Delft University of Technology
Master's Thesis in Embedded Systems

Enabling the Chaos Networking Primitive on Bluetooth LE

Coen Roest

TU Delft Delft University of Technology

Enabling the Chaos Networking Primitive on Bluetooth LE

Master's Thesis in Embedded Systems

Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Coen Roest
c.roest@student.tudelft.nl

12th October 2015

Author
Coen Roest (c.roest@student.tudelft.nl)
Title
Enabling the Chaos Networking Primitive on Bluetooth LE
MSc presentation
20th October 2015

Graduation Committee

Prof. Dr. K.G. Langendoen (Chair)	Delft University of Technology
M. A. Zuniga Zamaloa, PhD. (Supervisor)	Delft University of Technology
O. Landsiedel, PhD. (Daily Supervisor)	Chalmers University of Technology
A. Bozzon, PhD. (External Member)	Delft University of Technology

Abstract

Cyber-Physical Systems (CPS) integrate physical processes, sensors, and embedded computers to facilitate advanced control systems such as autonomous cars and *smart cities*. Communication in CPS has tight constraints regarding reliability and latency, while traditional networking primitives can not guarantee these constraints. We base our thesis on the Chaos networking primitive which is a new paradigm that overcomes these limitations of traditional networking primitives. The current radio standard on which Chaos relies, is used more in science and industry, but less in everyday devices. We want to bring Chaos from the lab to the real world by porting the primitive to Bluetooth Low Energy (BLE).

This thesis presents Chaos BLE: an implementation of the Chaos communication primitive on Bluetooth Low Energy. We characterised the capture effect on BLE and achieved accurate time synchronisation among nodes ($< 2.5\mu s$), both of which are key for the operation of the Chaos primitive. We validated our design and implementation on a 25 node BLE testbed that we have build at Delft University of Technology.

In order to improve the performance of Chaos BLE and to mitigate channel interference, we propose a multichannel approach. The Chaos Multichannel primitive enables the network to use multiple channels in parallel, such that smaller sub-networks arise. Chaos Multichannel outperforms the single channel primitive in terms of reliability and latency. It achieves reliabilities close to 100% while finding a consensus among the nodes up to 2 times faster, compared to the single channel approach.

Acknowledgements

The work in this thesis was partly done at Chalmers University of Technology (Gothenburg, Sweden) in the Networks and Systems division, Department of Computer Science and Engineering. It was funded by the IDEA League student grant. Their cooperation is hereby gratefully acknowledged.

Preface

This MSc. thesis reflects upon the work I performed in the last 9 months. I spend the first 6 months at Chalmers University of Technology, Gothenburg, Sweden and the last three months at the Embedded Software group, Delft University of Technology. Living, learning, and working in a different country was an invaluable experience in which I grew personally and professionally.

Research is like climbing a mountain covered in clouds: you have an idea where the top is but you don't know how to get there. Climbing this mountain would not have been possible without the help of others. My earnest gratitude goes out to my supervisor Olaf Landsiedel from Chalmers University of Technology. His positive attitude and intelligence made my period in Sweden a great and enriching experience. The many discussions we had sharpened my thinking, pushed me to go further, and encouraged me to ask the right questions. Special thanks go out to my supervisor from Delft University of Technology: Marco Zuniga. His patience, knowledge and bird′s eye view were immensely helpful while writing this thesis. I would also like to thank Beshr Al Nahas for our discussions and the tips and tricks regarding the practical part of this work.

Many thanks go out to my friends and colleagues for their company, distraction, and input. I want to thank Rick Fransman for his friendship, his humour, and for preventing me from writing all sorts of bad grammar. My love, respect, and gratitude go out to my family and my parents, Martin and Jacqueline, for all their support during my studies. Finally, I am very grateful to my wonderful girlfriend Janneke for her caring, attention, and love. Thank you for being there for me, in Sweden, The Netherlands, and my life.

Coen Roest

Delft, The Netherlands
12th October 2015

Contents

Chapter 1

Introduction

Our society is becoming more reliant on wirelessly connected devices that operate without any human involvement. Typically, those devices are embedded and consist of sensors, a low-power chip for processing, and a radio module that can be used to communicate with other devices. These types of systems are often called Cyber-Physical Systems (CPS) and are used to sense, process, and communicate data in various environments. An example of CPS is an autonomously driving car that needs to drive cooperatively with other cars. At a crossroads, the cars have to decide real-time who gets priority. Another example is a swarm of unmanned aerial vehicles (UAVs) that work on a common task and need to ensure that they do not collide.

Many Cyber-Physical Systems are mission critical and have stringent requirements regarding safety, availability, and efficiency. The different CPS applications all have a common denominator: they all require a *reliable*, *low-latency*, and *energy efficient* communication primitive.

1.1 Limitations on Traditional Networking

A typical operation in CPS is all-to-all communication. This communication scheme is used in situations where each node in the network makes a decision based on information of every other node. Appointing a master node or agreeing on the detection of an event are examples of this scheme. Traditional systems achieve all-to-all communication by using three distinct phases: data aggregation, computation, and data dissemination.

In the data aggregation phase, the measured sensor data from each node is sent to a central node, called the sink. The sink processes the received data in the computation phase. Finally, the sink sends the processed data back to all the other nodes in the data dissemination phase. A downside of this approach is that the sink is a single point of failure which can impair

the reliability of the system. The three-phase operation costs more time thus it results in *larger latencies* and a *higher energy consumption* per node. CPS require low latencies and traditional networking stacks can not always ensure this.

1.2 A new Approach: Chaos

The Chaos communication primitive overcomes the limitations of traditional networking stacks [12]. This new paradigm eliminates most of the communication layers and overhead of traditional networking primitives. Moreover, wireless networks are dynamic: channels can have a varying quality, and nodes can be mobile. Chaos embraces the dynamics of wireless networks and handles them in an efficient way. The Chaos primitive provides all-to-all communication and the ability to do in-network computation. Therefore, data aggregation, computation, and dissemination can be done at the same time, hence the latency and energy consumption are lower. A typical application of Chaos is finding a network-wide agreement. Chaos finds a consensus among *100 nodes within 100 ms* [12]. The main mechanism behind Chaos is the capture effect. We explain this effect in Section 2.3.1 and we discuss the Chaos primitive and its operations in Section 2.5.

1.3 Bringing Chaos from the Lab to the Real World

The current implementation of Chaos operates on a specific radio standard and hardware platform. This standard is used mostly in academic and industrial settings but not in everyday consumer electronics. If we want Chaos to have an impact, it should also be available on everyday products.

In contrast, Bluetooth and Bluetooth Low Energy (BLE) are standards which are present in almost every mobile device. Total shipments of Bluetooth capable devices are estimated to be 20 billion in 2017 [10]. Mobile phones and laptops incorporate BLE, as well as many other devices that operate autonomously include this standard. Examples of BLE enabled autonomous systems are connected lighting systems, UAVs, and smoke detectors. Chaos is a fast and reliable communication primitive for CPS and for that reason we enable Chaos on Bluetooth LE platforms in this thesis.

1.4 Challenges

Chaos is currently restricted to one specific communication standard and hardware platform. Chaos builds on top of two main pillars, the capture effect and synchronous transmissions, which are tightly coupled with this radio standard and hardware platform.

BLE has fundamental differences compared to the radio standard Chaos currently uses and it poses more stringent constraints regarding the capture effect. Additionally, due to the scale on which BLE devices are used, we anticipate that more devices operate in a close range. This can result in channel interference which impacts the performance of Chaos. This thesis addresses the following challenges:

- Characterisation of the constraints for the capture effect on Bluetooth Low Energy.
- Achieving precise time synchronisation ($< 8\mu s$) across all BLE nodes in the network.
- Enabling the Chaos primitive on Bluetooth LE hardware and controlling Chaos' operations.
- Deriving new mechanisms to mitigate channel interference which impairs Chaos' performance.

Networking primitives such as Chaos are designed, implemented, and analysed on testbeds. However, there is as far as we know no multihop Bluetooth LE testbed available yet, so we have build our own.

1.5 Thesis Contributions

This work delivers the following contributions:

- Chaos BLE: design, implementation, and evaluation of a Chaos adaption for Bluetooth Low Energy (see Chapter 4).
- Chaos Multichannel: design, implementation, and evaluation of a primitive using multiple parallel channels to improve the performance of Chaos BLE (see Chapter 5).
- A 25 node Bluetooth Low Energy testbed that we use for the evaluation of Chaos BLE (see Appendix A).

1.6 Thesis Organization

The rest of this thesis is organised as follows: Chapter 2 lays out the foundations on which this work builds. Chapter 3 describes other works related to this thesis. I present the contribution regarding porting the Chaos primitive to a Bluetooth Low Energy platform in Chapter 4. In Chapter 5, this presents a multichannel approach to improve Chaos' performance. I suggest ideas for future work in Chapter 6 and present my conclusions in Chapter 7.

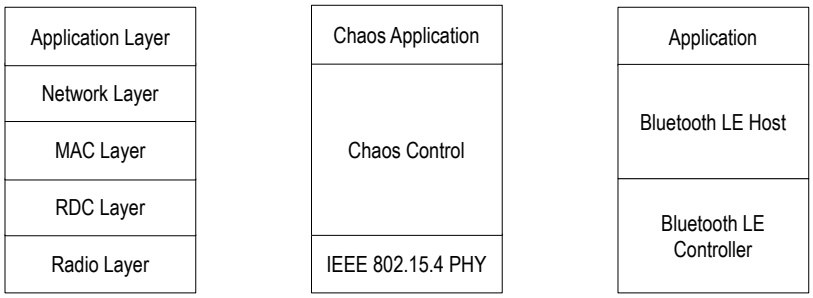
Chapter 2

Background

The background section presents the basic building blocks we need to understand for this thesis. Section 2.1 explains the traditional networking stack for low-power wireless networks. In this section we show the structure of a traditional networking stack and how it relates to the Chaos networking stack. Section 2.2 discusses the Bluetooth Low Energy (BLE) standard and compares it with the current radio standard used by Chaos: the IEEE 802.15.4 standard.
In Section 2.3 we introduce the concept of concurrent transmissions on which Chaos relies. There are two concepts that are important regarding concurrent transmissions: the capture effect and constructive interference. Section 2.3.1 explains the capture effect which is *fundamental* to the Chaos networking primitive. In order to enable Chaos on Bluetooth Low Energy we need to *characterise the capture effect for this radio standard*.
Additionally, we discuss constructive interference in Section 2.3.2. A recent work leverages constructive interference with time synchronised nodes in order to perform fast network flooding. Chaos builds upon this work to *achieve accurate time synchronisation* and we discuss this network flooding protocol in Section 2.4. Finally, we explain the Chaos networking primitive in more detail in Section 2.5.

2.1 Networking Stacks in Low-Power Wireless

Figure 2.1a shows the state of the art networking stack that low-power wireless networks use. The layers provide an abstraction of communication functions. Each layer is responsible for a subset of the communication functions and each function adds some overhead, which they require for their operation.
The highest and lowest layers shown in Figure 2.1a, the radio layer and application layer, represent the radio hardware and the application respectively. The main function of the radio duty-cycle (RDC) layer is to decrease



(a) WSN Stack (b) Chaos Stack (c) BLE Stack

Figure 2.1: **An overview of the layers in the network stacks of different communication standards.**

the energy consumption by switching the radio on only when it is needed. The media access control (MAC) layer optimises the use of the wireless channel and avoids colliding transmissions. The MAC layer performs clear channel assessment, determines the link quality, and schedules the transmissions per link between two nodes. The network layer is responsible for the routing of a packet from one node to the other. Determining neighbours, building and updating the topology, and finding an optimal route are all part of the tasks of the network layer.

Figure 2.1b shows an overview of the Chaos networking stack. It does not follow the traditional networking stack in order to increase reliability, decrease latency, and optimise energy usage as we will show in Section 2.5. Figure 2.1c presents the BLE stack and we explain it in the next section.

