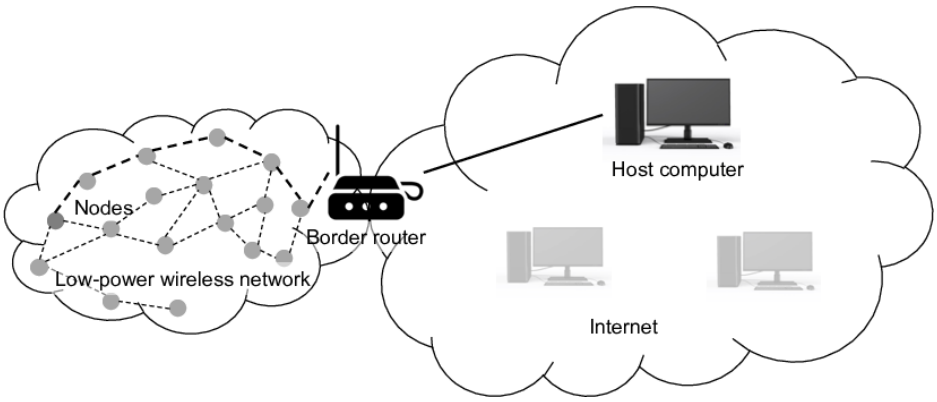
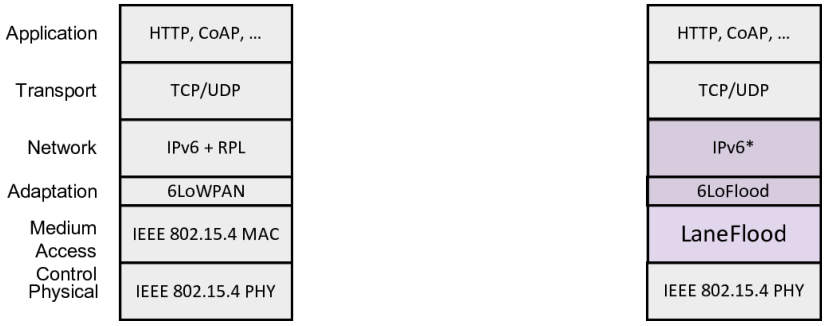


Figure 5.1: Interconnection of a low-power wireless network with the Internet. The border router is the access point to and from the low-power wireless network.

and maintain routing tables, and can only slowly adapt to network changes, unlike LaneFlood. LaneFlood quickly establishes a path between any source and destination in the network. Once the path is established, LaneFlood involves only the nodes along that path in the forwarding of data. Inactive nodes turn their radio off to preserve energy and periodically wake up to be available for further connections. Thus, LaneFlood has no need for routing tables or neighbor discovery services.

LaneFlood aims to efficiently support established Internet protocols like HTTP, CoAP, UDP, TCP, and IPv6. In particular, LaneFlood's design reflects the one-to-one communication pattern of these protocols: First, it sets up a connection between any two nodes in the network and afterwards facilitates the data exchange between these two nodes. For example, a CoAP request message triggers a connection setup, and LaneFlood maintains this connection until it is either closed or has timed out. LaneFlood builds upon two existing protocols exploiting synchronous transmissions: Glossy and the Concurrent Transmission Forwarder Selection (CXFS) protocol [25]. More precisely, it uses Glossy-floods to distribute messages and it uses an approach inspired by CXFS to support the energy-efficient message flow between any two nodes in the network. LaneFlood has, thus, three key advantages compared to existing solutions: (1) energy-efficient network operations for large, event-based data traffic, (2) full and transparent integration into the TCP/IP stack, and (3) native support for one-to-one communication.

Figure 5.2a shows the typical IP-based network stack considered for low-power wireless networks, similar to the one discussed in Section 2.2.4, while Figure 5.2b shows the network stack of nodes running LaneFlood. The latter figure illustrates that LaneFlood eliminates the need of incorporating routing protocols such as RPL [23] as well as MAC protocols like CSMA as in burst forwarding [41] or Disco [177]. Nodes do not need to perform neighbor discovery, maintain routing tables or deal with channel contention, allowing the



(a) Conventional IP-based IEEE 802.15.4-based network stack. *(b)* Network stack with LaneFlood.

Figure 5.2: Conventional IP-based IEEE 802.15.4-based network stack vs. network stack with LaneFlood. LaneFlood allows for lightweight implementations of IPv6 and 6LoWPAN, indicated with IPv6* and 6LoWPAN for LaneFlood (6LoFlood), respectively. A conceptional description of 6LoFlood is provided in Appendix A.1.

nodes to quickly adapt to channel fluctuations. Therefore, nodes running LaneFlood only need a lightweight implementation of IPv6 and 6LoWPAN, indicated as IPv6* and 6LoFlood in Figure 5.2b. In the following, we discuss LaneFlood, a routing-free protocol that enables to efficiently run standard Internet protocols based on IPv6* within a low-power wireless network. For the interested reader, we provide in Appendix A.1 a conceptual description of 6LoFlood that allows for packet fragmentation, header compression and global IPv6* addressing to support the interconnection of the low-power wireless network with other networks, e. g., the Internet.

The remainder of this chapter is structured as follows: We first introduce the terminology that we use and present the basic operations of LaneFlood in Section 5.1. Afterwards, we explain how a path is created in Section 5.2, and in Section 5.3, we show how LaneFlood operates. Section 5.4 discusses how LaneFlood transparently runs Internet protocols, and Section 5.5 reports our evaluation results. We review related work in Section 5.6 before summarizing and concluding this chapter in Section 5.7. Parts of the contributions presented in this chapter have been published in [20].

5.1 TERMINOLOGY AND BASIC OPERATION

LaneFlood creates an exclusive communication channel, called *lane*, between any two nodes – henceforth referred to as *source* and *destination* – in the low-power wireless network[2]. The lane is created through two executive network floods, initiated by source and destination, and involves only the nodes that are required for packet forwarding. Figure 5.3 illustrates how a lane is established. In particular, Figure 5.3a shows what we call the *Setup flood*,

2 This section is an extended version of Section II and Section III of our work previously published in [20].

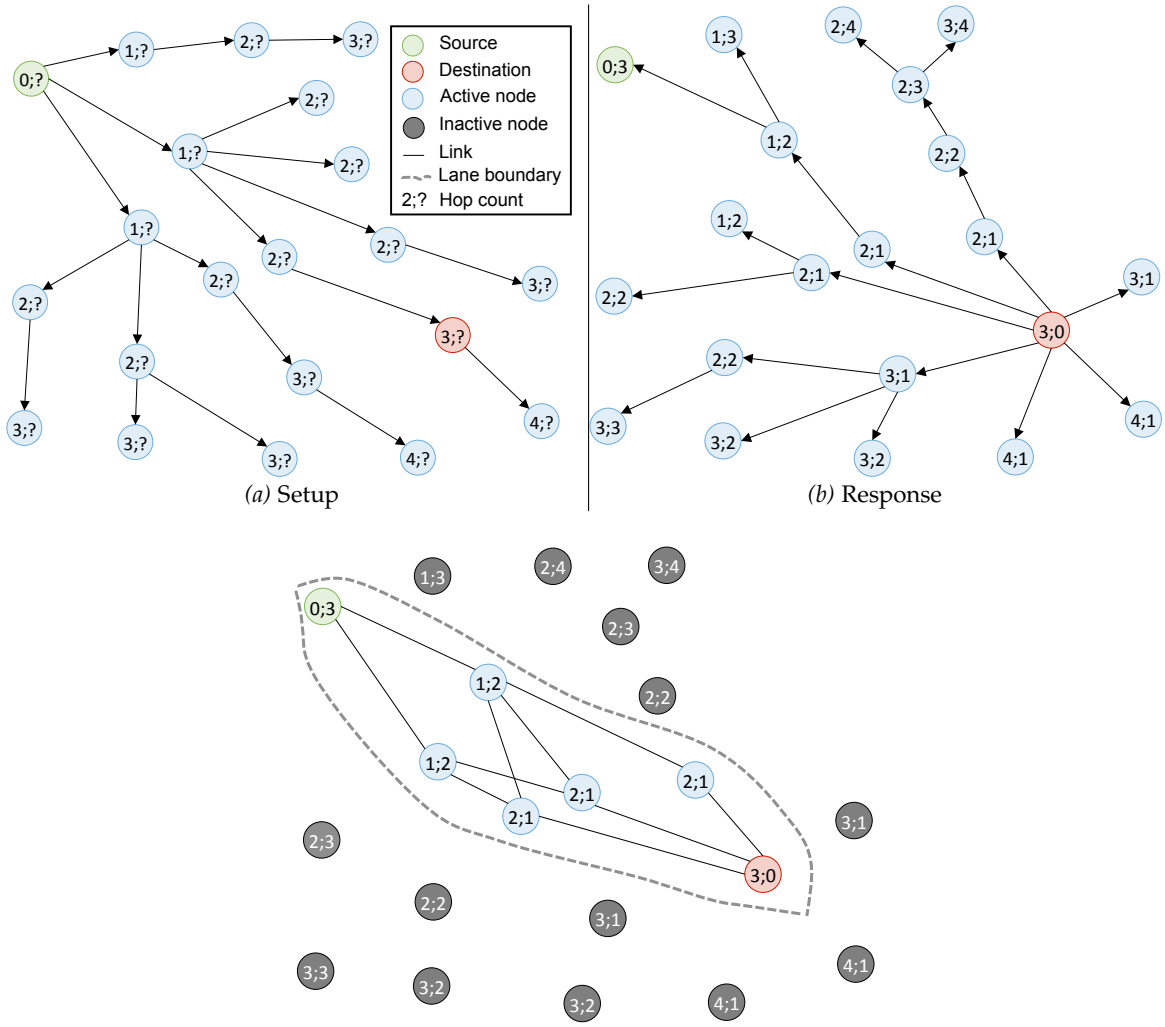


Figure 5.3: Establishing a lane. A lane is created by two consecutive network floods. First, the source disseminates a *Setup message*. Triggered by this message, the destination disseminates a *Response message*. The nodes compute whether they belong to the lane or not. If not, they turn their radio off until the next session, otherwise the help in forwarding messages during the *Data exchange*.

i. e., the first flood initiated by the source. The destination replies to the *Setup flood* with a *Response flood* as shown in Figure 5.3b. *Setup flood* and *Response flood* are disseminated using *network floods*, i. e., Glossy-floods initiated by either source or destination. The information collected during these two floods is used by the nodes to decide autonomously whether they belong to the lane or not. A node that belongs to the lane keeps its radio on to enable the data exchange using *Data floods* between the source and destination. Otherwise, it turns the radio off until a new lane must be created. Source and destination exchange data using *lane floods*, i. e., network floods propagated by nodes being part of the lane only, as long as the lane is active. Figure 5.3c illustrates an active lane.

During the *Setup, Response* and *Data flood*, the nodes propagate LaneFlood messages that correspondingly dubbed *Setup, Response, and Data messages*. Figure 5.4 shows the general format of these LaneFlood messages. The first byte is the *relay counter* c from Glossy, as discussed in Section 2.4.3. The relay counter is increased each time a message is retransmitted during a flood. We eliminated the Glossy fields that are not required by LaneFlood, leaving only the relay counter c. The next field in the LaneFlood message is the *type* field. With this field, nodes are able to distinguish the different LaneFlood messages. Table 5.1 summarizes the available messages. The type *Glossy* indicates that a source node must disseminate a message to all nodes in the network instead of to just a single node. Thus, instead of establishing a lane, it just propagates its message using network-wide Glossy floods. The *Sync*-type indicates a network synchronization, as we discuss in Section 5.3. The type field is followed by the *src* field. It indicates the identifier of the source and the field after, the *dest* field, is the identifier of the destination. The *dist* field holds the actual information that the nodes collect during lane establishment, i. e., it contains the minimal path between source and destination. We discuss how the nodes determine the minimal path and how they derive the lane from this path in detail in Section 5.2.

Figure 5.5 shows the protocol operation of LaneFlood. In particular, LaneFlood operates in *sessions* T_s and each session is further split in multiple *rounds* T_r. The session in Figure 5.5 contains for example of six rounds. Each round consists of two parts. In the first part of each round the application(s) running on the nodes operate and, e. g., collect sensor data or

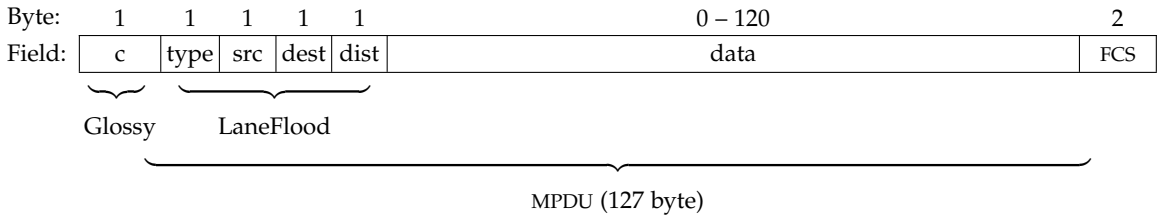


Figure 5.4: LaneFlood message structure. The relay counter c, which is provided by Glossy, is used to determine the distance measured in hops between nodes.

Table 5.1: LaneFlood message types. Using the *Glossy*-type, nodes disseminate messages to all nodes instead of creating a lane.

0	Glossy
1	Sync
2	Setup
3	Response
4	Data

perform computations. During this time the nodes have their radios turned off. The second part of a round is used for communication. We refer to this part as the *communication slot* T_c, accordingly. A session ends with a short guard period T_g that ensures that all nodes are ready for the next session. Section 5.3 provides a more detailed description of the protocol operation of LaneFlood.

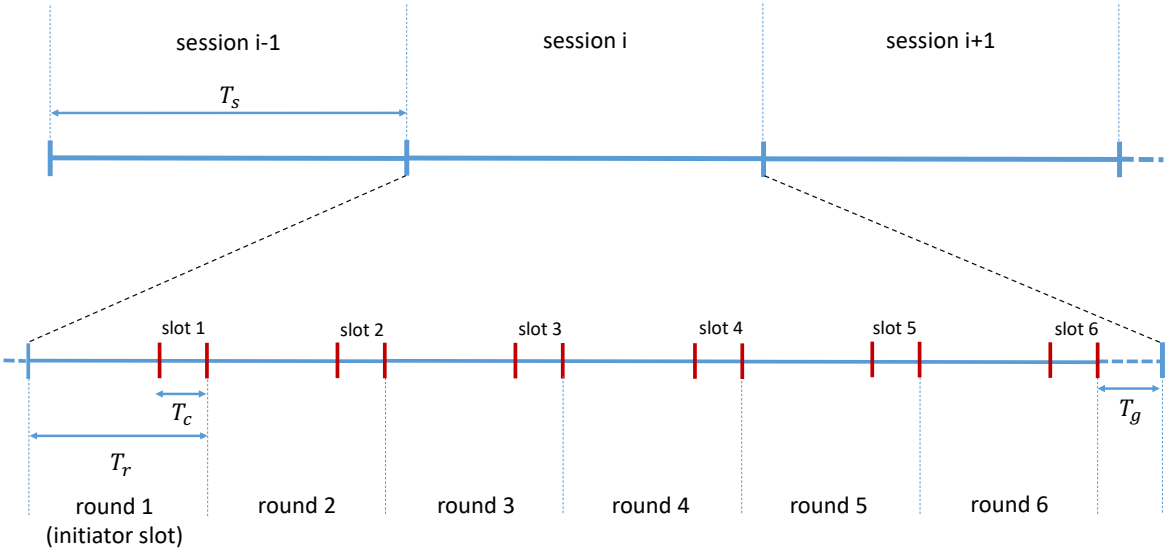


Figure 5.5: LaneFlood's protocol operation.