2.2 Bluetooth Low Energy

The first Bluetooth standard was introduced in 1999 and was designed to be a low-power, short-range, and low-cost replacement for cables [10]. Ten years after the first Bluetooth standard, Bluetooth Low Energy was introduced as part of the Bluetooth 4.0 core specification [2]. BLE is less energy consuming and has a lower latency compared to classic Bluetooth. It is designed as a low-energy control and monitoring solution, in contrast to Bluetooth V3.0 which is more focussed on larger data transfers [8]. The new Bluetooth stack introduced changes to each layer and older Bluetooth devices are not compatible with the newer Bluetooth Low Energy standard.

The protocol stack is split into two parts: the Bluetooth LE Controller consisting of the physical and link layer, and the Bluetooth LE Host which consists of upper layer functionality (see Figure 2.1c). The main focus for the Chaos BLE implementation is on the controller in general and the physical layer in specific. Table 2.1 shows a comparison between the physical layers

Table 2.1: **Physical layer comparison of Bluetooth Classic, Bluetooth Low Energy and IEEE 802.15.4 (data from [8]).** The Bluetooth Low Energy physical layer is simplified compared to classic Bluetooth and the IEEE 802.15.4 standard in order to save energy.

	Bluetooth (Classic)	Bluetooth LE	IEEE 802.15.4
Modulation	GFSK (v1.2), 4-DQPSK/8DPSK s(v2+EDR), 802.11 (v3+HS)	GFSK	O-QPSK
Data Rate (kbps)	$\leq$ 721 (v1.2), 3000 (v2+EDR), $\leq$ 24,000 (v3+HS)	1000	250
Bandwidth (MHz)	1	1	2
Channel Spacing (MHz)	1	2	5
Channels	79	40	16
Spreading Technique	FHSS	FHSS	DSSS
Preamble Length (bytes)	9	1	5
Max. Message Length (bytes)	127	47	358

of classic Bluetooth, Bluetooth Low Energy and the IEEE 802.15.4 standard on which the current implementation of Chaos is build.

All three radio standards shown in Table 2.1 operate on the 2.4 GHz Industrial Scientific Medical (ISM) band. The physical layer of BLE is simplified compared to Bluetooth classic in order to save energy. The maximum packet length and preamble are shorter such that the radio has a lower radio-on time and hence, less energy consumption. Bluetooth LE has less channels with a wider spacing to reduce adjacent channel interference. A simpler modulation technique is used compared to IEEE 802.15.4 and classic Bluetooth, which allows a simpler hardware implementation and a smaller chip size.

The current Chaos implementation is tightly coupled to the IEEE 802.15.4 radio standard [12], which differs in a number of ways from the Bluetooth Low Energy radio standard. It is not possible to directly place the Chaos controller functionality on top of the Bluetooth LE controller and this introduces two challenges.

The first challenge is to characterise the BLE radio specifics required for Chaos. The second challenge is to adapt the Chaos controller such that it can operate on the BLE physical layer. We show this in Chapter 4.

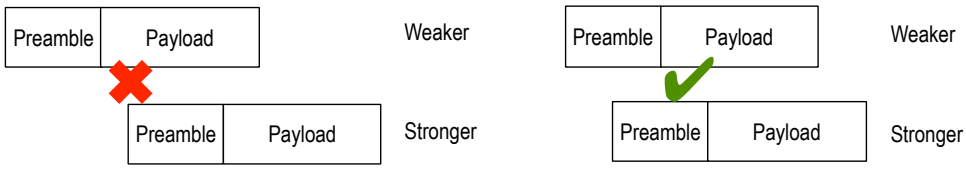


Figure 2.2: **Capture effect constraints.** A stronger packet that is transmitted concurrently within the preamble of the weaker packet can correctly be decoded.

2.3 Concurrent Transmissions

Generally, when two transmitters are sending concurrently within range of a single receiver, their packets will collide and the receiver will not be able to properly decode a packet. However, under certain conditions the receiver can still decode a signal. We distinguish two cases with different conditions: the capture effect (see Section 2.3.1) and constructive interference (see Section 2.3.2).

Chaos leverages the capture effect to perform concurrent transmissions (see Section 2.5) and controlling this effect is crucial for porting Chaos onto BLE. Constructive interference is the main mechanism behind the flooding architecture of Glossy (see Section 2.4). Chaos builds on top of Glossy to achieve accurate time synchronisation.

2.3.1 Capture Effect

The capture effect allows receivers to correctly decode a packet in the presence of interference [14]. The conditions under which this capture effect takes place are two-fold. First, the signal-to-noise ratio (SNR) between the stronger signal of interest and the interfering signal should be higher than the capture ratio. This capture ratio is radio specific and is around 3dB for the radio on which Chaos operates, the Texas Instruments (TI) CC2420 radio [26]. Second, the stronger signal may not come in later than the capture window. The capture window for IEEE 802.15.4 is 160μs which is the on-air time of the preamble [27].

Figure 2.2 is a graphical representation of the capture effect constraints. If the stronger packet comes in after the preamble of the weaker packet, both packets will collide and the receiver can not decode a signal. The collision of both packets happens likewise in the situation in which the SNR of the stronger packet is not higher than the capture threshold. However, if a new packet comes in within the preamble of the weaker packet and if the SNR is higher than the capture ratio, then the capture effect can take place.

2.3.2 Constructive Interference

Constructive interference is a baseband phenomena that can occur if the synchronously transmitted packets are exactly the same. Furthermore, the required synchronisation is much more stringent than for the capture effect. Constructive interference makes multiple packets collide non-destructively such that the superposition of the signals can be correctly decoded [5]. The synchronous transmissions should occur within the chip period T_c in order to have constructive interference [22]. This period is 0.5μs for IEEE 802.15.4 compatible receivers [4]. According to the authors, the number of concurrent transmitters can be higher than 90, while still maintaining a packet reception ratio (PRR) of 100% [4]. Constructive interference is used by Glossy to decrease the time to flood a packet through the network. Chaos builds upon Glossy and uses this flooding mechanism in the final stage of a Chaos round.

2.4 Glossy: Time Synchronised Network Flooding

Glossy is a flooding architecture that relies on concurrent transmissions and exploits the constructive interference of baseband symbols. It allows the network to rapidly flood a packet while having a probability higher than 99.99% that a node will receive the packet [7]. The three pillars on which Glossy builds, are concurrent transmissions, time synchronisation, and temporal decoupling of the Glossy network flooding from all other network activities.

The network application schedules a Glossy flood and this flood is divided into slots in which a node transmits or receives. The appointed initiator starts the flood by broadcasting a packet in the first slot (see Figure 2.3). Nodes receiving the packet in Slot 1 will rebroadcast this packet in Slot 2. More nodes overhear the packet and rebroadcast it in the consecutive slot. This cycle repeats itself until the maximum number of retransmissions is reached.

Glossy relies on synchronous transmissions in order to make the packets of the different nodes constructively interfere. This is achieved by synchronising the network with the reference time based on the first packet the initiator has sent out. In the next slot, the nodes must send out their packets within 0.5μs to benefit from constructive interference. Ferrari et al. show that the reliability is higher than 98% while having 10 concurrent transmitters [7].

In order to have a tight synchronisation between the nodes, a small and deterministic software delay is key. Glossy achieves this due to its radio-driven execution model were the state of the nodes solely changes according to activities of the radio. After sending a packet, the node goes back into a waiting state until the radio signals that there is an incoming packet. The processing during a Glossy round is minimised in order to have a small soft-

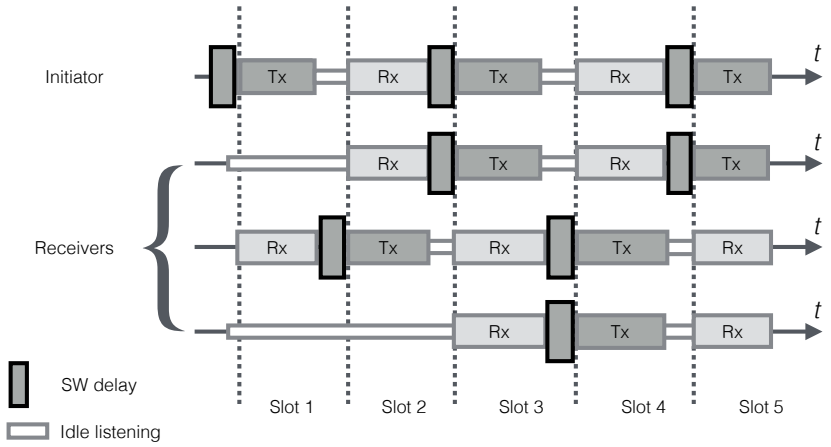


Figure 2.3: **A Glossy round in action.** The flooded packet spreads through the network. Due to constructive interference, concurrent transmitted packets are correctly received by the receiving nodes.

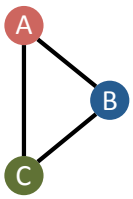
ware delay. The variable interrupt delay is compensated such that the software delay becomes deterministic and constructive interference is achieved.

2.5 The Chaos Networking Primitive

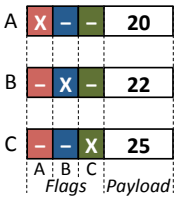
Chaos is a communication primitive for distributed systems that allows all-to-all communication and in-network processing in a fast and efficient way [12]. The three typical steps of all-to-all communication, i.e., data collection, processing, and data dissemination are combined and executed in parallel. The two key mechanisms in Chaos are synchronous transmissions and user-defined merge operators. The merge operators define how new received information should be merged with the known information, such as taking the minimum or maximum value. Chaos leverages the idea of synchronous transmissions used in Glossy, but is different since nodes transmit packets with a varying payload instead of flooding the exact same packet. This means that constructive interference can not occur and hence, Chaos relies on the capture effect.

Terminology

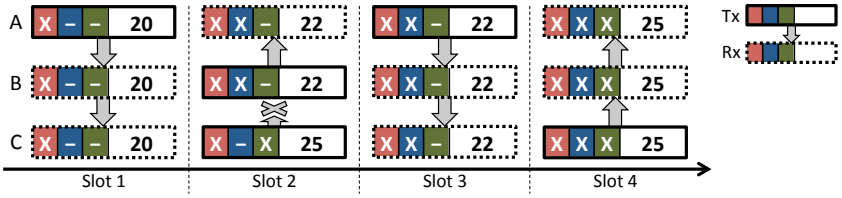
The Chaos primitive schedules a *round* in which all participating nodes run a specific Chaos application. The *rounds* are divided into multiple *slots*. The nodes can use each *slot* to either transmit, listen or switch off. One node, called the *initiator*, starts a *round* and the other nodes first need to synchronise to the *initiator* to be able to transmit concurrently within each *slot*.



(a) All nodes are in range of each other.



(b) The nodes set their flags and payload.



(c) Initiator node A starts the round by transmitting a packet. Node B and C receive the packet, merge their flags and payload, and transmit in the following slot. Node A receives the packet from node B due to the capture effect and replies with a merged packet. Node C has learned new information in Slot 4 and therefore transmits the merged information. The Chaos round ends at the end of Slot 4 since all nodes have the maximum value in the network.

Figure 2.4: **A basic Chaos application: find the maximum value from all nodes** (figures from [12]).

Basic Operation

Figure 2.4 shows the operation of Chaos when nodes look to determine the maximum value in the network. Node A, B and C are in communication range of each other. Each participating node in a Chaos round keeps track of the nodes that contributed to the current answer by assigning a flag per node (see Figure 2.4b). The initiating node A starts a Chaos round by broadcasting a packet in Slot 1 (see Section 2.5). The receiving nodes B and C merge their local flags with the received flags by performing a logical OR-operation. The value received from node A was not higher than the local value and hence, the nodes do not change their value. In Slot 2, both node B and C concurrently transmit the merged packets since they have received new information in the previous slot. Node A is able to receive the packet of node B due to the capture effect. It merges the received packet from node B, updates its flags and value, and transmits in Slot 3. Node B has not learned anything new and thus listens in Slot 4. Node C learns new information in Slot 3 and transmits in the consecutive slot. Finally,

everyone finds the maximum value based on the information of nodes A, B, and C in Slot 4.

Transmission Policies

The nodes in a Chaos network decide in each slot whether to transmit or not. The propagation policy used in Chaos never lets a node transmit in two consecutive slots. A node only transmits in a slot if the node has received new information in the previous slot. This ensures that the network does not get congested with old information and that new information travels fast through the network.
The transmitted packet contains the flags of the nodes that already contributed to the answer. This gives every node in the network an understanding to what degree the Chaos round is completed.

A situation could occur in which none of the nodes transmit in a particular slot. The nodes then stall because they do not have new information to transmit. In order not to have a prematurely termination of the round, a timeout policy is introduced. A node which has not received a packet in the previous three slots will start transmitting again.

Finally, a termination policy is in place to have a fast completion of the round as soon as one node has an answer based on the information of all the nodes. Section 2.5 shows in Slot 4 that Node C has found the final answer based on all nodes in the network. When a node has the final answer according to the flags in the packet, it starts flooding this answer for a number of times. Other nodes will pick up this information and rebroadcast this packet as well. The packets in this stage of the round are identical and thus constructive interference can occur. The final stage of a Chaos round is therefore similar to a Glossy flood. This increases the probability of completion at all nodes significantly while reducing the total radio on-time and thus the energy consumption [12].

Results

Chaos is evaluated on three different testbeds and measured on four different performance metrics: reliability, latency, radio on-time, and radio duty cycle. Chaos shows that it performs reliably for different payload lengths, network densities, and processing times. Chaos is compared to the state of the art protocol Low-Power Wireless Bus (LWB) [6] and the combination of the TinyOS' Collection Tree Protocol and data dissemination protocol Drip. Evaluations show that Chaos outperforms the state of the art by a factor in the range of $3 - 23\times$ in terms of energy efficiency while maintaining a reliability close to 100%.

Chapter 3

Related Work

The related work chapter is structured into three parts. We first compare Chaos with other communication primitives that leverage concurrent transmissions in Section 3.1. Thereafter we show wireless sensor networks that use Bluetooth Low Energy in Section 3.2.

The Chaos Multichannel primitive we present in Chapter 5 uses multiple parallel channels for communication. We show related work regarding the use of this technique in Section 3.3.

3.1 Communication Primitives using Concurrent Transmissions

While Leentvaar and Flint describe the capture effect as something that is unwanted and should be suppressed [14], more recent works use the capture effect to enable concurrent transmissions. *Flash* exploits the capture effect for network flooding [16]. It shows that controlled concurrent transmissions can reduce the latency of a network flood such that the theoretical minimum bound is approached. Whitehouse et al. leverage the capture effect to detect collisions and to recover the collided packets [25]. *Collocal* proposes a method based on the capture effect to reduce energy consumption in beacon based indoor localisation systems [21]. The average listening time of a mobile node can be reduced by sending out beacons concurrently while the capture effect allows beacons to be decoded correctly.

Low-Power Wireless Bus exploits constructive interference to create the wireless equivalent of a shared bus [6]. Each node in the network can communicate on this bus and this allows all-to-all communication as well as all-to-one communication. Nodes in an LWB network may access the bus according to a global communication schedule that is computed centrally on a host node. The network floods the packets through the network by using the Glossy mechanisms for concurrent transmissions of identical packets.

LWB makes the network function as one single device that runs on a single clock. The host is a single point of failure in the network, but LWB introduces a failover policy that enables other nodes to take over the host role on a different channel. Furthermore, LWB computes the schedule based on traffic demands and allocates slots in a fair way.
LWB provides all-to-all communication in a reliable, flexible, and energy efficient way compared to other state-of-the-art primitives [6]. However, LWB is not able to do in-network processing like Chaos does. The nodes need to exchange all the data first before they can process it. Comparing such a situation with Chaos shows that LWB has a 23× higher latency [12].

Sparkle is another work that leverages Glossy for its ability to do synchronised concurrent transmissions and providing a high reliability and a low latency [28]. It focusses on Cyber-Physical Systems (CPS) in general and one-to-one communications in CPS in specific. Glossy is not suitable for one-to-one communication since it uses the complete network to flood a packet from one point to the other. Sparkle picks a subset of nodes that form a path between the source and destination. This allows nodes that are not in the path to switch off and hence save energy.
Sparkle saves up to 84% in energy consumption compared to Glossy while still satisfying reliability requirements. Moreover, Sparkle improves the communication latency in the control loops of Cyber-Physical Systems. Sparkle relies on the endpoints of the one-to-one communication for the processing unlike Chaos, which provides a way to do in-network processing.

3.2 Bluetooth LE Based Sensor Networks

In most wireless sensor networks (WSN), IEEE 802.15.4 is the de facto radio standard. However, new works attempt to enable Bluetooth Low Energy (BLE) for these networks. In a typical wireless sensor networks, all nodes can serve both as a master or slave. The Bluetooth Specification v4.0 forces each node in a piconet to be either master or slave [2]. However, this restriction is let loose in the newer Bluetooth Specification v4.1. A connection between piconets can now be setup to form a multi-hop scatternet and nodes from different piconets can communicate with each other.

Guo et al. propose an on-demand scatternet formation and multi-hop routing protocol [9]. The work uses the full BLE stack and nodes can dynamically join or leave the scatternet since the routing is done on demand. To find a route between two nodes, the sender initiates a route discovery process. The piconet master searches in its slave list for the requested destination. When the destination is not found in the list, the master forwards the route request to slaves that are also part of another piconet. The possible routes are returned to the destination which chooses then the shortest

path.
Results show that the average delay to send a data packet of 128 bytes to a destination in a scatternet is large. It takes up to 1 second for the transmission of the packet with 5 nodes and two hops and almost 5 seconds for a 9 node scatternet with four hops. Due to the on-demand nature of the routing, it takes a long time to route a packet from one node to the other. All-to-all communication would introduce even larger delays. Chaos does not follow this traditional approach on communication and thus can achieve lower delays.

Lin et al. study BLE as a technique to use in intra-vehicular wireless sensor networks (IVWSN) [15]. Currently, the sensors and electronic control units (ECUs) in a car are connected through wires. Since more sensors are added to modern cars, this results in more cables and thus a higher cost and extra weight. This can be overcome by switching to wireless sensors and the authors do a feasibility study of BLE for this purpose. The work evaluates the packet reception ratio (PRR) of BLE for different locations in the car and BLE achieves over 95% in different scenarios.

The authors show that BLE is a promising technique for IVWSN. However, challenges remain regarding the MAC layer in order to facilitate high reliability and low latency for multiple sensors. The Chaos communication primitive shows that it can achieve a high reliability and a low latency on IEEE 802.15.4 compliant motes. We contributed a Chaos BLE port which can bring low latency and high reliability to Bluetooth Low Energy compliant motes and present this in Chapter 4.

3.3 Parallel Channels

Most multichannel communication primitives introduce frequency spreading by using channel hopping [24], [20], [1]. However, The Chaos Multichannel solution we propose in Chapter 5 uses multiple channels in parallel. This is a fundamentally different technique and we discuss two works that use such parallel channels.

McMAC is a parallel MAC protocol that uses multiple channels to avoid control channel congestion [19]. It focusses on improving the rendezvous phase performance by using multiple channels. Multiple pairs of nodes can agree simultaneously on using a channel for their communication. McMAC also makes the nodes estimate the relative clock speeds of their neighbours to ensure time synchronisation. It achieves a time synchronisation between neighbours within $200\mu s$.

Another MAC protocol that uses a multichannel approach is Y-MAC [11]. Y-MAC is contention-based and only uses multiple channels under heavy traffic. Nodes communicate on a base channel and nodes switch to other

channels when traffic bursts occur. Only one node is allowed to transmit on the base channel in a single slot. Y-MAC is TDMA-based and achieves a time synchronisation of $51\mu s$.

Y-MAC [19] only enables multichannel under heavy traffic and McMAC [11] uses multichannel solely for rendezvousing. Chaos Multichannel uses a different approach by adding multichannel to Chaos BLE. Chaos does not follow the traditional network stack and hence does not need to do rendezvousing, or account for traffic bursts. We present the Chaos Multichannel primitive in Chapter 5.

Chapter 4

Enabling Chaos on Bluetooth LE

We present the design, implementation, and evaluation of the Chaos implementation on Bluetooth Low Energy (BLE) in Chapter 4. Figure 4.1 shows the various components of Chaos BLE. We present our design for characterising the capture effect of Bluetooth LE (1) and the time synchronisation mechanisms (3) in Section 4.1. Section 4.2 shows the implementation details of the radio control (2) and the time synchronisation (3). We evaluate the capture effect (1) and the time synchronisation performance (3) in Section 4.3. This section also presents the evaluation of the overall Chaos BLE performance using a maximum aggregation application (5). We achieve further improvements in the performance using a multichannel approach (4) which we will present in Chapter 5.

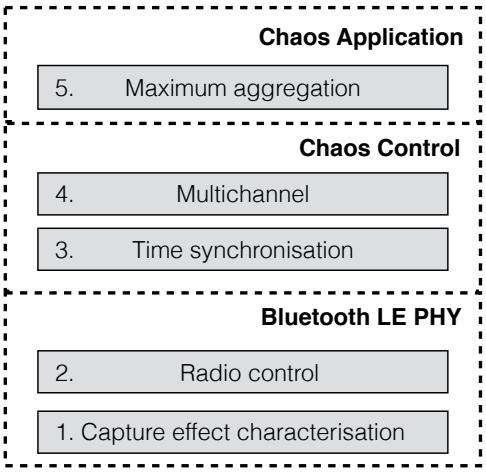


Figure 4.1: **Overview of Chaos BLE and its components.** Chapter 4 presents components $1-3$ and 5. We discuss component 4 in Chapter 5.

4.1 Design

We present the hardware platform we use for the design of Chaos BLE in Section 4.1.1. The design of Chaos BLE focusses on two parts: characterisation of the capture effect on Bluetooth Low Energy in Section 4.1.2 and achieving accurate time synchronisation in Section 4.1.3.

4.1.1 Hardware Platform and Operating System

Porting Chaos to a new *radio standard* also involves porting it to a *different chip and hardware platform*. The hardware platform we use for this thesis is the Nordic Semiconductor nRF51822. It is an ARM Cortex M0 System on a Chip (SoC) with a BLE compliant radio. The CPU runs on a 16MHz clock and it has a 32 bit timer available. Many BLE connected devices use this platform for all sorts of applications since it is a low-power, low-cost SoC with a small footprint.

The current Chaos implementation operates on the TMote Sky hardware platform. The main difference with the nRF51 is that our platform is a SoC and thus combines the CPU and the radio in a single chip. This results in our case in less control on the operation of the radio by the CPU of the nRF51. The original Chaos networking primitive uses Contiki, an operating system for low-power wireless embedded systems. We also build Chaos BLE using Contiki such that Chaos applications can run on both platforms without any adaption.

4.1.2 Characterising the Capture Effect on Bluetooth LE

Chaos relies for its operations on the capture effect [12]. The first step in porting Chaos on a Bluetooth Low Energy hardware platform is thus to get insight in the capture effect on BLE. *Collocal* shows that the capture effect is present in Bluetooth Low Energy compliant radios [21]. However, the authors do not specify details about the required power ratio threshold for capture or the maximum time window in which the capture effect can take place. The comparison of the required thresholds for capture on BLE and IEEE 802.15.4 gives us an insight on the differences between the two radio standards.

The nodes in the network should send synchronously in order to increase the probability of capture. The time window required for capture on BLE, gives a margin in which the nodes should synchronously transmit. If the nodes transmit concurrently outside this margin, the packets will collide and can not be correctly decoded. Hence, the time window gives a maximum bound on the required time synchronisation between the nodes in the network. We propose a procedure to characterise the capture threshold and the maximum time window required for capture in Bluetooth Low Energy.