5.2 ESTABLISHING A LANE

In this section[3], we discuss the establishment of a lane in LaneFlood. In particular, we describe how nodes collect distance information and how a node decides whether it belongs to the forwarder set or not. The protocol operation of LaneFlood is described in the following Section 5.3.

3 This section is based on Section III-A and Section III-B of our work previously published in [20].

5.2.1 *Collecting information*

A lane is established through two consecutive network floods that are initiated by the source and the destination. To this end, the source transmits a *Setup message*, i. e., a LaneFlood message of type *Setup*. In the message, the source includes its identifier and the one from the intended destination in the corresponding fields. Further, the node appends the data it must send (e. g., a TCP-SYN for establishing the TCP handshake, or a CoAP request). The *distance* field and the *relay counter* are set to zero. Nodes that receive the *Setup message* use the relay counter c to determine their distance d_{sf} in hops to the source, as done in [25] and shown in Figure 5.3a. When the Setup message arrives at the destination, the *relay counter* field holds the distance in hops between source and destination d_{sd}. For example, $d_{sd} = 3$ in Figure 5.3a .

Next, the destination of the *Setup message* sends a *Response message*, i. e., a LaneFlood message of type *Response*. The destination adds its identifier in the *source* field and the one of the source into the *destination* field. The destination can also append its data to transmit (e. g., the TCP-SYN-ACK as response to the TCP-SYN message, or a CoAP response message). Further, the destination sets the *distance* field to the value of d_{sd}. Alike the *Setup message*, the nodes receiving the *Response message* can read their distance to the destination d_{df} from the *relay counter* field, shown in Figure 5.3b. After the *Response message* has been disseminated, the nodes know the values of d_{sd}, which is included in the distance field of the Response message, as well as their distance to source and destination d_{sf} and d_{df}, respectively [25]. The *Setup message* and *Response messages* are disseminated using Glossy-floods. Thus, it is expected that all nodes in the network receive these messages with high probability and low latency. Using the collected distance information, the nodes determine whether they belong to the current lane or not, as depicted in Figure 5.3c. In the following we show how nodes decide if they belong to the lane.

5.2.2 *Decision making*

After having received the *Response message*, the nodes not being source or destination decide, whether they belong to the lane or not. A node always belongs to the current lane if it fulfills the condition:

$$d_{sf} + d_{df} \leqslant d_{sd} + s_i, \tag{5.1}$$

with s_i being the integer part of a configurable protocol parameter: the *slack* s. Condition 5.1 includes nodes to the lane that are either (a) on the minimal path between the source and the destination, and (b) on the non-minimal paths that are at most s_i hops longer than the minimal path. This methodology corresponds to the forwarder selection

strategy of CXFS [25]. The optimal value of s_i depends on the current network topology. However, if the lane contains only nodes on minimal paths, i. e., setting $s_i = 0$, it may cause the lane to consists of single-links due to the small number of nodes helping in forwarding messages. Thus, the advantages of synchronous transmissions are not exploited and the lane may become unreliable. Increasing s_i, and thus, including more nodes to the path helps in improving reliability, but might in turn cause an unnecessarily high number of forwarders being active, which increases the overall energy-consumption in the network. Thus, in practical settings the number of nodes helping in forwarding messages should be "somewhere in between" s_i and $s_i + 1$. In other words, we consider s_i to be small and further include additional nodes to the lane. As a result, we add the following composite condition:

$$[d_{sd} + s_i < d_{sf} + d_{df} \leqslant d_{sd} + s_i + 1] \cap [\text{rand}(0,1) \leqslant s_d], \tag{5.2}$$

where s_d corresponds to the fractional part of the slack s. For example, if $s = 2,30$, then this equals to $s_i = 2$ and $s_d = 0.30$. Condition 5.2 implies that nodes that lie on paths that are $s_i + 1$ hops longer than the minimal path are part of the lane with probability s_d. In this way, we can improve the packet reception reliability by increasing the number of nodes that are part of the lane in a far more fine-grained way compared to just considering integer values of the slack s_i. Thus, this allows us to save energy without sacrificing reliability.

5.3 THE PROTOCOL OPERATION OF LANEFLOOD

After discussing how a lane is created, we now explain how LaneFlood operates[4]. To this end, we consider an example scenario where the initiator (e. g., the border router) requests sensor data from a node using CoAP. As illustrated in Figure 5.5, the first round of a session is reserved for the initiator. In this round, the initiator either transmits a *Sync message* or a *Setup message*. The initiator transmits a *Sync message* in case it has no pending data. The *Sync message* ensures that the nodes in the network re-synchronize to the reference time of the initiator. Further, it signals the nodes that they are free to establish a lane in the next round by transmitting a *Setup message*. If the initiator, however, has data to transmit in the first round – as in our current example – it transmits a *Setup message*. All other nodes with pending data try to establish a connection in the next session. In the second round, the nodes expect a *Response message* from the destination. After this message is received, the nodes compute whether they belong to the forwarder set or not, as described in Section 5.2. The data is exchanged starting from the third round until the end of the current session with only the nodes that are part of the lane. All other nodes keep their radio off until the beginning of the next session. In each round, one packet is exchanged using lane floods. In

4 This section is based on Section III-D of our previous work published in [20].

case source and destination complete the data exchange before the session ends, the nodes keep listening for two consecutive rounds for potentially incoming messages. After these two "idle rounds", the lane is released and the nodes turn off the radio until the beginning of the next session. When source and destination are not able to finish the data exchange within one session, they continue in the next session. Irrespectively of whether and when a data exchange ends within a session, when the new session starts nodes compete to establish a new connection.

Because we assumed the initiator to be the source in our current example, the data exchange in the next session is guaranteed in the initiator slot. When the initiator transmits a *Sync message* in the first round of a session, all other nodes with pending data can establish a lane in the second round. In this round, several nodes may want to establish a lane and thus, they all transmit a *Setup message*. These messages compete against each other with three possible outcomes:

1. The *Setup messages* collide destructively and all *Setup messages* are lost.
2. Only one of the *Setup messages* is received by its intended destination.
3. Two or more *Setup messages* are received by their intended destination.

When the first case occurs, no *Response message* is disseminated in the subsequent round. This is detected by the nodes that transmitted the *Setup message*, and thus, they try to send the message again in the next round. When the second case occurs, the only destination that has received the *Setup message* replies with a *Response message* in the next round. As a result, a lane between a source- and destination-pair has been successfully established and the two nodes can continue with the data exchange as described above. The other senders also receive the *Response message* and thus, detect that two other nodes established a lane. Thus, they either keep their radio on to act as forwarders or turn their radio off. In either way, they try to establishment a new lane in the next session. In the third case, all destinations that have received a *Setup message* reply with a *Response message*. Thus, the transmitted *Response messages* are sent concurrently and compete again with three possible outcomes:

1. All *Response messages* are lost;
2. Only one of the *Response messages* reaches its intended destination;
3. Two or more *Response messages* reach their intended destination.

In the first case, the sources detect the missing *Response message* and try to disseminate the *Setup message* again in the following round. In the current version of LaneFlood, nodes try to establish a lane until the lane is successful established or until they received a

Response message from another source-destination-pair. If the second case occurs, one lane is successfully established between a source- and destination-pair. As before, the other sources detect that lane is established and either act as forwarders or turn their radio off. If the third case occurs, multiple lanes have been established concurrently. Since both *Setup* and *Response messages* have been successfully delivered, it is likely that several lanes can co-exist at the same time in the network.

5.4 RUNNING INTERNET PROTOCOLS ON TOP OF LANEFLOOD

Standard Internet protocols like, e. g., CoAP [19], generate communication requests on-demand, i. e., at any point in time[5]. LaneFlood, however, is a time-slotted protocol that operates according to a fixed schedule, i. e., it transmits data only at pre-specified time instants. To support on-demand traffic from application protocols like CoAP, LaneFlood uses a packet queue to gather messages from the upper layers. More precisely, this queue receives packets from layers sitting above LaneFlood and we refer to it as the *tx queue*. Once a packet is generated by the application and passed down the stack, LaneFlood first enqueues it in the tx queue.

In each round, before the communication slot starts, each node verifies whether it has any packet in its queue. If so, the node further verifies whether it is allowed to transmit in the current round. This is the case, e. g., when a *Sync message* was transmitted in the previous round. If the tx queue of a node contains at least one message and the node is allowed to send, it transmits a *Setup message* with a copy of the first message from its tx queue as payload. The packet is removed from the queue if and only if – by the end of the current communication slot – the node has received its own packet at least once. This implies that the packet has been forwarded by neighboring nodes and, thus, has most likely also reached the destination. If, instead, the node receives the packet of a contending sender, it passes this packet to the upper layer and keeps its own packet in the tx queue. A new transmission attempt is started in the next allowed round.

5.5 EVALUATION OF LANEFLOOD

In this section[6], we evaluate the performance of LaneFlood through extensive testbed experiments. We show that our forwarder selection strategy achieves a lower duty cycle and a higher reliability compared to CXFS and Glossy by choosing the number of participating nodes with more granularity. We also find that LaneFlood can transport Internet protocols with an end-to-end latency that can be tuned to less than 300 ms.

5 This section is based on Section III-F in [20].

6 This section is an extended variant of Section IV in [20]. Please note that we have further improved LaneFlood, and thus, the evaluation results in this thesis have been improved compared to the results shown in [20].

5.5.1 *Methodology*

We run LaneFlood in the FlockLab testbed, which we have described in Section 2.1.3. For this set of experiments, we consider a scenario in which a node sends CoAP requests to different nodes in the network. LaneFlood is used as underlying communication protocol. More precisely, the initiator, acting as client, requests a batch of data (e. g., sensor data or traffic monitoring information) from specific nodes in the network. The client sends requests of 90 byte. The time instants at which requests are generated are distributed within a random interval of [1,11] seconds. Thus, a request is sent on average every 5 seconds. The client sends 125 requests to each server before switching to the next one. We select the node with identifier 1 as our client. The target nodes in the (servers) reply immediately with 5 packets and a total packet size of 125 byte. Table 5.2 shows the server identifiers and their distance to the client. Client and servers have IPv6 addresses and application messages are transported through UDP.

We focus on three key performance metrics: reliability, latency, and duty cycle. The *reliability* is the ratio of received messages and total messages sent by the application. We measure the reliability of packets received at the client. The *latency* is measured end-to-end (at the application level) and describes the time interval between the sending of a packet and its reception. We distinguish between the latency of a packet transmitted from the client to the server, and the latency from server to client. The *duty cycle* indicates the ratio of the total time the radio of a node is on during an experiment and the total duration of the experiment. The duty cycle is averaged over all nodes within the network.

We set the duration of a session and a round to 4 s and 200 ms, respectively, and run each experiment at least 3 times for 1 hour. Results are averaged over the 3 runs, and error bars in the plots indicate the 5th and 95th percentiles.

5.5.2 *Impact of the slack*

EXPERIMENTS We first evaluate the impact of the slack s in different network topologies. We set the transmission power to -10 dBm and $s = 0,00$. We steadily increase the slack until all nodes participate in the data exchange. This corresponds to network-wide

Table 5.2: Server settings.

Server [node id]	*Distance to client [hops]*
16	2 to 3 (short distance)
13	3 to 4 (middle distance)
7	4 to 7 (long distance)

Glossy-floods and thus, we indicate this scenario with $s = \text{Glossy}$. A slack that only consists of the integer part, e. g., $s = 0,00$, $s = 1,00$, $s = 2,00$, and $s = 3,00$, corresponds to the use of CXFS [25]. We repeat the experiments with a transmission power of 0 dBm. Using a transmission power of 0 dBm creates a *dense topology* with only a few hops between client and server. Consequently, running LaneFlood with transmission power −10 dBm creates a *sparse topology* with long distances between client and server.

OBSERVATION 1 *The higher the slack, the higher the duty cycle.*
Figure 5.6 and Figure 5.7 show our evaluation results for the sparse and dense topology, respectively. In particular, the first and upper plots of Figure 5.6 and Figure 5.7 show the radio duty cycle for different slack values. Note that we did not increase the slack continuously at the same rate. We marked the areas where we changed the rate with dashed lines in the figures. As expected, the duty cycle increases with the slack, with a data exchange using Glossy-floods resulting in the highest duty cycle. Furthermore, the upper peak of the error bars indicates the duty cycle of the nodes participating in lane floods, while the lower peaks of the error bars marks the duty cycle of nodes that are not part of the data exchange and thus, entered the sleep state.