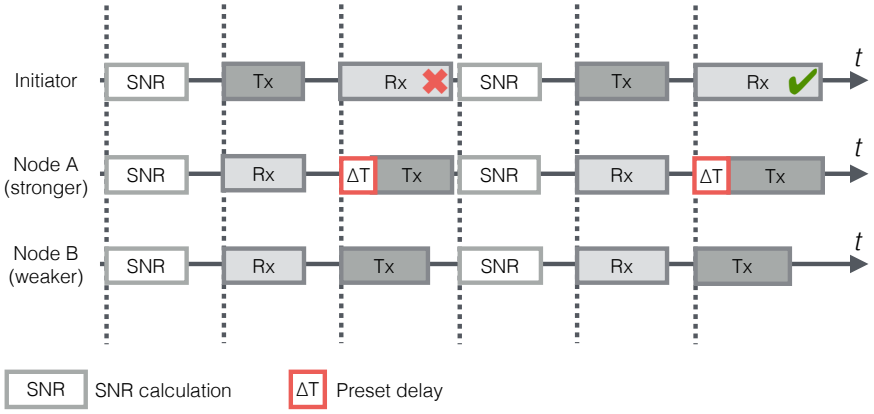


Figure 4.2: **Procedure for the capture effect characterisation.** The stronger transmitter node A sends out the packets with a preset delay. The steps to determine the SNR between the transmitters are consolidated for clarity.

The procedure uses a small network with 3 nodes that are in communication range of each other. An initiator node starts the run and acts as the receiver where the capture effect should take place. First we determine the received signal strength indicator (RSSI) of node A and B at the initiator side and calculate the signal-to-noise ratio (SNR) from this (see Figure 4.2). Thereafter, the initiator sends out a packet which node A and B receive and they both reply with some delay. Node A transmits a packet with an extra preset delay compared to node B and with a higher output power such that it is considered stronger by the receiver. We then log if the initiator received the packet and what the SNR and difference in delay are. We determine the SNR before every run since a small movement of a node can already result in a variation in the SNR.

4.1.3 Time Synchronisation

As we will show in Section 4.3.1, Bluetooth Low Energy has tighter constraints regarding the capture effect compared to IEEE 802.15.4. BLE nodes should be able to transmit synchronously within $8\mu s$ compared to $160\mu s$ for IEEE 802.15.4 nodes [3]. This is mostly due to the fact that the preamble of BLE is 1 byte compared to the 5 byte preamble of IEEE 802.15.4. Furthermore, the on-air data rate of BLE is four times higher than the rate of IEEE 802.15.4 (see Table 2.1). This results in a time window which is 20× smaller than in the original Chaos implementation.

One of the main features of Chaos is its ability to do in-network processing. Nodes merge the received information with their local information and compute an answer, e.g, what the maximum value is. This means inher-

ently that the different packets have different processing times. However, the transmissions in a slot need to overlap sufficiently in order to exploit the capture effect and to have a reliable communication. We need to ensure that *the time between two consecutive transmissions is constant while the processing times vary.* This is the key reason why we require accurate time synchronisation between nodes.

Chaos already incorporates synchronisation mechanisms but these are *radio and platform specific.* We need to adapt those mechanisms to the Bluetooth Low Energy platform and tune them to achieve the required synchronisation. We divide the design for time synchronisation into two parts: the reference time to which every node synchronises and the mechanisms to keep in sync with the reference.

Reference Time

The Chaos IEEE 802.15.4 implementation builds the synchronisation mechanisms around the start of frame delimiter (SFD) of the transmitted and received frame. The IEEE 802.15.4 receiver listens to incoming signals and searches for the SFD byte in the frame. When this byte is received, the radio locks onto this incoming transmission and fills its buffer with the packet. The CPU timestamps the detection of an SFD and uses this as the reference time for further transmissions. This timestamp, combined with the slot number in the packet, determines when the next slot is scheduled and when the actual round began. The next SFD should be timestamped exactly one slot length later than the previous one.

In contrast, Bluetooth Low Energy has a short and simplified frame layout and lacks a SFD. Thus, we need to find another reliable reference point for incoming and outgoing packets. The radio needs to signal to the CPU when it should timestamp the packet. The first byte in a BLE packet is the preamble which is used by the receiver to lock onto the incoming packet. [2]. After the preamble is received, the radio reads the following four bytes containing the BLE radio address. The radio of the nRF51 generates a trigger after it decoded the address in the frame (see Figure 4.3). We use this trigger to timestamp the packet and this provides us the same functionality as the SFD detection in IEEE 802.15.4 radios. However, the internal workings of the nRF51 radio are opaque. It is uncertain if the trigger after the address decoding has a constant and low latency as the SFD trigger in the IEEE 802.15.4 radios that the original Chaos primitive uses.

Synchronisation Control

The initiator sends out a first packet and the other nodes use this as the reference time on which they base the synchronisation. We define a constant slot length such that the nodes in the network can determine their next

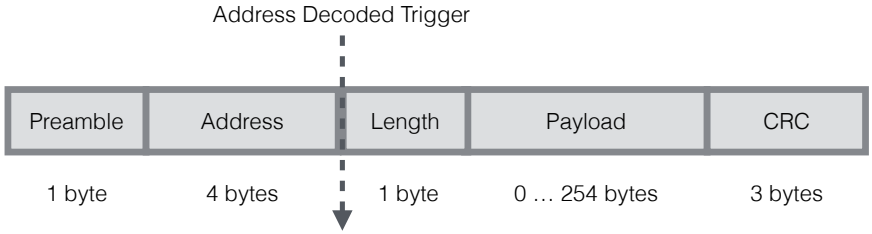


Figure 4.3: **Bluetooth Low Energy packet layout.** The radio reports a trigger after the address in a transmitted or received packet is decoded.

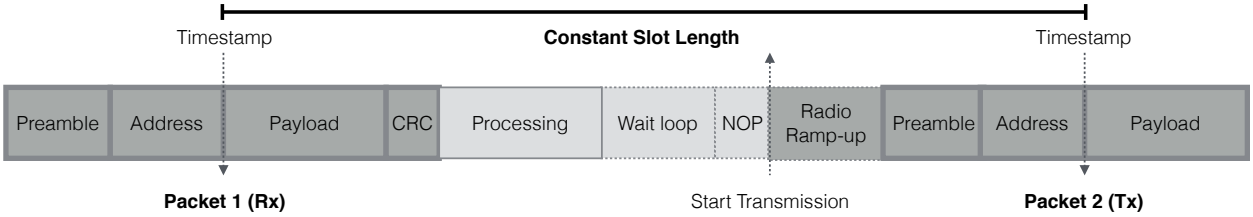


Figure 4.4: **Detailed view of a Chaos BLE slot.** The first packet is received by the node and the second is transmitted. The processing time is variable and need to be compensated for by two phases of waiting.

transmission time based on this length and the reference time. Figure 4.4 shows a more detailed view of a slot in Chaos BLE.

The node timestamps the incoming packet after it decodes the address. The radio keeps listening for the remaining part of the packet and checks its validity. After the packet is correctly received, the node processes the packet and decides whether or not it should transmit in the next slot. In order to meet the next deadline for transmitting a packet, we need a very precise control over the start of a transmission. The remaining time before the transmission starts, is filled with a wait loop with a coarse granularity and *no operations* (NOPs) with a fine granularity. We use the coarse wait loop to wait for the major part of the waiting time and we use the NOPs for the remaining smaller part.

Equation (4.1) shows the relation between the different durations of each element in a slot. The total slot length must be kept *constant* while the processing time is *variable*. The duration for the various parts of a packet is constant. The preamble, address, payload, and CRC all have fixed sizes which gives them a fixed duration. The radio ramp-up that follows after the start of the transmission in Figure 4.4 is the time the radio needs to startup and tune in to the used frequency. In Section 4.2.1 we show how we ensure a constant ramp-up time. The processing time is variable and thus we fill the time that is left over with waiting in two different phases: T_{wait} and T_{NOP}.

Section 4.2.2 goes more into detail about these waiting phases.

$$\underbrace{T_{slot}}_{\text{Constant}} = \underbrace{T_{ramp-up} + T_{preamble} + T_{address} + T_{payload} + T_{CRC}}_{\text{Constant}} + \underbrace{\overbrace{T_{processing}}^{\text{Variable}} + \overbrace{T_{wait}}^{\text{Variable}} + \overbrace{T_{NOP}}^{\text{Variable}}}_{\text{Constant}} \quad (4.1)$$

Note that interrupts are switched off during a Chaos round. The sole task of a node during a Chaos round is to process incoming packets and to achieve an accurate synchronisation. Energy saving measures, such as switching to a sleep mode after processing the packet, are not desirable. Accurate timers are switched off during sleep mode and this will result in decreased precision and missed deadlines.

Deadline Adjustment

When a node expects to receive a packet in a particular slot, the radio starts listening to the channel some time earlier. The Chaos synchronisation mechanism incorporates some flexibility to deal with packets that are received a bit earlier or later than the exact deadline. When a node correctly receives a packet with some offset, it adjusts the next deadline to this offset. This flexibility enables the network to compensate for the clock drift or multipath effects that can occur during a round. However, rigid deadlines are more desirable in a larger multi-hop network since clocks drift independently from each other. A node adjusts its deadlines to the incoming packets of its neighbours. In larger networks it can happen that nodes do not share any neighbours. Two nodes that are multi-hops separated from each other could thus end up with very different deadlines due to the adjustments of the deadlines. We added the time synchronisation mechanism with rigid deadlines to Chaos BLE and we evaluate the different types of deadline adjustment in Section 4.3.3.

We showed the design for porting the Chaos primitive to Bluetooth Low Energy in the previous section. Section 4.2 goes into the details of the implementation and Section 4.3 shows the evaluation of Chaos BLE.

4.2 Implementation

In this section we discuss the implementation of Chaos BLE. We structure this into two parts. First we look into the specific implementation details of the nRF51 hardware platform in Section 4.2.1. Thereafter, we go into

more detail on the busy waiting method of the time synchronisation in Section 4.2.2.

4.2.1 nRF51 Shortcuts

In order to make the time synchronisation more accurate, we need to have deterministic timings in the nRF51. The radio should always send out a packet with a fixed delay after we executed a start command. Furthermore, the time-stamping of a packet should happen immediately or with a fixed delay that we can account for. The Nordic Semiconductor nRF51822 allows different parts of the SoC to communicate with each other without interception of the CPU. Such a connection is called a *shortcut* and minimises the delay between a `task` and an `event` of one or multiple peripherals. We explain the two most important *shortcuts* used in Chaos BLE.

Time-stamping

The time synchronisation mechanism uses time-stamped packets for its operations. The timer should capture the time at which a packet is transmitted or received such that it can be used for further processing. It is not desirable to let the CPU handle this task. The timestamp can have a variable offset due to the interrupt and processing delays of the CPU. We timestamp the packets by using the `address decoded event` of the radio. The radio triggers this event after it decodes an address in the incoming or outgoing packet. The timers in the nRF51 can capture the current time by triggering a `capture task`. The time synchronisation mechanism then uses this captured time. We configured the nRF51 such that the `address decoded event` of the radio directly triggers a `capture task` of the timer. This *shortcut* is faster and more constant than using interrupts to notify the CPU which can then trigger the timer to capture the current time.

The timings in the time synchronisation mechanism are based on the 32bit timer of the nRF51. We need the highest precision in order to have a reliable time synchronisation and thus we register every clock tick with the timer. This gives us $\frac{2^{32}}{16Mhz} \approx 268s$, i.e., more than four minutes before the timer does a wrap-around. The probability of a wrap-around occurring during a round is low since Chaos is aimed at having a low duty cycle. However, if the wrap-around occurs during a slot, this will only affect that particular slot. The node misses its deadline but resynchronises on the following received packet.

Transmission Sequence

The time between the transmission start command and the time-stamping of a packet should be deterministic to ensure accurate time synchronisation.

Before the radio of the nRF51 can transmit or receive a packet over the air, it needs to go through a ramp-up cycle. In the ramp-up cycle, the radio tunes into the specific frequency of the channel used for the communication. Preliminary experiments have shown that the time to ramp-up the radio is held constant by the nRF51. After the ramp-up, the radio sets an `event` register to notify that the radio is ready for transmission. An interrupt could then notify the CPU to handle this `event` and start the actual transmission. This processing delay can be variable due to interrupt priorities and the current state of the CPU. We can not account for variable delays that occur after we started a transmission. We overcome this by creating a shortcut between the ramp-up cycle and the start of a transmission by the radio. This ensures that there is always a fixed delay between triggering a transmission and time-stamping the on-air packet. Since the complete transmission sequence from the start to end is now deterministic, we can account for these delays and ensure accurate time synchronisation.

4.2.2 Busy Waiting

The synchronisation control of Chaos BLE incorporates a waiting loop to compensate for the variable processing delays (see Figure 4.4). We show a more detailed view of the busy waiting method in Figure 4.5. The waiting loop is divided into two phases in order to have a high granularity of timing control. The first phase is the wait loop which is implemented by a `while()` loop. This loop compares the current time with the goal start time for sending a packet. If the current time approaches the goal time closely it escapes the `while()` loop. However, the loop alone is too coarse to achieve high precision. The comparison and the loop itself cost instructions and thus clock ticks. This can result into overshoots and therefore decreased precision.

To compensate for the overshoot, we check how much the overshoot is and we execute a number of NOPs afterwards such that the exact deadline can be met. Adding NOPs during the execution of the source code introduces extra delays. Hence, we put a maximum number of NOPs in the source code. The unnecessary NOPs are skipped by adding the required number of NOPs to the program counter. Chaos directly triggers the transmission sequence after escaping the fine grained waiting loop.

The transition from the first waiting phase to the second waiting phase happens as soon as the pre-deadline is met. Equation (4.2) shows how the mechanism sets the pre-deadline at a specific time before the actual deadline for the start of the transmission sequence. There are three delays that we considere: the calculation time of a jump (i), the jump execution time (ii), and the time a *no operations* instruction costs (iii).

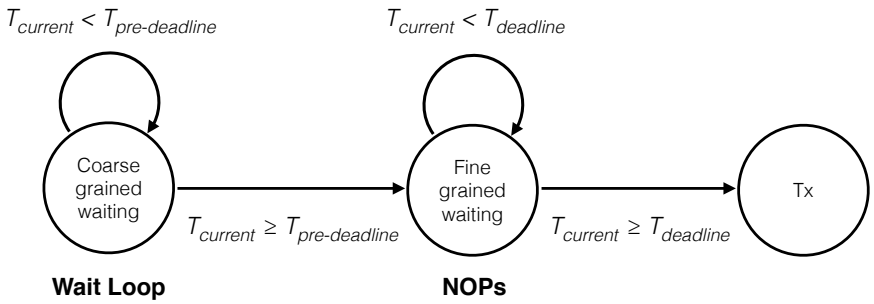


Figure 4.5: **Schematic view of the two-phased waiting loop.** The coarse grain waiting is done with a `while()` loop, the fine grained waiting is done by executing a number of NOPs.

$$T_{pre-deadline} = T_{deadline} - 2 \times \mathrm{NOP}_{\max} - T_{\mathrm{jump-calculation}} - T_{\mathrm{jump-execution}} \tag{4.2}$$

Calculating the amount of NOPs the fine grained waiting loop requires, costs a fixed time: $T_{\mathrm{jump-calculation}}$. The execution of a jump by adding a value to the program counter costs: $T_{\mathrm{jump-execution}}$. The pre-deadline is set with some room for small overshoots. The maximum overshoot is mostly determined by the maximum number of NOPs placed in the source code: $\mathrm{NOP}_{\max}$. A NOP instruction in the ARM architecture takes two clock cycles and thus we multiply it by two.

4.3 Evaluation

We perform the evaluation of the Chaos BLE primitive in four parts. First we evaluate the capture effect on Bluetooth Low Energy, then we evaluate the capture effect performance for multiple synchronous transmitters. Thereafter we have a look at the synchronisation performance of Chaos BLE. Finally, we do a complete evaluation of the performance of Chaos BLE on a Bluetooth Low Energy testbed.

4.3.1 Capture Effect Characteristics

The capture effect allows receivers to correctly decode a packet while multiple transmitters concurrently send a packet. Chaos uses this to allow concurrent transmissions while still being able to correctly receive a packet. The important characteristics regarding the capture effect are the capture threshold in dB's and the maximum time window in which the capture effect can take place. The capture threshold is the required SNR between

the signal of interest and the interferers. The time window is the maximum delay between concurrently transmitted packets.

Methodology

In order to determine the capture threshold, we use the setup and procedure presented in Section 4.1.2. We first set a preset delay and then we perform 20 test runs. The test runs give us information about the measured SNR and the packet reception ratio (PRR). To determine the lower bound for which the capture effect does occur, we increase the SNR between the packets until we reach a PRR of almost 1. We determine the lower bound by decreasing the SNR until the PRR is almost 0. To increase or decrease the SNR we adjust the transmission powers of transmitters and we change the distance between the nodes.

We repeat this procedure for multiple preset delays to find the maximum time window in which the capture effect can take place. The preset delays define the time difference between the start of the interfering packet and the signal of interest such that the stronger signal of interest comes in later. We pick the first four delays to see how the capture effect on Bluetooth Low Energy behaves when a stronger packet comes in during the preamble. The capture effect on IEEE 802.15.4 radios does not occur when a stronger packet comes in after the preamble and the packets will collide. We use delays 5 and 6 to investigate what happens in such a situation on BLE. Delays 7 to 9 can give us more insight on a situation in which the stronger packet arrives during the transmission of the payload.

Results

Figure 4.6 shows the required SNR for capture against the preset delay between the weaker and the stronger packet. The green bars in the graph show for which SNR we achieve a packet reception ratio close to 1. The red bars show at which SNR no packet can be correctly decoded by the receiver. The grey area denotes the SNRs where both the correct reception of the signal of interest or packet collisions can occur.

Finding 1. *The capture threshold is more than 4dB higher for BLE than for IEEE 802.15.4*

The capture threshold for IEEE 802.15.4 compliant radios is $\approx$ 3dB [26]. The graph shows that when the packets come in at the exact same time (1), the required SNR for a good reception is around 7dB. For larger delays the capture threshold rises between 10 and 11 dB and packets will get lost if the SNR is below 5dB.

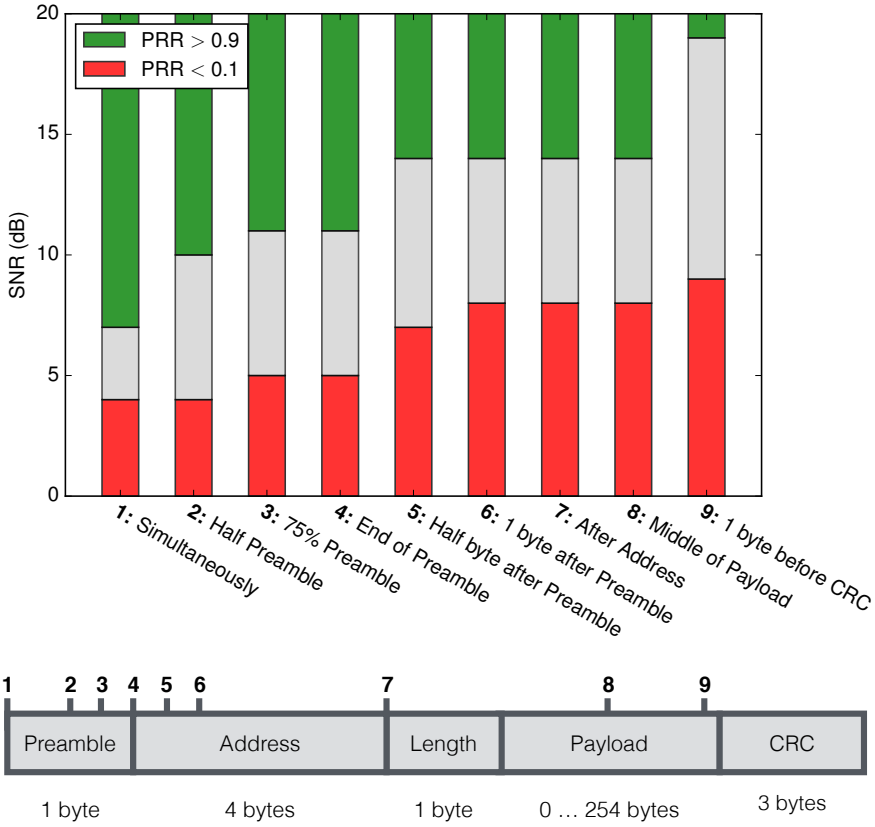


Figure 4.6: **The required signal-to-noise ratio in order to receive a packet in the presence of the capture effect vs the delay between the weaker and stronger packet.** The required threshold for capture in BLE is 7dB while it is 3dB for IEEE 802.15.4.

Finding 2. *The nRF51 radio allow message-in-message (MiM) capture.*

We define the PHY-capture region for delays within the preamble ($1-4$ in Figure 4.6). Thereafter we distinguish a second region ($5-9$). This region shows that the nRF51 allows MiM capture. With regular PHY-capture, packets with a sufficient SNR that come in after the preamble will result in a collision since the receiver is already locked to the first incoming packet. In contrast, if the radio supports MiM-capture, it can disengage from an ongoing signal reception and engage to a newer stronger signal. The required capture threshold for MiM-capture is higher than for PHY-capture and the second region follows this behaviour [18]. The capture threshold is higher for delay 9 which makes the stronger signal come in right before the CRC. The radio can not always timely disengage from the weaker signal and engage to

the stronger signal. This results in the corruption of the CRC and thus a higher threshold for the capture effect.
WiFi compliant radios allow MiM-capture [13] while IEEE 802.15.4 compliant radios do not support it. The Bluetooth Core Specification [2] does not specify MiM-capture capabilities. One explanation could be that this is a side-effect of an nRF51 radio design choice but this is outside the scope of this work.

4.3.2 Concurrent Transmissions

The tests in Section 4.3.1 give insight into the SNR threshold required for the capture effect. However, in a typical Chaos round more than two nodes are transmitting concurrently and this decreases the SNR. When the decreased SNR falls below the capture threshold, this will result in packet loss and this can affect the Chaos operation. We give some insight into the number of concurrent transmitters for which packets can still be correctly decoded.

Methodology

We use the 15 node testbed described in Appendix A for this evaluation. The nodes set their transmission power to +4dB in order to have a single-hop network where all nodes are in connection range of each other. An initiator sends out a packet and every node that correctly receives this packet can then reply. We add some randomisation in order to vary the number of nodes that concurrently send a packet back to the receiver. The initiator finally checks if a packet was received correctly or if the packets collided and calculates the PRR. We average the results of four runs with four different initiators and 29109 data points.

Results

Figure 4.7 shows the results for BLE and compares it with a similar evaluation on a different testbed that was done for IEEE 802.15.4 compliant nodes [12]. These measurements were averaged over four different receivers and 16126 data points. We find that:

Finding 3. *The PRR drops faster with more concurrent transmitters for BLE than for IEEE 802.15.4.*

The PRR for Bluetooth Low Energy drops rapidly for more concurrent transmitters. From a PRR of 0.65 for two concurrent transmitters, to a PRR of 0.08 for 8 concurrent transmitters. This is a very steep decline compared to the behaviour of IEEE 802.15.4. Every extra synchronous transmitter decreases the SNR such that it is less likely that the required capture threshold is met and therefore the perceived PRR drops. The number of concurrent

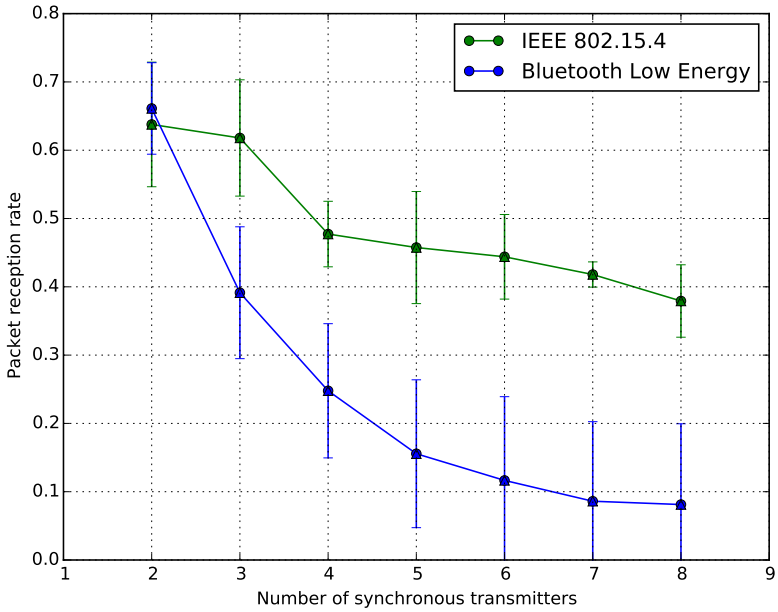


Figure 4.7: **The PRR against the number of synchronous transmitters in a BLE and IEEE 802.15.4 testbed.** The probability of receiving a packet drops faster with more transmitters for BLE than for IEEE 802.15.4.

transmitters in a Chaos round needs to be restricted in order to have more successful receptions. We propose a solution to this problem in Chapter 5. Note that the inter-node distances are different and the maximum transmission powers are higher for the BLE measurements compared to the IEEE 802.15.4 measurements. This introduces some uncertainty about the comparison between the two different evaluations of BLE and IEEE 802.15.4. However, Section 4.3.1 showed that the capture threshold for BLE is 4dB higher than for IEEE 802.15.4. Future results in Section 4.3.4 will also confirm the more harsh constraints for the capture effect and concurrent transmitters on BLE.