OBSERVATION 2 *A higher slack increases the reliability until it reaches a peak.*
The second plots in Figure 5.6 and Figure 5.7 display the impact of the slack on the reliability. The reliability increases until it reaches its maximum. Since more nodes participate in the data exchange and help in forwarding packets, the reliability increases. The reliability reaches its peak in the sparse topology at $s = 1,90$ and in the dense topology at $s = 0,90$. After reaching the peak, the reliability either remains constant or even decreases slightly as in the dense topology shown in Figure 5.7. At the peak, we have reached a saturation of participating nodes. The phenomenon of the reliability drop using Glossy in dense networks has been already observed by other authors and is often referred as *scalability problem* [36, 52, 112, 176]. As stated by *Mohammad et al.* [112]: *"This decrease in reliability with an increasing number of synchronous transmitters can be explained by the increasing likelihood of larger temporal displacements among synchronous transmitters [43], which can be caused by a combination of software, hardware, and signal propagation delays [185]."*

OBSERVATION 3 *The random slack achieves a higher reliability with a low network duty cycle compared to CXFS.*
We already showed that the reliability reaches a peak after which adding more nodes to the lane negatively affect the network duty cycle without increasing the reliability. The goal is, thus, to find the reliability peak with the lowest duty cycle. While the boundary b in CXFS is a fixed integer, our random slack approach allows for a fine-grand selection of nodes participating in a lane. This allows us to achieve a high reliability with only the amount

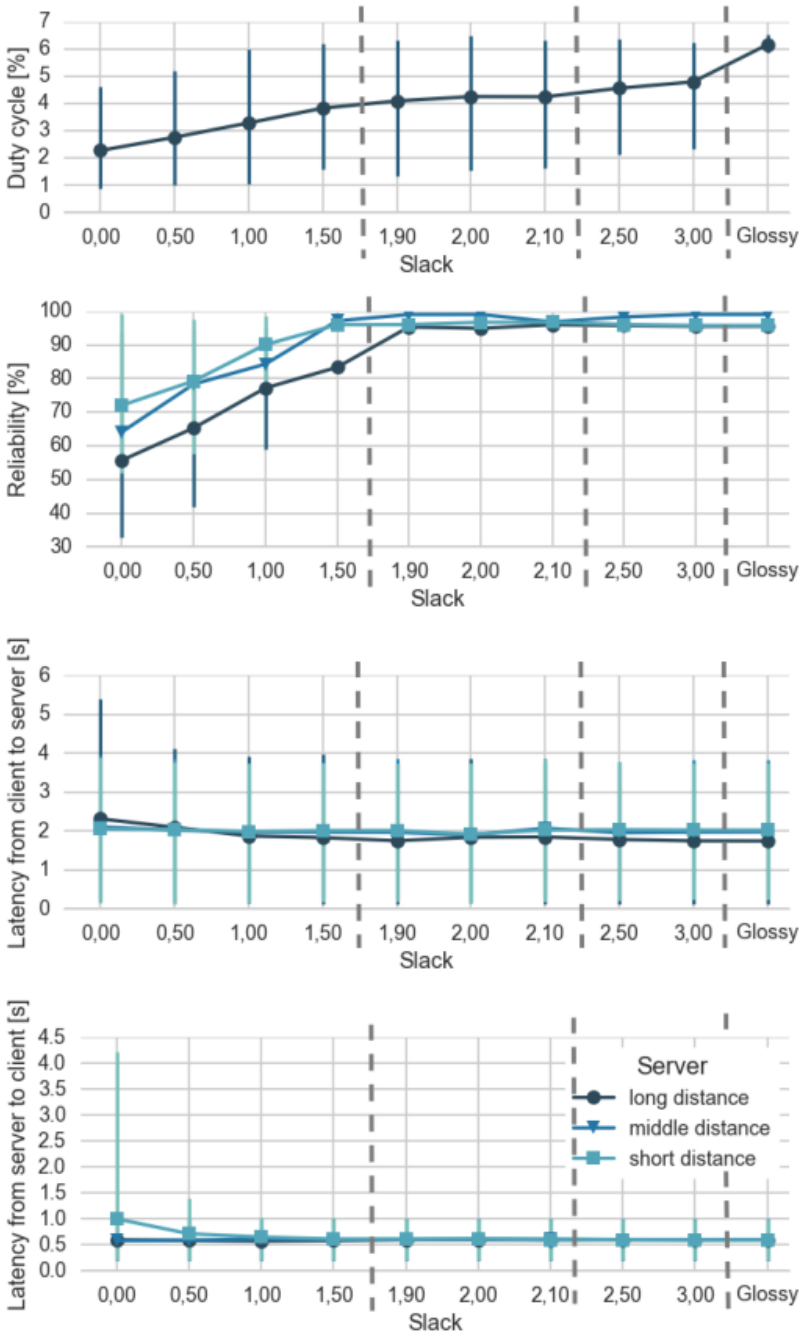


Figure 5.6: Impact of the slack in the sparse topology in FlockLab.

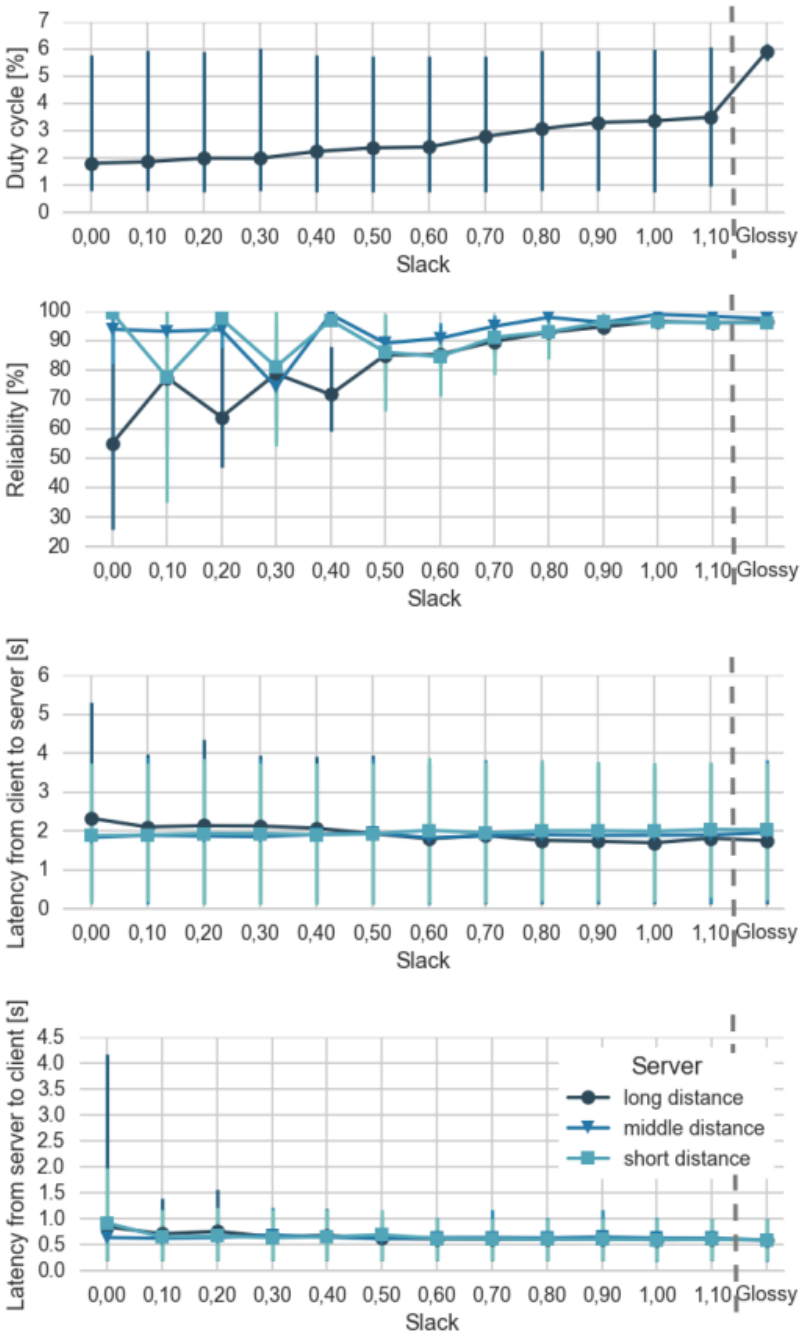


Figure 5.7: Impact of the slack in the dense topology in FlockLab.

of nodes that are necessary to forward the packets between client and server, reducing the energy consumption in the network.

OBSERVATION 4 *The optimal slack value for a client-server-pair depends on the topology.*
The optimal slack value regarding high reliability and low duty cycle is determined by the topology. For example, in the sparse topology, the servers achieve the highest reliability at around $s = 1,90$, shown in the second plot of Figure 5.6. The highest reliability in the dense topology is achieved at $s = 0,90$, shown in the second plot of Figure 5.7. Thus, the slack has to be adjusted according to the current topology.

OBSERVATION 5 *The latency is independent of the slack.*
The third plots in Figure 5.6 and 5.7 show the client to server latency, while the last plots of the figures show the server to client latency. In all four figures, the latency does not change with the slack, which indicates that the latency is independent of the slack value.

5.5.3 *Impact of the session and round length on latency*

We now evaluate the impact of the session and round length on the performance of LaneFlood. In this set of experiments, we run LaneFlood in the sparse topology with transmission power −10 dBm and set the slack to $s = 1,00$. We first run LaneFlood with a round length $T_r = 200$ ms and repeat the experiment with $T_r = 100$ ms.

OBSERVATION 6 *The server to client latency increases with the round length* T_r.
Figure 5.8 shows the reliability (upper plot, left), the duty cycle (upper plot, right), the client to server latency (lower plot, left), and the server to client latency (lower plot, right) per round duration. We observe from the figure that the server to client latency (lower plot, right) increases for all servers with the round length. More precisely, doubling the round duration increases the server to client latency by a factor of two. The client to server latency (lower plot, left), the reliability (upper plot, left) and the duty cycle (upper plot, right) remain constant. This is expected since the latency from server to client is mainly determined by T_r and the position of a packet in the tx queue. More specifically, after receiving the *Setup message* (i. e., the CoAP request) the server enqueues all data packets in the tx queue before transmission. In our evaluation scenario, the server creates 5 packets to send to the client but in each round only one packet is transmitted. Thus, the average server to client latency for the first packet in the tx queue is $T_r/2$. This is because we start measuring the server to client latency when the packet is created, and thus, before the communication slot starts. The average latency from server to client for each following packet increases by T_r, assuming the previous packet was transmitted correctly. The 5th

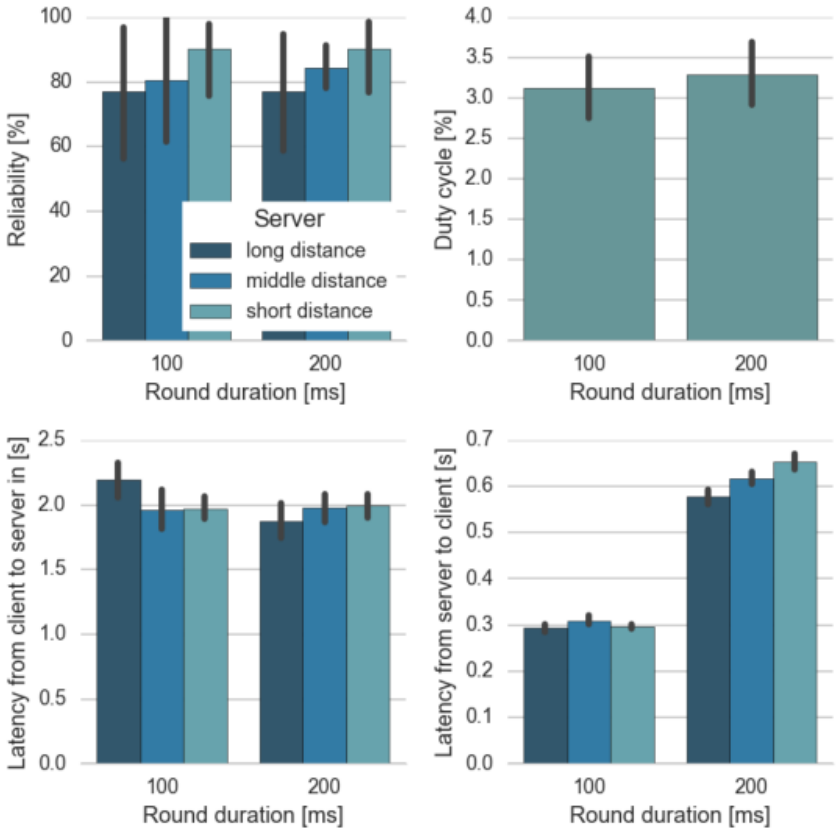


Figure 5.8: Impact of the round length on the performance of LaneFlood. Increasing the round length T_r increases the latency from client to server.

packet has, thus, an average server to client latency of $T_r/2 + 4 \cdot T_r$. Reducing T_r decreases the time until a packet is scheduled and hence, the overall server to client latency.

OBSERVATION 7 *The client to server latency can be reduced by decreasing the session length* T_s. Figure 5.6 and Figure 5.7 (third plot) show equally that the average client to server latency is ≈2 s. This is expected since the client to server latency is mainly influenced by the value of T_s, which we set to 4 s. Since LaneFlood runs completely detached from the application, it issues *Setup messages* from the client on average every $T_s/2$ seconds. Reducing T_s, thus, allows to reduce the client to server latency. However, a lower value of T_s results in (a) a higher duty cycle because *Setup* and *Response messages* are exchanged more often, and (b) less rounds available for data exchange (assuming T_r is fixed). This, in turn, may result in higher packet drops due to a full tx queue. In the current implementation of LaneFlood, the tx queue can hold 35 packets.

5.6 RELATED WORK

As discussed in Section 2.4.2, Glossy sparked a new research direction in low-power wireless networks protocols[7]. Several approaches improve, build upon or analyze Glossy. For example the Low-power Wireless Bus (LWB) [50] uses Glossy-floods to create a virtual network bus. The Glossy-floods are scheduled at a central node to support one-to-many, many-to-one, and many-to-many traffic patterns. Chaos [90] builds upon Glossy and the capture effect to support quick and efficient all-to-all data sharing. Splash [37] and P^3 [36] add pipelining to Glossy-like floods to increase reliability and network throughput. Ripple [189] further improves the throughput of Splash by assigning different channels to different packets instead to different levels of the data dissemination tree. The approaches mentioned above (1) do not support one-to-one traffic or at least without forwarder selection, and (2) they do not consider Internet protocols or other high-level protocols like TCP, UDP or CoAP.

CXFS [25] and Sparkle [191] are protocols that build upon Glossy to provide one-to-one communication. We have already discussed the differences between CXFS and LaneFlood within this chapter. Sparkle builds upon the capture effect to find the most reliable path between a source and a destination. To this end, it lets every node that receives a packet modify it by setting a bit in a bitmap that corresponds to the node, before retransmitting the packet. When the destination receives the packet, it can read from the bitmap the probably most reliable path between itself and the source. Sparkle's forwarder selection mechanism is interchangeable with our approach. More precisely, LaneFlood could also operate using Sparkle's forwarder selection mechanism. However, based upon the results reported in [191], it remains unclear if Sparkle can provide higher end-to-end reliability than CXFS. RTF [193] builds upon Sparkle and focuses on improving reliability and energy-efficiency in point-to-point traffic. It uses TDMA for scheduling messages. LaneFlood does not require any scheduling mechanism as we assume on-demand traffic rather than constant traffic.

LaneFlood enables the seamless operation of standard protocols like TCP, UDP and CoAP. Instead, protocols like CXFS [25], Sparkle[191], and RTF [193] support only proprietary transport protocols. A recently presented approach by *Hewage et al.* [60] shows that LWB [50] can be used to relay messages from high-level, standard Internet protocols. However, this approach relies on a central entity that schedules communication. Instead, in LaneFlood connections are established spontaneously between arbitrary source and destination nodes. Furthermore, *Hewage et al.* [60] do not consider any forwarder selection mechanism in their approach. Their main focus is on maximizing throughput. Instead, LaneFlood can trade-off energy consumption against and reliability. Burst forwarding [41] runs TCP on top of it. However, burst forwarding is not based on synchronous transmis-

7 This section is based on Section V in [20].

sions, and thus, does achieve the reliability and energy-efficiency of Glossy-like protocols in low-power wireless networks.

5.7 SUMMARY

In this chapter, we presented LaneFlood. LaneFlood is a routing-less, one-to-one communication protocol based on synchronous transmissions that efficiently runs standard Internet protocols like HTTP, CoAP, UDP, TCP, and IPv6. To this end, it quickly establishes a lane between any source and destination in the network. Once the lane is established, it only involves the nodes that are required for forwarding the data. All other nodes turn their radio off to preserve energy. Our testbed evaluation showed that LaneFlood achieves a lower duty cycle and a higher reliability compared to CXFS [25] and Glossy by choosing the number of nodes being part of the lane with more granularity.

CHAPTER 6

CONCLUSION

Low-power wireless networks support a plethora of applications including, e. g., monitoring the air quality in urban areas or controlling the heating, ventilation, and cooling in large buildings. The use of low-power wireless networks in such monitoring and actuating applications has the advantage of deployment flexibility and low cost compared to wired solutions. These applications often have stringent requirements on the data exchange and thus, on the used communication protocols. More precisely, the data exchange must be highly reliable and energy-efficient with low latency. However, wireless communication is notoriously error-prone. Message losses happen often and unpredictably, making it challenging to support applications requiring high reliability and low latency.

A quick yet energy-efficient technique called synchronous transmissions facilitates the fulfillment of these requirements. Using synchronous transmissions, two or more nodes transmit an identical packet at almost the same time instance, causing the packets to collide constructively at the same receiver(s). The constructively colliding packets significantly increase the likelihood of being correctly received. A short radio activity, and thus, the energy-efficient operation of protocols based on synchronous transmissions relies on the simultaneous wake-up of the nodes and the quick and reliable network-wide packet dissemination.

In this thesis, we first investigated methods that can ensure high performance of synchronous transmissions, even in harsh environments, in which the nodes are typically deployed. After that, we used synchronous transmissions as a low-level communication service and focused on event-driven data distribution. In particular, this thesis makes three main contributions:

- *On-the-fly clock offset compensation.* We presented Flock, a protocol primitive that compensates clock frequency deviations among synchronously transmitting nodes, even when the nodes are exposed to harsh environmental conditions, like strong temperature and humidity fluctuations.

- *Fast flooding of event-based small data portions.* We introduced Whisper, a protocol primitive exploiting a packet-in-packet technique to quickly yet efficiently distribute small data in a low-power wireless network.

- *Event-based one-to-one communication.* We enabled with LaneFlood the energy-efficient, reliable, and quick distribution of large on-demand messages in low-power wireless networks using synchronous transmissions and standard Internet protocols.

In the following, we summarize the three main contributions of this thesis, discuss limitations, and sketch interesting future directions. We finally close this thesis with concluding remarks.

6.1 CONTRIBUTIONS

In our first contribution, we addressed the performance of synchronous transmission in harsh environments. In particular, we evaluated and quantified the impact of clock frequency deviations on the temporal displacement Δ_{Td} of synchronously transmitting nodes. We showed that small clock offsets among synchronously transmitting nodes – which are common in commodity nodes – significantly decrease the probability of achieving constructive interference. We further showed that this is exacerbated when the nodes are exposed to different temperatures. To counteract the issue of deviating clock frequencies, we proposed Flock: On-the-Fly Clock Offset Compensation. Flock compensates for clock frequency deviations among synchronously transmitting nodes. The contributions that Flock makes are twofold: First, it makes Glossy and Glossy-based protocols more robust to operate in challenging environments. Second, Flock allows for intermediate operations, like packet processing or channel switching, to be executed between packet reception and retransmission while ensuring non-destructive interference of signals. Our results based on simulations and experiments on real nodes confirm that Flock allows Glossy to achieve constructive interference in 98 % of the message transmissions even when nodes are exposed to temperature differences of 30 °C. Thereby improves Flock the overall performance of Glossy without additional message overhead.

The second contribution provided a protocol for quick yet energy-efficient distribution of small, event-based data, like configuration parameters, or "wake-up calls" to prepare nodes for incoming (bulk) data traffic. In this context, we presented Whisper, a communication primitive exploiting a packet-in-packet approach and synchronous transmissions. Whisper's strength lies in its quick and reliable data transmissions. For example, our evaluation on the publicly available FlockLab testbed showed that Whisper disseminates data twice as fast compared to Glossy with no loss in reliability. Whisper's design allows for creating sampling strategies to reduce the nodes' idle listening time – the time the nodes wait for incoming packets – for a highly energy-efficient operation. In this thesis, we presented two sampling strategies: lazy sampling and direction-aware sampling. While the latter strategy targets applications like data dissemination or collection, where the traffic flows in only one direction, lazy sampling can be used in any generic application. This makes Whisper a highly modular but efficient protocol for various application scenarios.

In our last contribution, we presented LaneFlood to enable the energy-efficient one-to-one communication of standard Internet protocols with. LaneFlood is a routing-free protocol based on synchronous transmissions. It establishes a communication *lane* between any two nodes in the network. Only the nodes along the lane that are required for forwarding messages are involved in the communication. All other nodes turn their radio off to preserve energy. Our evaluation on the FlockLab testbed showed that by fine-granularly choosing the number of nodes being part of the lane, LaneFlood achieves a lower duty-cycle as well as higher reliability compared to the state-of-the-art.

6.2 LIMITATIONS AND FUTURE DIRECTIONS

In this section, we discuss limitations of this work. Building upon the identified limitations, we suggest possible extensions and interesting research directions.

APERIODIC COMMUNICATION SLOTS In this thesis, we achieve with Whisper and LaneFlood event-driven data transfer with periodic communication slots. Data exchange can only occur during these communication slots. Senders and forwarding nodes can assume that their one-hop neighbors are ready to receive data instead of having to wait until nodes at each hop turn their radios on – as it is typical in CSMA-based approaches and other MAC protocols [24, 40, 117, 186]. Nonetheless, achieving event-driven data transfer with periodic communication slots is also clearly a limitation of Whisper and LaneFlood for several reasons. The simultaneous wake-up requires the nodes to be synchronized to a common and fixed time reference, which, indeed, (1) represents a single point of failure and (2) triggers communication among the nodes. The synchronization has to be scheduled recurrently (however, independently and less often than communication slots) due to frequency deviations of the nodes' clocks, as discussed in Section 3. For example, using LaneFlood, the initiator triggers the synchronization when it has no data to transmit in the first round. Despite having communication slots, (3) the nodes still have to sample for incoming event-based packets due to their unpredictable occurrence. Further, (4) pending data can only be transmitted during periodic communication slots and not immediately when it is available. Thus, the data transfer delays on average for half the period before being transmitted. We have shown and discussed this in the evaluation of LaneFlood in Observation 6 and Observation 7. On one hand, frequent communication slots allow for a small data transfer delay but result in high energy consumption. On the other hand, infrequent communication slots cause high delay but consume less energy. We believe that the design of Whisper allows researchers to develop energy-efficient sampling strategies that match aperiodic communication patterns, and thus, to design a communication protocol with aperiodic communication slots.

ULTRA-RELIABLE COMMUNICATION Ultra-reliable, low-latency communication – i. e., packet deliveries with loss rates of less than 10^{-9} and deadline guarantees – has been a major goal in, e. g., industrial automation systems, since the introduction of Fieldbus systems [166, 195]. While wired solutions are able to serve the stringent requirements of mission-critical applications, most wireless solutions only focus on process monitoring, despite the benefits promised when adopting wireless communication.

The approaches discussed in this thesis provide best-effort data deliveries and do not guarantee the timely delivery of packets within a given deadline, as required in ultra-reliable, low-latency communication. For example, nodes in LaneFlood contend for exchanging data. However, the source-destination-pair that "wins" the contention is not necessarily the pair with the most time-critical data. This is a major limitation of LaneFlood that also applies to Whisper. Using Whisper, the nodes know that there is pending data, however, they do not know how many senders contend for communication and with which priority. We believe that an interesting solution approach to enable ultra-reliable, low-latency communication could be to combine Whisper with a packet prioritization and scheduling strategy, and a neighbor counting solution similar to PoC [185]. Thus, during the Whisper phase, the nodes contend for the channel access and afterwards the node that has the packet with the most time-critical data can disseminate its packet without contention. This solution could be then applied to protocols like LaneFlood or Crystal [73, 74] to enable ultra-reliable, low-latency communication.

6.3 CONCLUDING REMARKS

The contributions reported in this thesis demonstrate the efficient distribution of event-based data with respect to reliability, latency, and energy-efficiency. To this end, we rely on synchronous transmissions and ensure their temporal displacement in the order of sub-microseconds in even harsh environments. We further used synchronous transmissions as a low-level communication service for protocols targeting the event-driven data distribution. We believe that the design of our novel protocols has taken a significant step towards highly reliable, low-latency communication of event-based data in low-power wireless networks.

BIBLIOGRAPHY

[1] Advanticsys. *XM1000*. [online] `https://www.advanticsys.com/wiki/index.php?title=XM1000`. 2017.

[2] G. Sierra Aiello, I. Demirkol, A. Calveras, C. Gomez, E. Garcia Villegas, and A. Betzler. "Competition: Interference-Aware Multi-Channel Cross Layer Protocol for Energy-Efficient and Low-Delay Networking." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[3] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. "Wireless Multimedia Sensor Networks: A Survey." In: *Computer Networks* 14 (6 2007).

[4] H. A. H. Alhalabi, T. C. Wan, L. Missif, and M. Ehalabi. "Performance Analysis of the Constructive Interference Flooding in Wireless Sensor Networks." In: *Proceedings of the International Conference on Information Technology (ICIT)*. 2017.

[5] D. Amaxilatis and I. Chatzigiannakis. "Competition: An Adaptive Protocol Stack for High-Dependability based on the Population Protocols Paradigm." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[6] Arago Systems. *WiSMote datasheet*. 2011.

[7] Á. Asensio, R. Blasco, Á. Marco, and R. Casas. "Hardware Architecture Design for WSN Runtime Extension." In: *International Journal of Distributed Sensor Networks* 9 (4 2013).

[8] A. Ayadi, D. Ros, and L. Toutain. *TCP header compression for 6LoWPAN*. Internet draft. 2011.

[9] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. "SensorScope: Out-of-the-Box Environmental Monitoring." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2008.

[10] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. "PermaDAQ: A Scientific Instrument for Precision Sensing and Data Recovery in Environmental Extremes." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2009.

[11] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. "MACAW: A Media Access Protocol for Wireless LAN's." In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM)*. 1994.

[12] M. Z. A. Bhuiyan, J. Wu, G. Wang, Z. Chen, J. Chen, and T. Wang. "Quality-Guaranteed Event-Sensitive Data Collection and Monitoring in Vibration Sensor Networks." In: *IEEE Transactions on Industrial Informatics* 13 (2 2017).

[13] E. S. Biagioni and K. W. Bridges. "The Application of Remote Sensor Technology To Assist the Recovery of Rare and Endangered Species." In: *International Journal of High Performance Computing Applications* 16 (3 2002).

[14] R. Bischoff, J. Meyer, and G. Feltrin. "Wireless Sensor Network Platforms." In: *Encyclopedia of Structural Health Monitoring*. 2009.

[15] C. A. Boano, J. Brown, Z. He, U. Roedig, and T. Voigt. "Low-Power Radio Communication in Industrial Outdoor Deployments: The Impact of Weather Conditions and ATEX-Compliance." In: *Proceedings of the International Conference on Sensor Networks Applications, Experimentation and Logistics (SENSAPPEAL)*. 2010.

[16] C. A. Boano, N. Tsiftes, T. Voigt, J. Brown, and U. Roedig. "The Impact of Temperature on Outdoor Industrial Sensornet Applications." In: *IEEE Transactions on Industrial Informatics* 6 (3 2010).

[17] C. A. Boano, H. Wennerström, M. Zúñiga, J. Brown, C. Keppitiyagama, F. J. Oppermann, U. Roedig, L.-Å. Nordén, T. Voigt, and K. Römer. "Hot Packets: A Systematic Evaluation of the Effect of Temperature on Low Power Wireless Transceivers." In: *Proceedings of the Extreme Conference on Communication and Computing (ExtremeCom)*. 2013.

[18] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer. "TempLab: A Testbed Infrastructure to Study the Impact of Temperature on Wireless Sensor Networks." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2014.

[19] C. Bormann, K. Hartke, and Z. Shelby. *The Constrained Application Protocol (CoAP)*. RFC 7252. 2015.

[20] M. Brachmann, O. Landsiedel, and S. Santini. "Concurrent Transmissions for Communication Protocols in the Internet of Things." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2016.

[21] M. Brachmann, O. Landsiedel, and S. Santini. "Keep the Beat: On-The-Fly Clock Offset Compensation for Synchronous Transmissions in Low-Power Networks." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2017.

[22] Martina Brachmann, Olaf Landsiedel, Diana Göhring, and Silvia Santini. *Whisper: Fast Flooding for Low-Power Wireless Networks*. 2018. eprint: `arXiv:XXXX.XXXXX`.

[23] A. Brandt, J.-P. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, T. H. Clausen, and T. Winter. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. 2015.

[24] M. Buettner, C. V. Yee, E. Anderson, and R. Han. "X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2006.

[25] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali. "Forwarder Selection in Multi-Transmitter Networks." In: *Proceedings of the Conference Distributed Computing in Sensor Systems (DCOSS)*. 2013.

[26] D. Carlson and A. Terzis. "Flip-MAC: A Density-Adaptive Contention-Reduction Protocol for Efficient Any-to-One Communication." In: *Proceedings of the Conference Distributed Computing in Sensor Systems (DCOSS)*. 2011.

[27] V. Cerf, R. Kahn, and J. Postel. *Transmission Control Protocol*. RFC 793. 1981.

[28] M. Ceriotti, M. Corrà, L. D'Orazio, R. Doriguzzi, D. Facchin, S. T. Gună, G. P. Jesi, R. L. Cigno, L. Mottola, A. L. Murphy, M. Pescalli, G. P. Picco, D. Pregnolato, and C. Torghele. "Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2011.

[29] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2009.

[30] D. Cheng, Y. Mao, Y. Wang, and X. Wang. "Improving Energy Adaptivity of Constructive Interference-Based Flooding for WSN-AF." In: *International Journal of Distributed Sensor Networks* 11 (6 2015).

[31] Cisco Systems, Inc. *Integrating an Industrial Wireless Sensor Network with Your Plant's Switched Ethernet and IP Network*. White Paper. 2000.

[32] International Electrotechnical Commission. *Industrial communication networks - Fieldbus specifications - WIA-PA communication network and communication profile*. IEC 62601. 2009.

[33] International Electrotechnical Commission. *Industrial communication networks - Fieldbus specifications - WirelessHART™communication network and communication profile*. IEC 62591. 2009.

[34] International Electrotechnical Commission. *Industrial communication networks - Fieldbus specifications - Wireless systems for industrial automation: process control and related applications*. IEC 62734. 2012.

[35] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. 2017.

[36] M. Doddavenkatappa and M. C. Chan. "P3: A Practical Packet Pipeline Using Synchronous Transmissions for Wireless Sensor Networks." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2014.

[37] M. Doddavenkatappa, M. C. Chan, and B. Leong. "Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks." In: *Proceedings of the Symposium on Networked Systems Design & Implementation (USENIX NSDI)*. 2013.

[38] W. Du, J. C. Liando, H. Zhang, and M. Li. "When Pipelines Meet Fountain: Fast Data Dissemination in Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2015.

[39] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert. "6TiSCH: Deterministic IP-enabled Industrial Internet (of Things)." In: *IEEE Communications Magazine* 52 (12 2014).