4.3.3 Time Synchronisation Performance

Chaos relies on the capture effect to enable concurrent transmissions. The transmitting nodes need to send the packets synchronously to allow the receiving nodes to benefit from the capture effect. The basic operation of Chaos takes place in rounds that consist of multiple slots (see Section 2.5). In every slot, each node decides if it transmits or listens and hence, the concurrent transmissions appear in consecutive slots during a Chaos round. This means that accurate time synchronisation must be achieved during

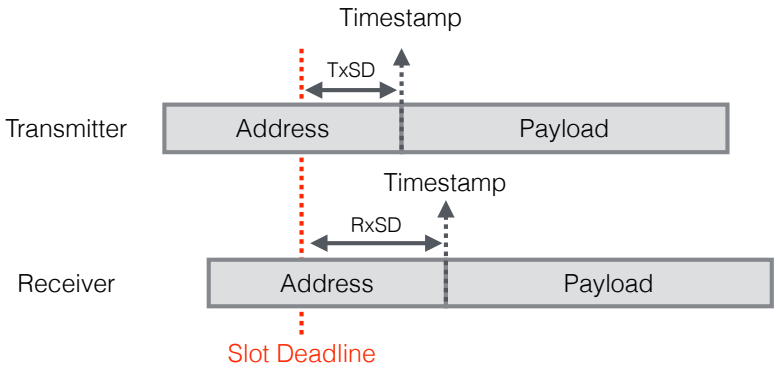


Figure 4.8: **The difference between the timestamp and the actual deadline.** TxSD is the delay the transmitter causes, RxSD is the delay the receiver observes.

each Chaos round and in each slot. The slot length is fixed and this gives the time difference between two consecutive time-stamped packets. After the nodes receive the first packet, they automatically have a deadline for the following incoming or outgoing packet.

We showed in Section 4.3.1 that a smaller delay between two concurrently transmitted packets results in a lower capture threshold. Hence, the nodes must meet the deadline of each slot as close as possible to increase the probability of capture. We evaluate the Chaos BLE time synchronisation mechanisms in this section using the following methodology.

Methodology

In every slot, the nodes have a deadline at which they should timestamp the incoming or outgoing packet. We compare the deadlines for each packet with the actual time at which the corresponding packets were transmitted or received (see Figure 4.8). The difference between the deadline and the actual timestamp is the metric on which we evaluate the time synchronisation. We call this difference the *timestamp difference* and differentiate between the time difference for the packet reception (reception timestamp difference or **RxSD**) and transmission (transmission timestamp difference or **TxSD**). The time difference is expressed in 16MHz clock ticks: the clock frequency on which the Nordic Semiconductor nRF51822 operates.

We perform a test runs using six nodes that are all in communication range of each other. An initiator transmits in the first two slots of a round such that the other nodes can synchronise with the round. In the following slots each node transmits in its appointed slot while the other nodes can receive the packet. This helps us to answer the following questions:

- Can nodes keep in sync with the network during rounds?
- Is there a difference between the TxSD and RxSD?
- How accurate is the time-stamping mechanism?
- Does switching between receiving and transmitting affect the synchronisation performance?
- How does clock drift affect the synchronisation?

First Attempt: an Offset Affecting Synchronisation

We perform a test run containing 30 rounds with 40 slots in each round. Figure 4.9 shows the RxSD and TxSD for each slot during those 30 rounds. The different coloured lines represent the different nodes. The TxSD is constantly -1 across all nodes and during all slots and thus the lower graph shows only one coloured line that overlaps the others. According to the TxSD graph, the transmitting nodes send out the packets close to the deadline and hence, these nodes achieve a very accurate time synchronisation.

The upper graph shows the RxSD of the receiving nodes. This is the time difference the receiving nodes observe between the deadline of a slot and the timestamp of an incoming packet. The behaviour of the RxSD is similar in every node during each one of the 30 rounds. The RxSD takes on three distinct values as we will show in Table 4.1. We observe that the nodes have a RxSD of 0 clock ticks at the beginning of each round and thereafter the RxSD of each receiving node is much larger.

While comparing the graph of the TxSD with the graph of the RxSD, we find that there is a large difference between the TxSD that the transmitting nodes measure and the RxSD that the receiving nodes measure. This means that the transmitting node in a slot regards itself on time although the receiving nodes measure a difference between the goal and the actual timestamp of an incoming packet. According to the receiving nodes, there is no accurate time synchronisation even though the transmitting nodes believe there is. Because of this difference, we *do not achieve a consistently accurate time synchronisation* in the network and this will *affect the performance of Chaos BLE.*

Analysis and Solution

Due to the scale of Figure 4.9, it only shows the time synchronisation performance over the full test run. To pinpoint the cause of the difference between the RxSD and the TxSD, we analyse the log files of the nodes. The log files show a very specific RxSD and TxSD pattern. This pattern with these particular values occurred in every round and in every slot which we

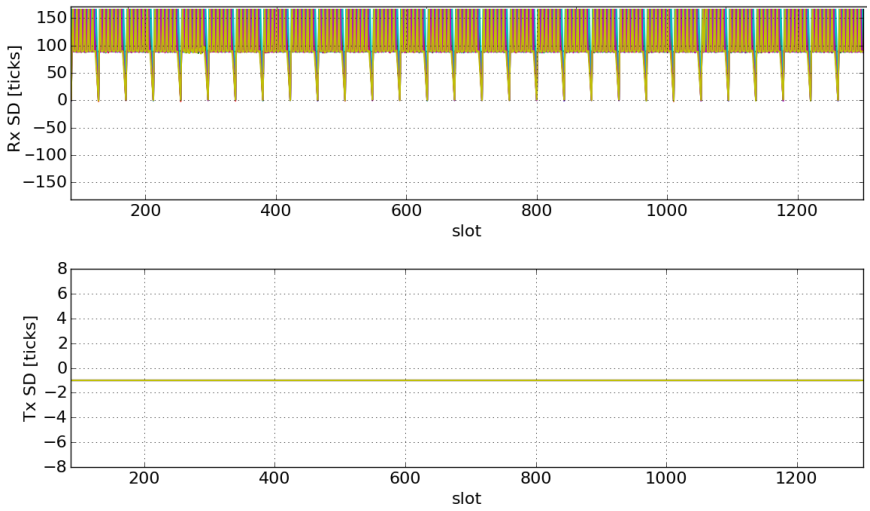


Figure 4.9: **The RxSD and TxSD in 16 MHz clock ticks for multiple rounds.** The receiving nodes perceive a RxSD while the transmitting nodes perceive a very small TxSD.

summarise in Table 4.1. The nodes that transmit in a particular slot measure a TxSD and the nodes that receive packets measure a RxSD.
Node 1 transmits in Slot X and observes a TxSD of 0 clock ticks. The other nodes receive the packet from node 1 and they all observe a RxSD of 89 ticks.

In the following slot, node 2 transmits and the other nodes receive the packet. Nodes 3 to 6 again observe a RxSD of 89 ticks while node 1 measures a RxSD of 178 ticks. The slot thereafter shows that node 3 transmit with a TxSD of 0 while node 1 and 4 to 6 have a RxSD of 89. We observe that node 3, which transmitted in the previous slot, measures a RxSD of 178 ticks in the slot directly after its transmission. We account this doubling in ticks to the deadline adjustment of the time synchronisation mechanism we discussed in Section 4.1.3.

Due to this deadline adjustment on the receiver side, the next deadline gets shifted 89 clock ticks later in time. However, this shift does not happen at the transmitting node. This results in a RxSD of 178 ticks for the node that transmitted in the previous slot.

We now have appointed the reason for the jump from 89 ticks to 178 ticks to the deadline adjustment method of the time synchronisation. This still does not explain the 89 tick difference between the RxSD and the TxSD. We suspect we can find the reason for this offset in the internal architecture of the radio. The time synchronisation mechanisms of Chaos BLE generate time-stamps based on the `address decoded event` of the radio (see Sec-

Table 4.1: **Measured RxSD and TxSD in ticks for an arbitrary set of slots.** Nodes adjust their next deadline based on the RxSD. This results on a doubled RxSD for the previously transmitting node that did not adjusted its deadline.

	Slot X		Slot X + 1		Slot X + 2		Slot X +3	
	TxSD	RxSD	TxSD	RxSD	TxSD	RxSD	TxSD	RxSD
Node 1	0	-	-	**178**	-	89	-	89
Node 2	-	89	0	-	-	**178**	-	89
Node 3	-	89	-	89	0	-	-	**178**
Node 4	-	89	-	89	-	89	0	-
Node 5	-	89	-	89	-	89	-	89
Node 6	-	89	-	89	-	89	-	89

tion 4.2.1). The radio most likely generates the `address decoded event` before it starts transmitting the actual packet.
The logs show that the 89 ticks difference between the RxSD and TxSD is constant. Our solution for this offset is to trick the transmitting node into sending the packet 89 ticks earlier. Hence, we compensate for the delay that the transmitting side of the radio hardware introduces.

Second Attempt: Validation of Offset Compensation Method

In order to validate the compensation method for the difference between RxSD and TxSD, we perform another longer test run with the same 6 nodes. Figure 4.10 shows the RxSD and the different colours denote the different nodes. We have not included the TxSD since it shows the exact same behaviour as in Figure 4.9. The TxSD is constantly -1 for every node and in every slot of the test run. We find that:

Finding 4. *We achieve accurate time synchronisation on every node in the network for many consecutive rounds and slots.*

We observe that receiving nodes now have a smaller RxSD between -3 and 5 clock ticks. The RxSD is between these two boundaries for every node, during every round and in each slot. Both the receiving nodes and the transmitting nodes now measure a small RxSD and TxSD. This means that the transmitting nodes meet the deadline of the slots and the receiving nodes receive the packets when they expect them. Switching from transmission to reception and vice-versa does not affect the RxSD or TxSD anymore. The remaining RxSD can be accounted to variations between the clocks of each node and multipath effects. The difference between the nodes is at a

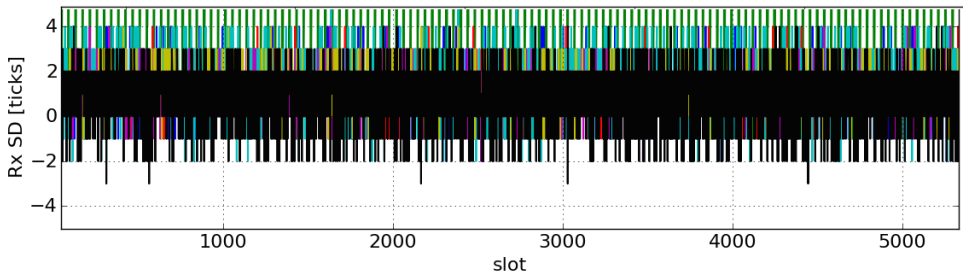


Figure 4.10: **The RxSD in 16 MHz clock ticks for multiple rounds after timing adjustment in the transmitter.** The RxSD of each node is low and constant for consecutive rounds.

maximum of 8 clock ticks in the 16MHz clock and thus we achieve the time synchronisation required for the capture effect.