[40] A. Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. 2011.

[41] S. Duquennoy, F. Österlind, and A. Dunkels. "Lossy Links, Low Power, High Throughput." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2011.

[42] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. "A-MAC: A Versatile and Efficient Receiver-initiated Link Layer for Low-power Wireless." In: *ACM Transactions on Sensor Networks* 8 (4 2012).

[43] P. Dutta, R. Musăloiu-E., I. Stoica, and A. Terzis. "Wireless ACK Collisions Not Considered Harmful." In: *Proceedings of the Workshop on Hot Topics in Networks (ACM HotNets)*. 2008.

[44] J. Elson, L. Girod, and D. Estrin. "Fine-grained Network Time Synchronization Using Reference Broadcasts." In: *Proceedings of the Symposium on Operating Systems Design & Implementation (USENIX OSDI)*. 2002.

[45] A. Elsts, X. Fafoutis, A. Adeleke, R. J. Piechocki, G. C. Oikonomou, S. Duquennoy, A. Liñán, and M. Fàbregas. "Competition: Adaptive Time-Slotted Channel Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[46] A. T. Erman, L. v. Hoesel, P. Havinga, and J. Wu. "Enabling Mobility in Heterogeneous Wireless Sensor Networks Cooperating with UAVs for Mission-Critical Management." In: *IEEE Wireless Communications* 15 (6 2008).

[47] A. Escobar, F. J. Cruz, J. Garcia-Jimenez, J. Klaue, and A. Corona. "RedFixHop with Channel Hopping: Reliable Ultra-Low-Latency Network Flooding." In: *Proceedings of the Conference on Design of Circuits and Integrated Systems (DCIS)*. 2016.

[48] A. Escobar, J. Garcia, F. Cruz, J. Klaue, A. Corona, and D. Tati. "Competition: RedFixHop with Channel Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[49] A. Escobar, F. Moreno, B. Saez, A. J. Cabrera, J. Garcia-Jimenez, F. J. Cruz, U. Ruiz, A. Corona, J. Klaue, and D. Tati. "Competition: BigBangBus." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[50] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. "Low-power wireless bus." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2012.

[51] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. "Virtual Synchrony Guarantees for Cyber-physical Systems." In: *Proceedings of the International Symposium on Reliable Distributed Systems (IEEE SRDS)*. 2013.

[52] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. "Efficient network flooding and time synchronization with Glossy." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2011.

[53] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. 1999.

[54] M. Franceschinis, C. Pastrone, M. A. Spirito, and C. Borean. "On the performance of ZigBee Pro and ZigBee IP in IEEE 802.15.4 networks." In: *Proceedings of the International Conference on Wireless and Mobile Computing, Networking and Communications (IEEE WiMob)*. 2013.

[55] B. Galloway and G. P. Hancke. "Introduction to Industrial Control Networks." In: *IEEE Communications Surveys Tutorials* 15 (2 2013).

[56] S. Ganeriwal, R. Kumar, and M. B. Srivastava. "Timing-sync Protocol for Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2003.

[57] P. H. Gomes, T. Watteyne, P. Gosh, and B. Krishnamachari. "Competition: Reliability Through Timeslotted Channel Hopping and Flooding-based Routing." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[58] V. C. Gungor and G. P. Hancke. "Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches." In: *IEEE Transactions on Industrial Electronics* 56 (10 2009).

[59] Z. He, K. Hewage, and T. Voigt. "Arpeggio: A Penetration Attack on Glossy Networks." In: *Proceedings of the Conference on Sensor, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*. 2016.

[60] K. Hewage, S. Duquennoy, V. Iyer, and T. Voigt. "Enabling TCP in Mobile Cyber-Physical Systems." In: *Proceedings of the International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS)*. 2015.

[61] K. Hewage, S. Raza, and T. Voigt. "An Experimental Study of Attacks on the Availability of Glossy." In: *Computers & Electrical Engineering* 41 (2015 2015).

[62] K. Hewage, S. Raza, and T. Voigt. "Protecting Glossy-Based Wireless Networks from Packet Injection Attacks." In: *Proceedings of the International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS)*. 2017.

[63] L. Hou and N. W. Bergmann. "System Requirements for Industrial Wireless Sensor Networks." In: *Proceedings of the Conference on Emerging Technologies Factory Automation (IEEE ETFA)*. 2010.

[64] J. W. Hui and D. Culler. "IP is Dead, Long Live IP for Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2008.

[65] J. Hui and P. Thubert. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. RFC 6282. 2011.

[66] N. Q. V. Hung, H. Jeung, and K. Aberer. "An Evaluation of Model-Based Approaches to Sensor Data Compression." In: *IEEE Transactions on Knowledge and Data Engineering* 25 (11 2013).

[67] IEEE Computer Society. *IEEE Standard for Low-Rate Wireless Networks*. 2016.

[68] Silicon Laboratories Inc. *The Net Benefits of Single-Chip Integration for ZigBee SoC Solutions*. 2013.

[69] Texas Instruments. *CC2520 datasheet*. 2007.

[70] Texas Instruments. *MSP430F1611 datasheet*. 2011.

[71] Texas Instruments. *CC1101 datasheet*. 2013.

[72] Texas Instruments. *CC2420 datasheet*. 2013.

[73] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza. "Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2016.

[74] T. Istomin, M. Trobinger, A. L. Murphy, and G. P. Picco. "Interference-Resilient Ultra-Low Power Aperiodic Data Collection." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2018.

[75] J. Jeong, J. Park, H. Jeong, J. Jun, C. J. M. Liang, and J. Ko. "Low-Power and Topology-Free Data Transfer Protocol with Synchronous Packet Transmissions." In: *Proceedings of the Conference on Sensor, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*. 2014.

[76] H. Jiang, Z. Brodard, T. Chang, A. Bouabdallah, N. Montavont, G. Texier, P. Thubert, T. Watteyne, and G. Z. Papadopoulos. "Competition: Controlled Replication for Higher Reliability and Predictability in Industrial IoT Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[77] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. 2005.

[78] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle. "When HART Goes Wireless: Understanding and Implementing the WirelessHART Standard." In: *Proceedings of the Conference on Emerging Technologies Factory Automation (IEEE ETFA)*. 2008.

[79] N. Kimura and S. Latifi. "A Survey on Data Compression in Wireless Sensor Networks." In: *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC)*. 2005.

[80] A. King, J Hadley, and U. Roedig. "Competition: ContikiMAC with Differentiating Clear Channel Assessment." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[81] J. Klaue, A. Corona, M. Kubisch, J. Garcia-Jimenez, and A. Escobar. "Competition: RedFix-Hop with Channel Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[82] L. Kleinrock and F. Tobagi. "Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics." In: *IEEE Transactions on Communications* 23 (12 1975).

[83] M. König and R. Wattenhofer. "Effectively Capturing Attention Using the Capture Effect." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2016.

[84] M. König and R. Wattenhofer. "Maintaining Constructive Interference Using Well-Synchronized Sensor Nodes." In: *Proceedings of the Conference Distributed Computing in Sensor Systems (DCOSS)*. 2016.

[85] M. Kovatsch, S. Duquennoy, and A. Dunkels. "A Low-Power CoAP for Contiki." In: *Proceedings of the International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS)*. 2011.

[86] C. P. Kruger and G. P. Hancke. "Implementing the Internet of Things Vision in Industrial Wireless Sensor Networks." In: *Proceedings of the International Conference on Industrial Informatics (IEEE INDIN)*. 2014.

[87] J. Kumagai. "Life of birds [wireless sensor network for bird study]." In: *IEEE Spectrum* 41 (4 2004).

[88] B. Kusy, P. Dutta, P. Levis, M. Maroti, Á. Ledeczi, and D. Culler. "Elapsed Time on Arrival; a Simple and Versatile Primitive for Canonical Time Synchronisation Services." In: *International Journal of Ad Hoc and Ubiquitous Computing* 1 (4 2006).

[89] L. Lamont, M. Toulgoat, Mathieu Déziel, and G. Patterson. "Tiered Wireless Sensor Network Architecture for Military Surveillance Applications." In: *Proceedings of the Conference on Sensor Technologies and Applications (SENSORCOMM)*. 2011.

[90] O. Landsiedel, F. Ferrari, and M. Zimmerling. "Chaos: Versatile and Efficient All-to-All Data Sharing and In-Network Processing at Scale." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2013.

[91] S. H. Lee, S. Lee, H. Song, and H. S. Lee. "Wireless Sensor Network Design for Tactical Military Applications: Remote Large-Scale Environments." In: *Proceedings of the Military Communications Conference (IEEE MILCOM)*. 2009.

[92] C. Lenzen, P. Sommer, and R. Wattenhofer. "PulseSync: An Efficient and Scalable Clock Synchronization Protocol." In: *IEEE/ACM Transactions on Networking* 23 (3 2015).

[93] C. J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis. "Surviving Wi-fi Interference in Low Power ZigBee Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2010.

[94] W. Liang, X. Zhang, Y. Xiao, F. Wang, P. Zeng, and H. Yu. "Survey and Experiments of WIA-PA Specification of Industrial Wireless Network." In: *Wireless Communication and Mobile Computing* 11 (8 2011).

[95] C.-H. Liao, Y. Katsumata, M. Suzuki, and H. Morikawa. "Revisiting the So-Called Constructive Interference in Concurrent Transmission." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2016.

[96] C.-H. Liao, T. Sakdejayont, M. Suzuki, Y. Narusue, and H. Morikawa. "Competition: Wireless-Transparent Sensing Platform." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[97] C.-H. Liao, M. Suzuki, and H. Morikawa. "Receiver Performance Evaluation and Fading Duration Analysis for Concurrent Transmission." In: *IEICE Transactions on Communications* E101.B (2 2017).

[98] C.-H. Liao, G. Zhu, D. Kuwabara, M. Suzuki, and H. Morikawa. "Multi-Hop LoRa Networks Enabled by Concurrent Transmission." In: *IEEE Access* 5 (2017).

[99] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. "FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2013.

[100] T. H. Lim, I. Bate, and J. Timmis. "Competition: Multimodal Reactive-Routing Protocol to Tolerate Failure." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[101] X. Liu, J. Cao, S. Tang, and J. Wen. "Enabling Reliable and Network-Wide Wakeup in Wireless Sensor Networks." In: *IEEE Transactions on Wireless Communications* 15 (3 2016).

[102] Y. Liu, Qi Chen, Hao Liu, Chen Hu, and Qing Yang. "A Non Destructive Interference Based Receiver-Initiated MAC Protocol for Wireless Sensor Networks." In: *Proceedings of the Annual Consumer Communications Networking Conference (IEEE CCNC)*. 2016.

[103] Olimex Ltd. *MSP430-CCRF development board*. User's manual. 2013.

[104] J. Lu and K. Whitehouse. "Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks." In: *Proceedings of the Conference on Computer Communications (IEEE INFOCOM)*. 2009.

[105] X. Ma, W. Tang, W. He, F. Zhang, and J. Wei. "Competition: Using OF∂COIN under Interference." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[106] X. Ma, P. Zhang, W. Tang, X. Li, W.i He, F. Zhang, J. Wei, and O. Theel. "Competition: Using Enhanced OF∂COIN to Monitor Multiple Concurrent Events under Adverse Conditions." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[107] F. Mager, C. Herrmann, and M. Zimmerling. "One for All, All for One: Toward Efficient Many-to-Many Broadcast in Dynamic Wireless Networks." In: *Proceedings of the Workshop on Hot Topics in Wireless (ACM HotWireless)*. 2017.

[108] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. "Wireless Sensor Networks for Habitat Monitoring." In: *Proceedings of the International Workshop on Wireless Sensor Networks and Applications (ACM WSNA)*. 2002.

[109] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. "The Flooding Time Synchronization Protocol." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2004.

[110] A. Maskooki, V. Toldov, L. Clavier, V. Loscrí, and N. Mitton. "Competition: Channel Exploration/Exploitation Based on a Thompson Sampling Approach in a Radio Cognitive Environment." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[111] M. Mohammad and M. C. Chan. "Codecast: Supporting Data Driven In-network Processing for Low-power Wireless Sensor Networks." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2018.

[112] M. Mohammad, M. Doddavenkatappa, and M. C. Chan. "Improving Performance of Synchronous Transmission-Based Protocols Using Capture Effect over Multichannels." In: *ACM Transactions on Sensor Networks* 13 (2 2017).

[113] M. Mohammad, X.-F. Guo, and M. C. Chan. "Competition: Tackling Cross-technology Interference using Spatial and Channel Diversity for Robust Data Collection." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[114] D. Moss and P. Levis. *BoX-MACs: Exploiting physical and link layer boundaries in low-power networking*. 2008.

[115] Moteiv. *Tmote Sky datasheet*. 2006.

[116] J. R. Moyne and D. M. Tilbury. "The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data." In: *Proceedings of the IEEE* 95 (1 2007).

[117] R. Musăloiu., C. J. M. Liang, and A. Terzis. "Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2008.

[118] B. A. Nahas, S. Duquennoy, and O. Landsiedel. "Network-wide Consensus Utilizing the Capture Effect in Low-power Wireless Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2017.

[119] B. A. Nahas and O. Landsiedel. "Competition: Towards Low-Latency, Low-Power Wireless Networking under Interference." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[120] B. A. Nahas and O. Landsiedel. "Competition: Towards Low-Power Wireless Networking that Survives Interference with Minimal Latency." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[121] B. A. Nahas and O. Landsiedel. "Competition: Aggressive Synchronous Transmissions with In-network Processing for Dependable All-to-All Communication." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[122] C. Noda, C. M. Pérez-Penichet, B. Seeber, M. Zennaro, M. Alves, and A. Moreira. "On the Scalability of Constructive Interference in Low-Power Wireless Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2015.

[123] T. O'Donovan, J. Brown, F. Büsching, A. Cardoso, J. Cecílio, J. Do Ó, P. Furtado, P. Gil, A. Jugel, W.-B. Pöttner, U. Roedig, J. S. Silva, R. Silva, C. J. Sreenan, V. Vassiliou, T. Voigt, L. Wolf, and Z. Zinonos. "The GINSENG System for Wireless Monitoring and Control: Design and Deployment Experiences." In: *ACM Transactions on Sensor Networks* 10 (1 2013).

[124] J. Park, J. Jeong, H. Jeong, C. J. M. Liang, and J. Ko. "Improving the Packet Delivery Performance for Concurrent Packet Transmissions in WSNs." In: *IEEE Communications Letters* 18 (1 2014).

[125] F. De Pellegrini, D. Miorandi, S. Vitturi, and A. Zanella. "On the Use of Wireless Networks at Low Level of Factory Automation Systems." In: *IEEE Transactions on Industrial Informatics* 2 (2 2006).

[126] G. P. Picco, D. Molteni, A. L. Murphy, F. Ossi, F. Cagnacci, M. Corrà, and S. Nicoloso. "Geo-referenced Proximity Detection of Wildlife with WildScope: Design and Characterization." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2015.

[127] K. Pister and L. Doherty. "TSMP: Time Synchronized Mesh Protocol." In: *Proceedings of the International Symposium on Distributed Sensor Networks (DSN)*. 2008.

[128] J. Polastre, J. Hill, and D. Culler. "Versatile Low Power Media Access for Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2004.

[129] J. Postel. *User Datagram Protocol*. RFC 768. 1980.

[130] F. Sutton R. Lim R. D. Forno and L. Thiele. "Competition: Robust Flooding using Back-to-Back Synchronous Transmissions with Channel-Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[131] S. Nürnberger R., Karnapke J., and Nolte. "Sensorium – An Active Monitoring System for Neighborhood Relations in Wireless Sensor Networks." In: *Proceedings of the International Conferenceon Ad Hoc Networks (ADHOCNETS)*. 2010.

[132] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. "Sympathy for the Sensor Network Debugger." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2005.

[133] V. S. Rao, M. Koppal, R. V. Prasad, T. V. Prabhakar, C. Sarkar, and I. Niemegeers. "Murphy loves CI: Unfolding and Improving Constructive Interference in WSNs." In: *Proceedings of the Conference on Computer Communications (IEEE INFOCOM)*. 2016.

[134] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. "Practical Data Prediction for Real-World Wireless Sensor Networks." In: *IEEE Transactions on Knowledge and Data Engineering* 27 (8 2015).

[135] U. Raza, Y. Jin, and M. Sooriyabandara. "Competition: Synchronous Transmissions based Flooding for Dependable Internet of Things." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[136] U. Raza, Y. Jin, A. Stanoev, M. Baddeley, and M. Sooryiabandara. "Competition: CROWN – Concurrent ReceptiOns in Wireless Sensor and Actuator Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[137] C. Rojas and J.-D. Decotignie. "Competition: Synchronous Transmissions + Channel Sampling = Energy Efficient Event-Triggered Wireless Sensing Systems." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[138] K. Römer and F. Mattern. "The Design Space of Wireless Sensor Networks." In: *IEEE Wireless Communications* 11 (6 2004).

[139] A. A. Kumar S., K. Ovsthus, and L. M. Kristensen. "An Industrial Perspective on Wireless Sensor Networks – A Survey of Requirements, Protocols, and Challenges." In: *IEEE Communications Surveys Tutorials* 16 (3 2014).

[140] S. Saha and M. C. Chan. "Design and Application of a Many-to-One Communication Protocol." In: *Proceedings of the Conference on Computer Communications (IEEE INFOCOM)*. 2017.

[141] S. Saha, O. Landsiedel, and M. C. Chan. "Efficient Many-to-Many Data Sharing Using Synchronous Transmission and TDMA." In: *Proceedings of the Conference Distributed Computing in Sensor Systems (DCOSS)*. 2017.

[142] J. Sallai, B. Kusý, Á. Lédeczi, and P. Dutta. "On the Scalability of Routing Integrated Time Synchronization." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2006.

[143] C. Sarkar. *LWB and FS-LWB implementation for Sky platform using Contiki*. 2016.

[144] C. Sarkar, R. V. Prasad, R. T. Rajan, and K. Langendoen. "Sleeping Beauty: Efficient Communication for Node Scheduling." In: *Proceedings of the International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS)*. 2016.

[145] G. Scheible, D. Dzung, J. Endresen, and J. E. Frey. "Unplugged But Connected [Design and Implementation of a Truly Wireless Real-Time Sensor/Actuator Interface]." In: *IEEE Industrial Electronics Magazine* 1 (2 2007).

[146] T. Schmid, P. Dutta, and M. B. Srivastava. "High-resolution, Low-power Time Synchronization an Oxymoron No More." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2010.

[147] M. Schuß, C. A. Boano, M. Weber, and K. Römer. "A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[148] C. Sergiou, V. Vassiliou, C. Georgiou, C. Ioannou, N. Temene, and A. Paphitis. "Competition: Dynamic Alternative Path Selection in Wireless Sensor Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[149] Z. Shelby and C. Bormann. *6LoWPAN: The Wireless Embedded Internet*. 2009.

[150] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. "Sensor Network-based Countersniper System." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2004.

[151] K. Sohraby, D. Minoli, and T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*. 2007.

[152] P. Sommer, Y. A. Pignolet, S. Marinkovic, A. Monot, M. Kabir-Querrec, and R. Birke. "Competition: Energy-Efficient Many-to-Many Communication with Channel-Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[153] P. Sommer and Y.-A. Pignolet. "Competition: Dependable Network Flooding Using Glossy with Channel-Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[154] P. Sommer and Y.-A. Pignolet. "Competition: Energy-Efficient Network Flooding with Channel-Hopping." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[155] D. Son, J. Heidemann, and B. Krishnamachari. *Towards Concurrent Communication in Wireless Networks*. 2007.

[156] D. Son, B. Krishnamachari, and J. Heidemann. "Experimental Study of Concurrent Transmission in Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2006.

[157] D. Son, B. Krishnamachari, and J. Heidemann. *Evaluating the Importance of Concurrent Packet Communication in Wireless Networks*. 2007.

[158] D. Son, B. Krishnamachari, and J. Heidemann. *Wireless Medium Access for Concurrent Communication*. 2008.

[159] D. Spenza, M. Magno, S. Basagni, L. Benini, M. Paoli, and C. Petrioli. "Beyond Duty Cycling: Wake-up Radio with Selective Awakenings for Long-lived Wireless Sensing Systems." In: *Proceedings of the Conference on Computer Communications (IEEE INFOCOM)*. 2015.

[160] T. Stathopoulos, R. Kapur, D. Estrin, J. Heidemann, and L. Zhang. "Application-Based Collision Avoidance in Wireless Sensor Networks." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2004.

[161] F. Sutton, B. Buchli, J. Beutel, and L. Thiele. "Zippy: On-Demand Network Flooding." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2015.

[162] F. Sutton, R. Da Forno, J. Beutel, and L. Thiele. "BLITZ: A Network Architecture for Low Latency and Energy-efficient Event-triggered Wireless Communication." In: *Proceedings of the Workshop on Hot Topics in Wireless (ACM HotWireless)*. 2017.

[163] F. Sutton, R. Da Forno, D. Gschwend, T. Gsell, R. Lim, J. Beutel, and L. Thiele. "The Design of a Responsive and Energy-efficient Event-triggered Wireless Sensing System." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2017.

[164] M. Suzuki, C.-H. Liao, Y. Katsumata, K. Jinno, and H. Morikawa. "Competition: Is Concurrent Transmission Flooding a Good Idea for Random Traffic?" In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[165] M. Suzuki, Y. Yamashita, and H. Morikawa. "Low-Power, End-to-End Reliable Collection Using Glossy for Wireless Sensor Networks." In: *Proceedings of the Vehicular Technology Conference (IEEE VTC Spring)*. 2013.

[166] S. Svensson. *Challenges of Wireless Communication in Industrial Systems*. Keynote talk at ETFA 2011.

[167] F. Talucci and M. Gerla. "MACA-BI (MACA By Invitation). A Wireless MAC Protocol for High Speed Ad Hoc Networking." In: *Proceedings of the International Conference on Universal Personal Communications (ICUPC)*. 1997.

[168] J. Thelen, D. Goense, and K. Langendoen. "Radio Wave Propagation in Potato Fields." In: *Proceedings of the International Workshop on Wireless Network Measurements (WiNMee)*. 2005.

[169] J. P. Thomesse. "Fieldbus Technology in Industrial Automation." In: *Proceedings of the IEEE* 93 (6 2005).

[170] P. Thubert, T. Watteyne, M. R. Palattella, X. Vilajosana, and Q. Wang. "IETF 6TSCH: Combining IPv6 Connectivity with Industrial Performance." In: *Proceedings of the International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. 2013.

[171] M. Trobinger, T. Istomin, A. L. Murphy, and G. P. Picco. "Competition: CRYSTAL Clear: Making Interference Transparent." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[172] J.-P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. 2010.

[173] J. Wang, H. Tall, and G. Chalhoub. "Competition: Smart Flooding with Multichannel for Industrial Wireless Sensor Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2018.

[174] P. Wang and W. Zhuang. "An Improved Busy-Tone Solution for Collision Avoidance in Wireless Ad Hoc Networks." In: *Proceedings of the International Conference on Communications (IEEE ICC)*. 2006.

[175] Y. Wang, Y. He, D. Cheng, Y. Liu, and X. Li. "TriggerCast: Enabling Wireless Collisions Constructive." In: *Proceedings of the Conference on Computer Communications (IEEE INFOCOM)*. 2013.

[176] Y. Wang, Y. He, X. Mao, Y. Liu, and X.-y. Li. "Exploiting Constructive Interference for Scalable Flooding in Wireless Networks." In: *IEEE/ACM Transactions on Networking* 21 (6 2013).

[177] Y. Wang, Y. Liu, Y. He, X. Y. Li, and D. Cheng. "Disco: Improving Packet Delivery via Deliberate Synchronized Constructive Interference." In: *IEEE Transactions on Parallel and Distributed Systems* 26 (3 2015).

[178] H. Wennerström, F. Hermans, O. Rensfelt, C. Rohner, and L.-Å. Nordén. "A Long-Term Study of Correlations between Meteorological Conditions and 802.15.4 Link Performance." In: *Proceedings of the Conference on Sensor, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*. 2013.

[179] J. Werb. *Adoption of Wireless for Safety*. Presentation at the Process Control and Safety Symposium (PCS). 2015.

[180] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. "Exploiting the Capture Effect for Collision Detection and Recovery." In: *Proceedings of the Workshop on Embedded Networked Sensor Systems (IEEE Emnets)*. 2005.

[181] M. Wilhelm, V. Lenders, and J. B. Schmitt. "On the Reception of Concurrent Transmissions in Wireless Sensor Networks." In: *IEEE Transactions on Wireless Communications* Volume (2014).

[182] A. Willig. "An Architecture for Wireless PROFIBUS." In: *Proceedings of the Conference of the IEEE Industrial Electronics Society (IEEE IECON)*. 2003.

[183] A. Willig. "Recent and Emerging Topics in Wireless Industrial Communications: A Selection." In: *IEEE Transactions on Industrial Informatics* 4 (2 2008).

[184] A. Willig, K. Matheus, and A. Wolisz. "Wireless Technology in Industrial Networks." In: *Proceedings of the IEEE* 93 (6 2005).

[185] D. Wu, C. Dong, S. Tang, H. Dai, and G. Chen. "Fast and Fine-Grained Counting and Identification via Constructive Interference in WSNs." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2014.

[186] W. Ye, F. Silva, and J. Heidemann. "Ultra-Low Duty Cycle MAC with Scheduled Channel Polling." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2006.

[187] P. Yu, L. Qinghua, and P. Xiyuan. "The Design of Low-Power Wireless Sensor Node." In: *Proceedings of the International Instrumentation and Measurement Technology Conference (IEEE I2MTC)*. 2010.

[188] D. Yuan and M. Hollick. "Let's Talk Together: Understanding Concurrent Transmission in Wireless Sensor Networks." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2013.

[189] D. Yuan and M. Hollick. "Ripple: High-throughput, Reliable and Energy-efficient Network Flooding in Wireless Sensor Networks." In: *Proceedings of the Symposium on a World of Wireless Mobile and Multimedia Networks (IEEE WoWMoM)*. 2015.

[190] D. Yuan and M. Hollick. "Competition: Sparkle: Energy Efficient, Reliable, Ultra-low Latency Communication in Wireless Control Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2016.

[191] D. Yuan, M. Riecker, and M. Hollick. "Making 'Glossy' Networks Sparkle: Exploiting Concurrent Transmissions for Energy Efficient, Reliable, Ultra-Low Latency Communication in Wireless Control Networks." In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*. 2014.

[192] P. Zand, S. Chatterjea, K. Das, and P. Havinga. "Wireless Industrial Monitoring and Control Networks: The Journey So Far and the Road Ahead." In: *Journal of Sensor and Actuator Networks* 1 (2 2012).

[193] J. Zhang, A. Reinhardt, W. Hu, and S. S. Kanhere. "RFT: Identifying Suitable Neighbors for Concurrent Transmissions in Point-to-Point Communications." In: *Proceedings of the Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (ACM MSWiM)*. 2015.

[194] P. Zhang, Y. Gao, and O. Theel. "Less is More: Learning More with Concurrent Transmissions for Energy-Efficient Flooding." In: *Proceedings of the International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (ACM MobiQuitous)*. 2017.

[195] G. Zhao. "Wireless Sensor Networks for Industrial Process Monitoring and Control: A Survey." In: *Network Protocols and Algorithms* 3 (2 2011).

[196] M. Zimmerling, F. Ferrari, L. Mottola, and L. Thiele. "On Modeling Low-Power Wireless Protocols Based on Synchronous Packet Transmissions." In: *Proceedings of the International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS)*. 2013.

[197] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. "pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2012.

All websites and online documents were last accessed in January 2019.

AUTHOR'S PUBLICATIONS

[1] Martina Brachmann, Olaf Landsiedel, Diana Göhring, and Silvia Santini. *Whisper: Fast Flooding for Low-Power Wireless Networks*. In preparation for ACM Transactions on Sensor Networks. 2018. eprint: `arXiv:1809.03699`.

[2] M. Brachmann, O. Landsiedel, and S. Santini. "Keep the Beat: On-The-Fly Clock Offset Compensation for Synchronous Transmissions in Low-Power Networks." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2017.

[3] M. Brachmann, O. Landsiedel, and S. Santini. "Concurrent Transmissions for Communication Protocols in the Internet of Things." In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2016.

[4] M. Stein, T. Petry, I. Schweizer, M.Brachmann, and M. Mühlhäuser. "Topology Control in Wireless Sensor Networks: What Blocks the Breakthrough?" In: *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*. 2016.

[5] M. Brachmann, D. Becker, and S. Santini. "Poster Abstract: Towards Enabling Concurrent Transmissions in Heterogeneous Networks." In: *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. 2015.

[6] P. M. Scholl, M. Brachmann, S. Santini, and K. v. Laerhoven. "Integrating Wireless Sensor Nodes in the Robot Operating System." In: *Cooperative Robots and Sensor Networks*. 2014.

[7] S. L. Keoh, O. Garcia Morchon, S. S. Kumar, M. Brachmann, and B. Erdmann. *Methods, Devices and Systems for Establishing End-To-End Secure Connections and for Securely Communicating Data Packets*. Patent No: US20140143855A1. 2014.

[8] M. Brachmann and S. Santini. "Poster Abstract: Towards the Benchmarking of Ultra-Low Latency Communication Protocols for Wireless Sensor and Actuator Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2013.

[9] I. Gurov, P. E. Guerrero, M. Brachmann, S. Santini, K. v. Laerhoven, and A. Buchmann. "Poster Abstract: A Site Properties Assessment Framework for Wireless Sensor Networks." In: *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*. 2013.

[10] M. Brachmann, O. Garcia Morchon, S. Keoh, and S. Kumar. "End-to-End Transport Security in the IP-based Internet of Things." In: *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*. 2012.

[11] M. Brachmann, O. Garcia-Morchon, S. Keoh, and S. Kumar. "Security Considerations around End-to-End Security in the IP-based Internet of Things." In: *Workshop on Smart Object Security*. 2012.