Time Synchronisation without Deadline Adjustments

We used the time synchronisation mechanism with deadline adjustments in the evaluations above. The timestamp of the last successfully received packet determines the deadlines for the next slots. The receivers thus compensate for small delays caused by multipath or clock drift. We described this at the end of Section 4.1.3 and noted that this flexibility could cause that nodes, which are multiple hops apart from each other, end up with very different slot deadlines. Accordingly, in this section we evaluate the performance of the time synchronisation mechanisms with rigid deadlines for each slot. We perform a test run with six nodes, and we use rounds consisting of 100 slots with a duration of 6ms. Figure 4.11 shows the RxSD during the test run while the TxSD shows the same behaviour as in Figure 4.9. The different colours in Figure 4.11 denote the nodes in the network. We find that:

Finding 5. *Time synchronisation with rigid deadlines exposes the clock drifts of the nodes.*

In the beginning of each round, the nodes set their deadlines according to the first transmitted packet of the initiator. The nodes do not adjust their deadlines to the received packets during the consecutive slots in a round. The rigid deadlines per slot are based on the slot length in clock ticks of the 16MHz clock. However, different nodes have different clocks which will all have a small deviation. The same slot length defined in clock ticks will result in a different slot length in seconds for each node. The receiving nodes expect a packet at the deadline but they receive the packet before or after

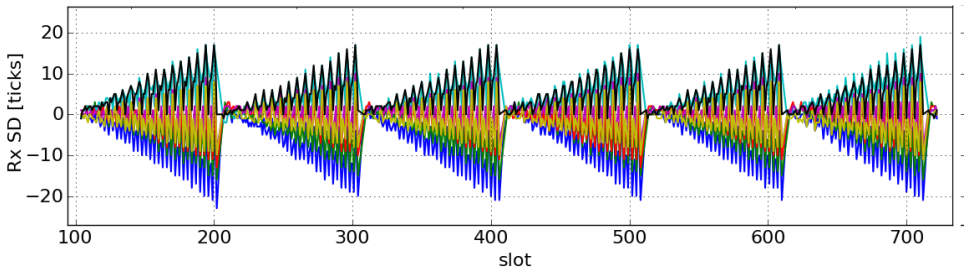


Figure 4.11: **The RxSD in** 16MHz **clock ticks for multiple rounds without adjustment of deadlines.** The RxSD of each nodes drift away from each other. They re-adjust at the beginning of each new round.

the deadline. The clock drift becomes more conspicuous over time since the clocks drift further apart.

Figure 4.11 shows that the deviation between two nodes has a maximum of 40 clock ticks per round of 0.6 seconds. The frequency tolerance for the 16 MHz crystal on which the clocks are based, is specified as $\pm 40ppm$ [17]. For the time period of 0.6 seconds, a maximum deviation of 768 clock ticks is tolerated and thus we can account these deviations to clock drift.

Discussion

We achieved a synchronisation between the nodes of less than 8 clock ticks of 16MHz with the time synchronisation mechanism that adjusts its deadlines. The time synchronisation without deadline adjustments achieves a synchronisation performance of 40 clock ticks or less. This results in an accuracy between $0.5\mu s$ and $2.5\mu s$. We showed in Figure 4.6 that a smaller delay results in a lower threshold required for the capture effect. The capture threshold is 7dB when the packets arrive simultaneously and it is 10dB if the stronger packet arrives within the $8\mu s$ of the preamble. While evaluating the overall Chaos BLE performance in Section 4.3.4, we will use the time synchronisation *with* deadline adjustments due to its increased accuracy.

4.3.4 Chaos BLE Performance

In this section we first show the network behaviour of the nodes in the network during a typical Chaos BLE round. Thereafter, we evaluate the impact on the key performance indicators, *reliability* and *latency* for different parameters. We show how different network sizes and transmission powers impact the performance. Finally, we show the impact of an interfered channel on the Chaos performance.

Methodology

We evaluate the Chaos BLE performance on the 15 node Bluetooth Low Energy testbed (see Appendix A). We use a Chaos application in which the nodes look to find the maximum value in the network and run this application for five minutes. A new Chaos round starts every two seconds and each round stops after a maximum of 120 slots. The network is considered static, i.e., nodes do not join or leave the network. At the beginning of a round, each node prepares a packet with its own flag set and its node ID as payload. The goal of a round is to make all nodes agree on the maximum node ID.

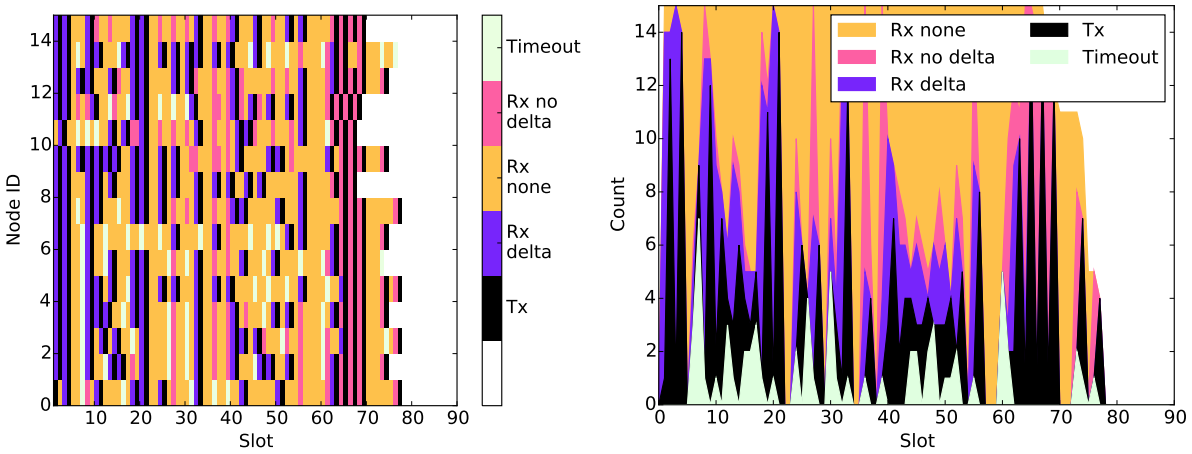
The performance indicators on which we evaluate Chaos BLE are *reliability* and *latency*. We define latency as the amount of slots it takes for all nodes to find the maximum value. The last node that finds this maximum thus determines the latency. We define reliability as the percentage of rounds during a run in which all the nodes complete and find the maximum value. If only *one node* does not complete in a *single round* and we run 100 rounds, this results in a reliability of 99%. This stringent definition of reliability is required since Chaos is designed for mission critical Cyber-Physical Systems.

Network Activity

Figure 4.12a shows the activity of each node per slot during a typical Chaos BLE round with 15 nodes. The different colours denote what the activity of a node in a particular slot is. We distinguish three different receiving activities: **Rx delta**, **Rx no delta** and **Rx none**. When a node correctly receives a packet in a slot and this packet contains different information than it has locally, we denote this with Rx delta. If the received information is the same then we denote this slot with Rx no delta. Rx none represents a collided packet or no packet reception at all. Additionally, we distinguish two transmission activities: **Tx** and **Timeout**. Tx is the transmission of a packet after a Rx delta in the previous slot. A timeout occurs when a node has not received a packet for some slots due to collisions. The timeouts prevent the Chaos round from terminating prematurely [12].

Node 1 is the initiator and starts the round by transmitting a packet in the first slot. Almost every node receives this packet due to the dense testbed setup. The nodes that receive the packet have learned new information and hence reply with a packet containing their own information merged with the new information. Many nodes transmit concurrently and this decreases the probability of a correct reception in the receiving nodes.

Figure 4.12b shows the cumulative activity of all the nodes in the network during this typical Chaos BLE round. It shows that the majority of



(a) Activity of individual nodes.

(b) Activity across all nodes.

Figure 4.12: **Activity of the nodes in a typical Chaos BLE round on the 15 node testbed**

the nodes has the same activity in the same slot and this results in the steep peaks around Slot 2, 4, 21, and between Slot 60 and 70. Due to the dense setup of the testbed, the nodes receive the same packet and reply concurrently which causes collisions at the receiving nodes. A timeout restarts the network operations which again causes many receiving nodes to reply in the next slot, e.g., in Slot 21.

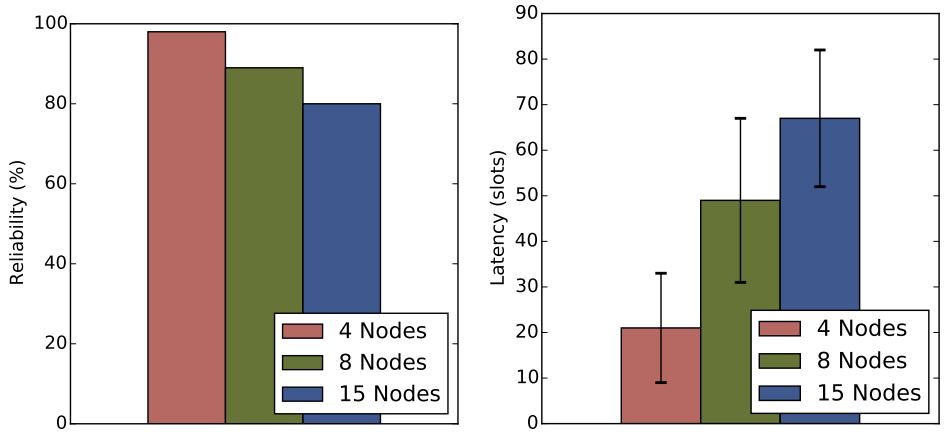
Moreover, we observe that towards the end of the round less nodes receive new information which is denoted by Rx delta. The rate of new information gradually decreases towards the end of a round. The nodes do not receive new information after Slot 65 and thus the nodes have completed. Each node retransmits the final answer five times to ensure that other nodes will complete as well.

Impact of Network Size

We evaluate the Chaos BLE performance for different network sizes. We use the 15 node testbed for this evaluation and use the first 4, the first 8 or all 15 nodes. Figure 4.13 shows the reliability and latency for a five minute test run with a Chaos round every two seconds. We find that:

Finding 6. *Smaller networks perform better in terms of reliability and latency.*

The reliability is higher and the latency is lower for smaller networks. This is expected since less nodes in a network means less information to exchange.



(a) Reliability against network size (b) Latency against network size

Figure 4.13: **Impact of network size.** Smaller networks have a lower latency as well as a higher reliability compared to larger networks.

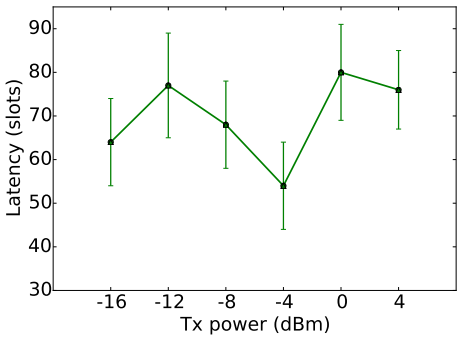
The nodes find the final answer faster, hence the decreased latency. As stated before, the reliability is defined as the percentage of rounds in which all nodes complete. This means that if a single node does not complete, the round is invalid. For smaller networks this means that there are less nodes that can impair the reliability and hence, the performance is improved.

Impact of Transmission Power

Figure 4.14 shows the impact of the transmission power on the reliability and latency of Chaos BLE in the 15 node testbed. The transmission power of a node increases the connectivity between nodes and thus can result in a lower average hop count and vice-versa. We calculate the reliability and latency over three 5 minute test runs, in which a Chaos round starts every 3 seconds. We find that:

Finding 7. *The transmission power has little impact on the reliability and the latency appears to be independent from it.*

The latency is variable but is on average 67 slots over the different transmission powers. The reliability increases on average for higher transmission powers. Differences exist between the runs and the reliability and latency give the appearance that they are independent from the transmission power. The Chaos IEEE 802.15.4 primitive benefits from spatial diversity and is designed for a large number of nodes [12]. The small BLE testbed we use, can not provide this. Additionally, the constraints for the capture effect are harsher in BLE. We account the highly variable results in Figure 4.14 to the



(a) Reliability against transmission power

(b) Latency against transmission power

Figure 4.14: **Impact of network properties.**

combination of these two difficulties. In Chapter 5 we present our solution to improve the performance of Chaos BLE and to make it less variable.