[12] M. Brachmann, O. Garcia-Morchon, and M. Kirsche. "Security for Practical CoAP Applications: Issues and Solution Approaches." In: *Proceedings of the GI/ITG KuVS Fachgespraech Sensornetze (FGSN)*. 2011.

CURRICULUM VITAE

PERSONAL INFORMATION

Name	Martina Brachmann
Date of birth	April 14, 1987
Place of birth	Dresden, Germany
Nationality	German

EDUCATION

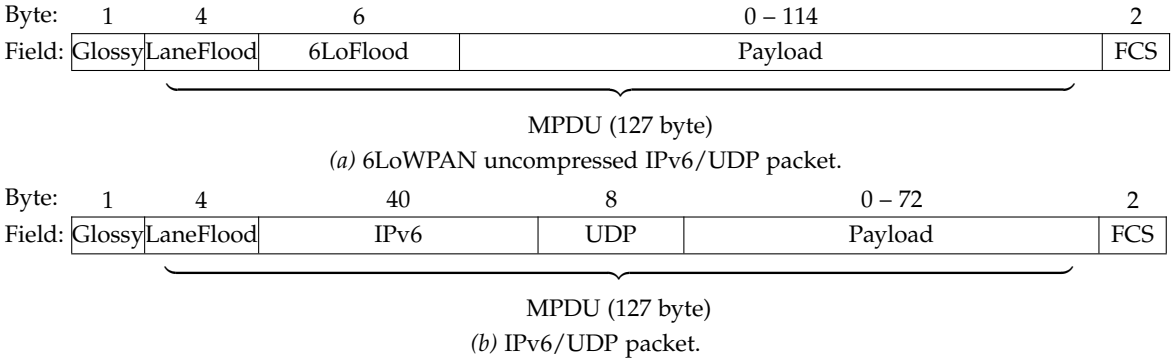
02/2015 – 11/2018	Technische Universität Dresden, Germany Doctoral candidate at the Department of Computer Science
05/2015 – 07/2015	Chalmers University of Technology, Gothenborg, Sweden Visiting scholar at the Computer Science and Engineering Group
12/2012 – 01/2015	Technische Universität Darmstadt, Germany Doctoral candidate at the Department of Electrical Engineering and Information Technology
03/2010 – 11/2012	Brandenburg University of Technology Cottbus, Germany Master studies in Information and Media Technology with major in Communication Technologies and Networks Degree: Master of Science
10/2006 – 01/2010	Brandenburg University of Technology Cottbus, Germany Bachelor studies in Information and Media Technology Degree: Bachelor of Science
08/2003 – 07/2006	Berufsschulzentrum für Elektrotechnik Dresden, Germany Gymnasium with major in Computer Science and German Degree: Abitur
08/1997 – 07/2003	86. Mittelschule Dresden, Germany Degree: Realschulabschluss
04/1994 – 07/1997	83. Grundschule Dresden, Germany
08/1993 – 04/1994	8. Grundschule Dresden, Germany

APPENDIX

A.1 6LoFlood

LaneFlood facilitates standard Internet protocols to efficiently run in a low-power wireless network. As mentioned before, Internet protocols like HTTP/CoAP, UDP/TCP and IPv6 enable the interoperability and interconnectivity of different networks. In this section, we describe the concept of 6LoFlood, which efficiently enables one-to-one communication between a host system on the Internet and nodes running LaneFlood within a low-power wireless network, as shown in Figure 5.1.

In the figure, a border router acts as access gate and thus, interconnects both networks, the Internet and the low-power wireless network. When a host on the Internet transmits a message to a node within the low-power wireless network, the border router forwards it, after including the Glossy and LaneFlood header, to the low-power wireless network. The resulting packet on the example of IPv6/UDP is shown in Figure A.1b. As indicated in the figure, more than 40 % of the available 127 byte in the IEEE 802.15.4 packet is used for headers and a footer. Further, a typical message size of a CoAP request can have several tens of byte [85], which may exceed the size of IEEE 802.15.4 packets. Using 6LoFlood, the border router compresses the protocol headers and fragments the message if necessary before propagating the packet into the low-power wireless network. Figure A.1a shows the resulting 6LoFlood packet. Protocol headers and footers only require 10 % of the available



(a) 6LoWPAN uncompressed IPv6/UDP packet.

(b) IPv6/UDP packet.

Figure A.1: IPv6/UDP packet.

127 byte. In case a node sends a message to a host on the Internet, the border router collects the fragments and forwards the unfragmented and de-compressed message to the host.

Typically, 6LoWPAN [65] is used for header compression and encapsulation as well as address resolution, and packet fragmentation. Further, it provides mechanisms for neighbor discovery and routing. This is all necessary to enable the interconnection of a low-power wireless network with the Internet. However, LaneFlood is a routing-free protocol, and thus, as mentioned before, nodes in the low-power wireless network do not require a full-fledged implementation of IPv6 and 6LoWPAN.

6LoFlood as adaption layer for the low-power wireless network relies on three cornerstones: node addressing, packet fragmentation, and header compression. All this information is encoded in the 6LoFlood header, shown in Figure A.2. In particular, the first field in the 6LoFlood header is the *fragmentation* field, which we discuss in Section A.1.2. Before, we explain in Section A.1.1 how nodes in a low-power wireless network derive their IPv6 addresses. In the same section, we further detail the message forwarding using the *Source Address Encoding (SAE)* and *Destination Address Encoding (DAE)* fields, shown in Figure A.2. These two and the following fields – i. e., the *Source Port Encoding (SPE)*, *Destination Port Encoding (DPE)*, the *global IPv6 address*, and the *compressed transport protocol fields* – are part of the outcome of the 6LoFlood header compression, which we discuss in Section A.1.3. We conclude this section in Section A.1.4.

A.1.1 *Node addressing and message forwarding*

IPv6 distinguishes between link-local and global addresses. Link-local addresses are used within a network, while global addresses are used to approach devices in different networks. To create a link-local IPv6 address, the nodes extend the link-local prefix `fe80::` with zeros to a 64-bit network prefix and append a unique 64-bit identifier, e. g., the hardware

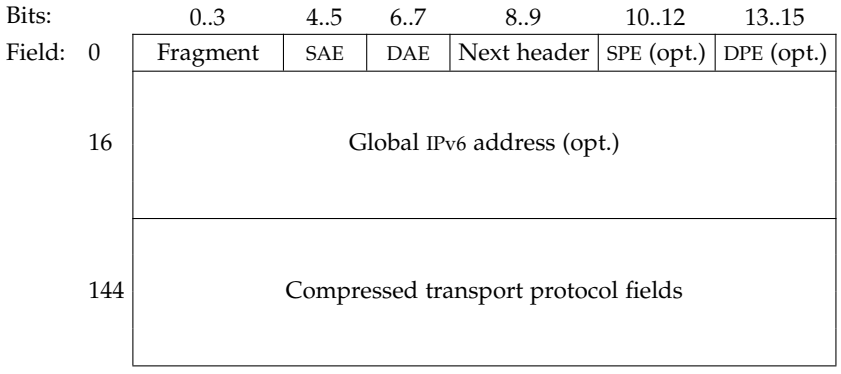


Figure A.2: 6LoFlood packet format.

address assigned by the manufacturer. Nodes running LaneFlood have a fixed 8-bit node identifier that is unique within the low-power wireless network. Thus, the nodes use this identifier to create their link-local IPv6 address. For example, a node with identifier `1` has the link-local IPv6 address `fe80::1`. For assigning a global IPv6 addresses, we use StateLess Automatic Address Configuration (SLAAC). Using SLAAC, routers periodically broadcast Router Advertisement (RA) messages, i. e., Internet Control Message Protocol (ICMP) messages from type 134 that contain the global network prefix. In addition, devices can request such RA messages directly via a Router Solicitation (RS) message, i. e., ICMP messages of type 133. By knowing the global prefix, devices create their global IPv6 address the same way they created their link-local IPv6 address.

The border router is connected to at least two networks. In our example, it is connected to the low-power wireless network and the Internet. It, therefore, has two network interfaces, each with its own link-local and global IPv6 address. In the following, we denote with *if-Int* the interface that connects the border router with the Internet and *if-LPWN* the interface for the low-power wireless network. Depending on the interface at which a message arrives, the border router can distinguish if a message is sent from or destined to the low-power wireless network. The message is then processed accordingly. In particular, the border router compresses the IPv6 addresses depending on whether messages are exchanged within the low-power wireless network, i. e., locally, or between an Internet host and a node in the low-power wireless network, i. e., globally. The corresponding address encoding is shown in Table A.1. In the following, we explain this address encoding in more detail.

A message sent by an Internet host to a node in the low-power wireless network always passes the border router. In particular, when the packet arrives at *if-Int*, the border router compresses the headers as described in Section A.1.3 and forwards the message to the low-power wireless network from *if-LPWN*. Because the Internet host and the node are located in two different networks – the Internet and the low-power wireless network – the source and destination IPv6 addresses are global. Therefore, the border router sets the *SAE* field in the 6LoFlood header to `11`, according to Table A.1. The host's global IPv6 address is included in the corresponding field in the 6LoFlood header, as shown in Figure A.2. Further, the *DAE* field is set to `10`. A destination address field is not required as the nodes' identifier is provided in the LaneFlood header, and the global IPv6 prefix is known by border router and

Table A.1: 6LoFlood address encoding of IPv6 addresses. The encoding is used to compress IPv6 addresses, and the values are transmitted in the *SAE* and *DAE* fields in the 6LoFlood header.

00	Reserved for future use
01	Link-local IPv6 address
10	Global IPv6 address of a node within the low-power wireless network
11	Global address of a device located on the Internet

node. However, the border router inserts his identifier in the source field of the LaneFlood header to enable LaneFlood to create a lane between itself and the destination node.

A node sending a message to an Internet host adds its identifier in the source field of the LaneFlood header and sets the *SAE* field to `10`. With this information, the border router is later able to create the node's global IPv6 address. The node further includes the border router's identifier in LaneFlood's destination field. This is required to create a lane between border router and the node. It further sets the *DAE* field to `11` and includes the host's global IPv6 address in the following field.

When the border router receives a packet at *if-LPWN*, where both the source and the destination addresses are link-local addresses, it behaves like a regular node in the low-power wireless network and does not perform header (de)-compression. The *SAE* and *DAE* fields are set to `01`, and the global IPv6 address field in the 6LoFlood header is omitted. Because the low-power wireless network is not a transit network, i. e., it cannot be used to reach another network, the border router does not forward messages through *if-LPWN* if the global IPv6 prefex does not match.

A.1.2 *Packet fragmentation*

IPv6 allows the transmission of packets with a Maximum Transfer Unit (MTU) of 1280 byte, while the maximum packet size in IEEE 802.15.4 is 127 byte. Therefore, 6LoFlood supports the fragmentation of packets. When a host on the Internet transmits a message that does not fit within an IEEE 802.15.4 packet, the border router fragments it and sends the fragments to the destination node. After receiving all fragments, the destination node recomposes the fragments to a message. The fragmentation process of 6LoFlood is rather simple. Knowing the (compressed) header sizes of the protocols to transmit, it tries to fit the message into an IEEE 802.15.4 packet. If it is successful, it transmits the packet, otherwise it transmits the message part that fits within the packet as fragment. Thus, all fragments except the last one have the maximum packet size.

Fragmentation information is encoded in the first four bits in a 6LoFlood packet, the *fragment* field, shown in Figure A.2. Table A.2 summarizes the encoding of the fragment field. In particular, the first two bits include information whether the packet is an unfragmented packet, the first or last fragmented packet, or a fragment in between. The second two bits

Table A.2: 6LoFlood Fragmentation information.

00	00	Unfragmented packet
11	00	First fragment
11	nn	Last fragment with sequence number Xnn in bits
01	nn	Fragment with sequence number 0nn
10	nn	Fragment with sequence number 1nn

carry the sequence numbers of the fragments. Each fragment gets a sequence number that is consecutively assigned before transmitting a fragment. When the first two bits of the fragment field are `01`, this value is substituted with `0` to derive the sequence number and when the first two bits are `10`, this value is substituted with `1`. For example, the fragment field of the first fragment is set to `11 00`, the second fragment is `01 00` ($000_2 + 1_{10} = 1$), the third fragment is `01 01` ($001_2 + 1_{10} = 2$), the sixth fragment is `10 00` ($100_2 + 1_{10} = 5$) and so on. This encoding allows us to split a message into at most 10 fragments, with the last fragment being encoded as `11 11`. If the Internet host tries to transmit a message larger than $10 \cdot 127\,\text{byte} = 1270\,\text{byte}$, the border router responds with a corresponding ICMP message. We will leave the transmission of such messages for future work.

LaneFlood natively facilitates an ordered message exchange between devices. The fragment sequence numbers are, thus, used to detect packet losses. Since the sequence numbers are consecutive, the receiver can compute which sequence number to expect next. Only the first and the last fragment requires a closer consideration. Let us assume the second last packet has the fragment field `10 01` ($101_2 + 1_{10} = 6$). Thus, the fragment field of the last packet is `11 01`. The receiver therefore checks if the last two bits of the last and second last packet in the fragment field match. The same applies for the first (`11 00`) and the second (`01 00`) fragment. If packet loss is detected, the current implementation of 6LoFlood discards all received fragments and thus relies on the retransmission of the message at the application layer.

When creating a lane, LaneFlood follows a request-response-scheme. More precisely, the source node disseminates a *Setup flood* and expects a *Response flood* in return. However, if the border router has to fragment the *Setup flood*, 6LoFlood does not process the packet at the destination until it received all fragments. Thus, the source waits for the *Response flood* while the destination waits for the remaining fragments. To resolve this issue, 6LoFlood enqueues an empty packet at the destination, consisting only of a Glossy and LaneFlood header and a footer to let LaneFlood transmit the *Response flood*. Afterwards, the source node continues transmitting the remaining fragments during LaneFlood's regular data exchange.

A.1.3 *Header compression*

Using LaneFlood, nodes that have pending data traffic create a lane at the beginning of a session, independently, i. e., without information, of the previous session. Therefore, 6LoFlood uses *stateless header compression* that only exploits redundancy in the packet without the need to keep state information. In the following, we first detail how 6LoFlood compresses an IPv6 header in Section A.1.3.1, and afterwards, in Section A.1.3.2, we discuss the header compression of UDP, TCP and ICMP. Lastly, we discuss the resulting packet sizes for different scenarios in Section A.1.3.3.

A.1.3.1 *IPv6 header compression*

Figure A.3 shows the IPv6 header structure. In the following, we discuss each field and detail how 6LoFlood compresses the particular field. We further compare our compression method with 6LoWPAN's IPv6 header compression, called HC1.

- *Version* The version field distinguishes IPv4 and IPv6 packets. Similar to 6LoWPAN, 6LoFlood only accepts IPv6 and therefore, 6LoWPAN and 6LoFlood omit this field.
- *Traffic class* and *flow label* These fields are used to provide Quality of Service (QoS), e. g., to prioritize packets or to report impending congestions. These two fields are rarely used in Internet traffic and are thus, set to zero [149]. 6LoWPAN allows the uncompressed transmission of these fields in cases they are different from zero. However, the current version of LaneFlood implements a first-in, first-out queue and therefore, does not handle packet prioritization or any other type of QoS. Thus, this field is omitted in 6LoFlood. However, in case a host on the Internet has set these fields to a value different from zero, the border router drops this packet. We leave the implementation of a notification message via ICMP for future work.
- *Payload length* The payload length can be derived from the IEEE 802.15.4 packet length, i. e., the first byte after the synchronization header, as described in Section 2.3.1, and the lengths of the headers in the packet, which are also known. Thus, similar to 6LoWPAN, this field is omitted by 6LoFlood.
- *Next header* This field specifies the next header after IPv6. LaneFlood currently only supports UDP, TCP, and ICMP and thus, compresses this field to 2 bit. The encoding is shown in Table A.3. 6LoWPAN also compresses this field to two bits. However, it transmits the next header in-line when both bits are zero. In 6LoFlood, the border router drops the packet instead. We leave the implementation of a notification message via ICMP for future work.
- *Hop limit* This field limits the number of routers a packet can pass and thus, prevents packet life-locks, i. e., packets that are trapped in a routing loop. In 6LoWPAN *"the*

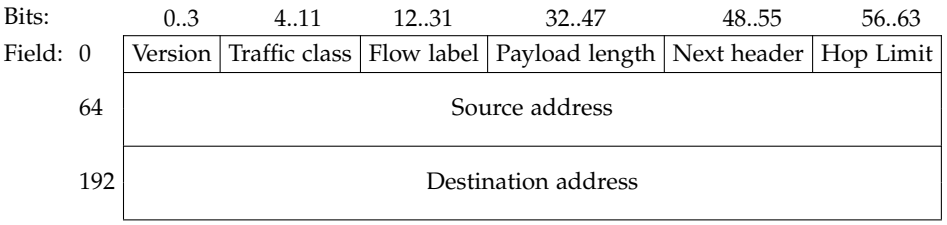


Figure A.3: Structure of an IPv6 header.

Hop Limit was considered too difficult to compress and therefore is always sent in-line in the non-compressed fields" [149]. However, LaneFlood is a routing-free protocol, and therefore, it omits this field.

- *Source* and *destination address* As already discussed in Section A.1.1, 6LoFlood uses LaneFlood's source and destination field for addressing nodes within the low-power wireless network. When exchanging data with an Internet host, it further includes the global IPv6 address of the host in the corresponding field of the 6LoFlood header shown in Figure A.2. The addressing is selected using the address encoding shown in Table A.1 and transmitted in the *SAE* and *DAE* fields. 6LoWPAN uses a similar approach and also derives the node identifiers from layer 2, which is the MAC layer.

A.1.3.2 *UDP and TCP header compression*

As mentioned in the previous section, 6LoFlood supports TCP, UDP and ICMP. In particular, it compresses the protocol headers of TCP and UDP. This is a difference to 6LoWPAN because first, 6LoWPAN compresses only UDP headers with its HC2 called compression method, and second, the use of HC2 is optional while 6LoFlood always compresses the mentioned transport protocols. In the following, we shortly describe the three protocols, detail their protocol headers and show how 6LoFlood compresses the header fields.

ICMP IPv6 uses ICMP Version 6 (ICMPv6) for diagnostic functions such as `ping6`, or SLAAC as discussed in Section A.1.1, as well as error reporting, e. g., when the destination is unreachable. The structure of an ICMP message is shown in Figure A.4. The field *type* identifies the control message, and the *code* field provides additional information. The *message body* contains the data specific for the type. The ICMP message further contains a *checksum* to provide integrity. Due to the already minimal structure of the ICMP message, 6LoFlood transmits the full message without compression. However, the *next header* field in the 6LoFlood header is set to `10`, and the *SPE* and *DPE* fields are omitted.

UDP The User Datagram Protocol (UDP) protocol is often preferred in low-power wireless networks because it has less overhead with respect to message size and the amount of exchanged messages compared to its counterpart TCP. Figure A.5b shows the structure of a

Table A.3: Next header encoding of 6LoFlood.

00	Reserved for future use
01	UDP
10	ICMP
11	TCP

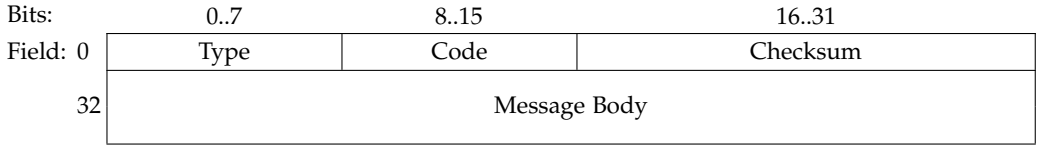
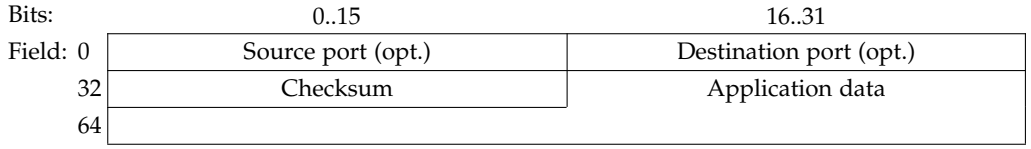
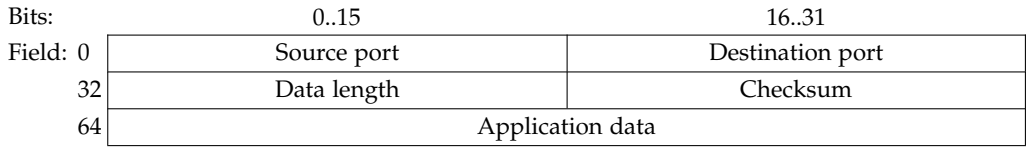


Figure A.4: ICMP message structure.



(a) 6LoFlood-UDP message structure.



(b) UDP message structure.

Figure A.5: Compressed 6LoFlood-UDP header vs. "standard" UDP header.

UDP message, and Figure A.5a shows the compressed 6LoFlood-UDP message for comparison. In the following, we describe the protocol fields and our compression method.

- *Source* and *destination port* Port numbers are used to bind to a specific application. 6LoWPAN uses in total 10 bits – 5 bit per source/destination – for the port compression. More precisely, it uses one bit to indicate whether the port is transmitted compressed or uncompressed. The compressed port number is transmitted in the remaining 4 bit as so-called *short_value*. The actual port number can be later computed with `port_number = short_value + 0xF0B0` [172].

  6LoFlood uses a different approach for compressing the port numbers. It simply uses fixed port numbers that are encoded as shown in Table A.4. These encoded port numbers are carried in the 6LoFlood header's *SPE* and *DPE* fields. In case one or both fields are set to `000`, the port numbers are sent in the source and destination port fields. Otherwise, these fields are omitted.

- *Data length* The data length can be derived from the packet length, i. e., the first byte in the IEEE 802.15.4 packet as described in Section 2.3.1, and the protocol headers. Thus, 6LoFlood omits this field. In contrast, 6LoWPAN indicates with a cleared bit that the data length field is omitted, otherwise it is transmitted uncompressed.

- *Checksum* To ensure data integrity, 6LoFlood always transmits the full UDP checksum. The receiving nodes re-model the UDP header based on the 6LoFlood header to compute

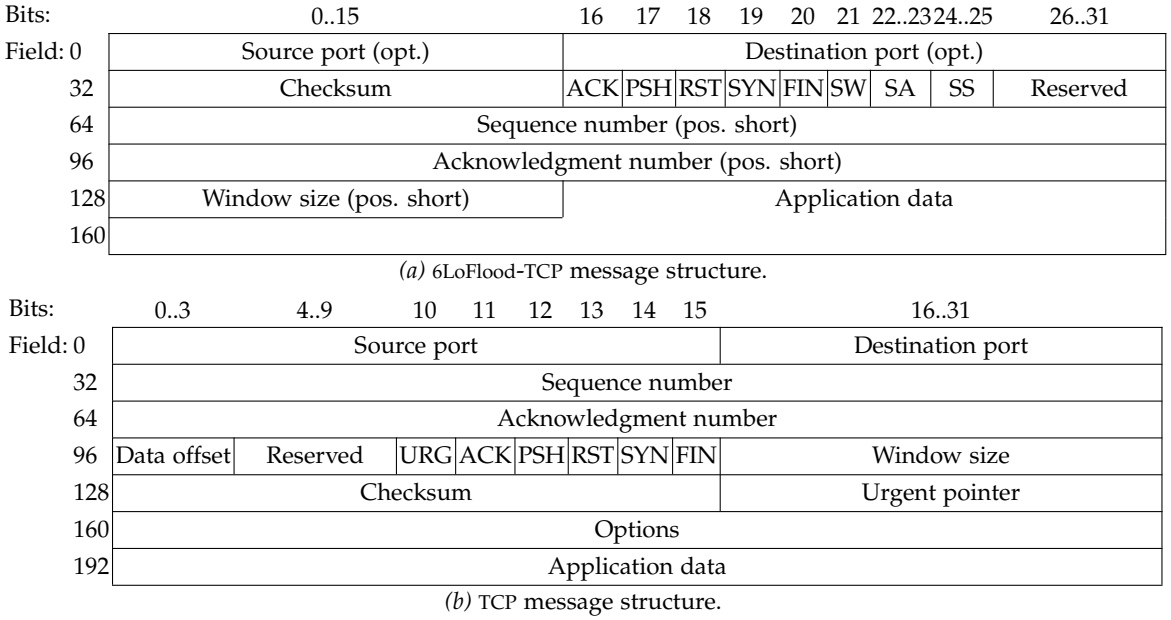
its checksum. It then compares the computed checksum with the received check sum to assure integrity.

TCP The Transmission Control Protocol (TCP) provides a reliable and ordered message exchange between two devices. Its message structure is illustrated in Figure A.6b and the compressed 6LoFlood message is shown in Figure A.6a. In the following, we explain the single fields in the message in more detail and also detail our compression strategy.

- *Source* and *destination port* As in UDP, 6LoFlood omits these fields when the *SPE* or *DPE* in the 6LoFlood header differ from `000`.
- *Checksum* The checksum provides data integrity and therefore, is transmitted uncompressed. For the integrity check, the receiver re-creates the TCP messages based upon the 6LoFlood-TCP messages, computes its checksum and compares it with the received one.
- *Sequence number* This field is used for message reordering and loss detection. If the *SYN* flag is set, this field carriers the initial sequence number. If the *SYN* flag is not set, this field is increased byte-wise, i. e., by the size of the application data, for each transmission. The initial sequence number is negotiated during the TCP handshake. Depending on the current sequence number and the amount of bits it requires, 6LoFlood reduces the size of the sequence number field. The resulting size is encoded in the *Short Sequence number (SS)* flag. More precisely, the SS flag being set to `00` indicates a resulting field size of 1 byte, the flag being set to `01` denotes a 2-byte sequence number, and so on.
- *Acknowledgment number* This field holds the sequence number up to which a destination acknowledges the reception of packets. This field is only valid if the *ACK* flag is set. Thus, 6LoFlood omits this field when the *ACK* flag is unset. However, if the flag is set, 6LoFlood reduces the field size of the acknowledgment numbers depending on

Table A.4: Port number encoding in 6LoFlood.

000	port numbers are sent in-line in full
001	port 80 for HTTP
010	port 8080 as alternative HTTP port
011	reserved for future use
100	port 3001, first default UDP port in LaneFlood
101	port 3000, second default UDP port in LaneFlood
110	reserved for future use
111	reserved for future use



(a) 6LoFlood-TCP message structure.

(b) TCP message structure.

Figure A.6: Compressed 6LoFlood-TCP header vs. "standard" TCP header.

the bits the number currently requires, similar to the sequence number field. The size of the field is encoded in the *Short Acknowledgement number (SA)* flag with the same encoding scheme as used in the SS flag.

- *Window size* This field is used for flow control to indicate the sender how many bytes to transmit until the queue at receiver side is full. Similar to the sequence and acknowledgment numbers, this field can be shortened depending on the current value. More precisely, if the *SW (Short Window)* flag is set, the window size field is reduced to 1 byte.

- *Urgent pointer* If the *URG* flag is set, the data after the TCP header is immediately processed by the application. The application, thus, stops processing the data of the current TCP segment and reads out all bytes after the header up to the byte pointed to by the urgent pointer field. However, most applications do not process the *URG* flag. 6LoFlood, thus, removes the *URG* flag and the urgent pointer field. In case the border router receives a TCP packet where the *URG* flag is set, it drops the packet. We leave the implementation of a notification message via ICMP for future work.

- *Options* TCP allows to transmit additional information. Since this field is usually ignored in low-power wireless networks, it could be omitted in 6LoFlood. However, the receiver cannot create the original TCP header for integrity check without knowing the options. In our implementation, the border router drops packets that are

not compressible by 6LoFlood without further notification. We leave the design of a corresponding error message for future work.

A.1.3.3 *6LoFlood packet sizes*

Using 6LoFlood, we can significantly reduce the header sizes of IPv6, UDP, and TCP. Table A.5 summarizes the savings when using 6LoWPAN or 6LoFlood. Local indicates that a message is exchanged only within the low-power wireless network while global includes a second network like the Internet. Thus, e. g., *Local, UDP* refers to IPv6 with a link-local IPv6 prefix and UDP on top of it.

As shown in the table, the compressed IPv6/UDP headers result in a similar size for 6LoWPAN and 6LoFlood, independent of whether the IPv6 prefix is local or global. This is because the compression methods are almost identical. The resulting savings range from 80 % to 90 % with a link-local IPv6 prefix and 45 % to 60 % in a global scenario. Using TCP instead of UDP, 6LoFlood saves 65 % to 85 % in the local case and 40 % to 60 % in the global case. The 6LoWPAN standard does not yet support TCP header compression, even though a draft exists since 2011 [8].

A.1.4 *Summary*

In this section, we illustrated the conceptional design of 6LoFlood to enable the interoperability and interconnectivity of different networks. In particular, we discussed the node addressing and message forwarding, the packet fragmentation, and the header compression. However, the main difference between 6LoFlood and 6LoWPAN is the absence of any routing and neighbor discovery mechanisms in 6LoFlood. This reduces the total memory consumption of 6LoFlood/LaneFlood, which has to be further evaluated.

Table A.5: Resulting packet sizes using 6LoWPAN and 6LoFlood.

Scenario	*Uncompressed*	*6LoWPAN (HC1 + HC2)*		*6LoFlood*	
		Size	*Savings*	*Size*	*Savings*
	[byte]	*[byte]*	*[%]*	*[byte]*	*[%]*
Local, UDP	48	6 – 10	79.17 – 87.5	4 – 8	83.3 – 91.67
Local, TCP	60	—	—	9 – 20	66.67 – 85.0
Global, UDP	48	22 – 26	45.83 – 54.17	20 – 24	50.0 – 58.3
Global, TCP	60	—	—	25 – 36	40.0 – 58.3