Impact of Channel Interference

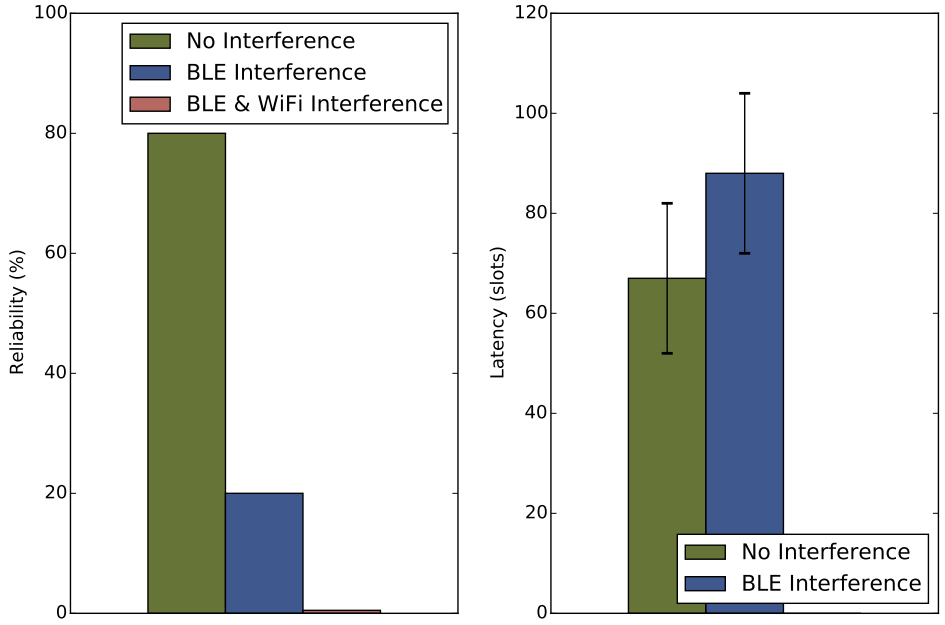
Bluetooth Low Energy is a radio standard that is used on a larger scale than IEEE 802.15.4. This increases the probability of interference by other BLE devices. Therefore, we evaluate how channel interference impacts the reliability and latency of Chaos BLE. We define three types of channels:

- **Good**: a channel without any major interferer.
- **Medium**: a channel on which three Bluetooth Low Energy nodes transmit a constant carrier wave.
- **Bad**: a channel with BLE interference and WiFi downloading.

We perform the evaluation on the 15 node testbed with 5 minute runs in which a Chaos round starts every two seconds. Figure 4.15a shows the reliability and latency for the three different channels. We find that:

Finding 8. *Channel Interference has a high impact on the Chaos BLE performance.*

The BLE constant carrier interference already impairs the reliability substantially. The rounds that are able to complete have a higher latency. Even if one node is affected by the BLE interference and can not complete, this results in a lower reliability due to our strict definition. During the test run with BLE and WiFi interference, some nodes could not synchronise with the network and therefore the rounds did not complete. This results in a reliability of 0%. We present our solution to the problem of channel interference in Chapter 5.



(a) Reliability against channel quality (b) Latency against channel quality

Figure 4.15: **Impact of channel quality.** Using a channel with interference results in unsynchronised nodes which makes none of the rounds completing.

Discussion

We ported Chaos on the popular nRF51 Bluetooth Low Energy platform. We characterised the capture effect on BLE and we achieved accurate time synchronisation between nodes in a network. We evaluated Chaos BLE on a BLE testbed. Chaos BLE achieves a *reliability of 80%* at an average *latency of 67 slots* for 15 nodes.

Due to the dense setup of the testbed, the average hop count in the network is low. Many nodes receive new information in the same slot and they will all reply in the next slot. As shown in Section 4.3.2, the probability of receiving a packet decreases rapidly as the number of concurrent transmitters increases. This impairs the operations of Chaos BLE and leads to a higher latency and a lower reliability. Moreover, we showed that channel interference has a high negative impact on the Chaos BLE performance.

We exposed new challenges in this chapter regarding too many concurrent transmitters and channel interference. In Chapter 5 we solve these challenges and we improve the performance of Chaos BLE.

Chapter 5

Chaos Multichannel

We enabled the Chaos primitive on Bluetooth Low Energy in Chapter 4. The evaluation of Chaos BLE showed that it does not accomplish performance that is comparable to the performance of the original Chaos networking primitive. This degradation has two specific causes:

Problem 1. *High density networks affect the Chaos BLE performance.*

In a dense network with little multi-hop behaviour there are on average too many concurrent transmitters. Section 4.3.2 shows that the packet reception ratio (PRR) decreases rapidly with more concurrent transmitters. The probability of the capture becomes less and this is not desirable for the operations of Chaos. Therefore, the number of concurrent transmitters needs to be decreased in order to have a better flow and to increase the Chaos performance.

Problem 2. *Channel interference impacts the Chaos BLE performance.*

We showed in Figure 4.15 that channel interference negatively impacts the overall Chaos BLE performance. We need to make Chaos BLE more resilient against interference since real world scenarios with more devices are likely to cause interference.

Solution. *Use multiple channels for the operation of Chaos BLE.*

We propose to make Chaos go multichannel during rounds in order to limit the number of transmitters on a single channel. Additionally, Chaos becomes less dependent on a single channel that may be interfered. We explain the multichannel design in Section 5.1 and we show implementation details in Section 5.2. We present the results in Section 5.3 and we discuss our results in Section 5.4.

5.1 Design

Dense networks such as our deployed testbed (see Appendix A) can have a clique-like topology: each node is in communication range of every other node. During a Chaos round, this means that many nodes will learn new information in a slot and then all reply in the next slot. The channel gets congested and no packet can be correctly decoded by the few nodes that are listening.

We could limit the number of concurrent transmissions by adopting a different transmission policy. The current transmission policy of Chaos makes a node solely transmit if there is a difference between the received information and the local information. A new policy could be to only allow a node to transmit when this difference is larger than a certain threshold. However, this could stall the network unnecessarily if nodes that have new information do not spread this further. We require a different solution that does not limit nodes from transmitting new information but that does limit the number of concurrent transmitters. Therefore, we propose Chaos to go *multichannel*.

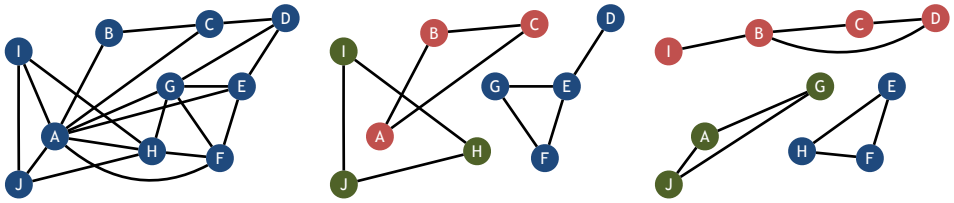
Section 4.3.4 shows that a higher reliability and a lower latency can be achieved in smaller networks. We propose to *divide* the larger network *into sub-groups* which then can benefit from the smaller size. Each group operates on a different channel such that it does not interfere with the other groups. However, the group composition should not be fixed during a round. Fixed compositions result in nodes finding their local group's answer but not the global answer based on all nodes in the network. Hence, we need to create group diversification such that the network can still operate in an all-to-all way.

Each group operates on a different channel and this results in frequency diversification. The network does not rely on a single channel for its operations, thus one bad channel will have less impact. This makes Chaos more resilient against interference on a single channel. We discuss Chaos Multichannel in Section 5.1.1.

Whenever the chosen subset of channels contains one or more interfered channels, this can still result in degraded performance. To overcome this, we propose to vary the assigned channel of each group in every slot of a Chaos round. Section 5.1.2 shows our solution that combines Chaos Multichannel and channel hopping.

5.1.1 Chaos Multichannel

We show our design for Chaos Multichannel in this section. We first discuss the basic operation, then we explain the group selection method and finally we discuss the relation between BLE channels and groups.



(a) **Slot 1:** bootstrapping with initiating node A

(b) **Slot n:** random group selection

(c) **Slot n + 1:** another random group selection

Figure 5.1: **A topology overview for different slots with the Chaos Multichannel primitive.**

Basic Operation

Figure 5.1 shows the basic operation of Chaos Multichannel. All the nodes in the network use a main channel for bootstrapping and synchronising (see Figure 5.1a). The initiator starts a round and all other nodes listen for a packet on this main channel. At the end of the slot, each node decides to either stay on the main channel or join another group with an associated channel (see Figure 5.1b). The nodes that all picked the same group operate on the same associated channel, hence they can communicate with each other.

The key insight to overcome the problem of nodes that solely find their local group answer, is to make *the decision of changing groups random*. The correlation between group compositions in consecutive slots will be low and thus nodes can learn new information. The random decision happens at the end of each slot until the round ends (see Figure 5.1c). Every node will switch back to the main group at the end of a round in order to overhear the start of a new round.

Group Selection

Dividing the network in smaller groups limits the number of possible concurrent transmitters and this benefits the PRR (see Section 4.3.2). The selection of a group happens randomly such that the group compositions are different in each slot. This allows nodes to learn new information in each slot while nodes can also spread new information to other nodes. Furthermore, the number of groups must be limited such that nodes do not end up unaccompanied in a group. In order to optimise the performance of Chaos Multichannel, the number of groups should be defined with the network size and density in mind. However, in Section 5.3.3 we show that the conservative approach of two multichannel groups already achieves a significant improvement in performance.

A larger network requires more groups such that the number of group members and thus concurrent transmitters is limited. In a large network

with a low density, the expected group sizes can be larger. Nodes may not be in communication range of each other and hence, more nodes can use the same channel and reside in the same group.

Channel Selection

Each group is associated with a single channel in the basic Chaos Multichannel approach. Nodes that operate on the same channel, belong inherently to the same group and vice versa. Theoretically, the groups could be associated to channels that are adjacent to each other. However, wide-band interference will interfere multiple channels and thus multiple groups. Accordingly, we introduce a spacing between channels to avoid this.

$$ch_{proposed} = ch_{main} + group \cdot \Delta ch \tag{5.1}$$

$$ch = ch_{\text{proposed}} - ch_{max} \left\lfloor \frac{ch_{proposed}}{ch_{\text{max}}} \right\rfloor \tag{5.2}$$

Equation (5.1) and Equation (5.2) show the relation between the group and the BLE channel. The main channel on which the nodes bootstrap and synchronise with the network is ch_{main}. The channels that belong to each group have a spacing between them and is denoted by Δch. Bluetooth Low Energy is limited to 40 channels which is denoted by ch_{max}. The group number is in the range of $[0 \ ... \ groups_{max} - 1]$.

5.1.2 Chaos Multichannel with Channel Hopping

We showed in Section 4.3.4 that channel interference has a negative impact on the performance of Chaos BLE. Chaos Multichannel can solve this problem by using multiple channels and thus increasing the frequency diversity. However, if interference free channels were picked initially, this does not guarantee that the channels will remain of good quality. An interferer can come within range of the network or the channel could degrade over time due to multipath fading [23]. A technique that is widely used to decrease the probability of operating on a bad channel is channel hopping [24]. This effectively decreases the duration for which the network operates on an interfered channel. It requires the nodes in a network to be time synchronised such that the nodes can all switch at the same time to a different channel.

We propose to incorporate channel hopping to Chaos Multichannel to increase the frequency diversity even more. Instead of using a subset of the Bluetooth Low Energy channels, our multichannel approach uses all 40 BLE channels. Channel hopping limits the time for which a group operates on a single channel and thus it decreases the probability of being affected by an interfered channel [24].

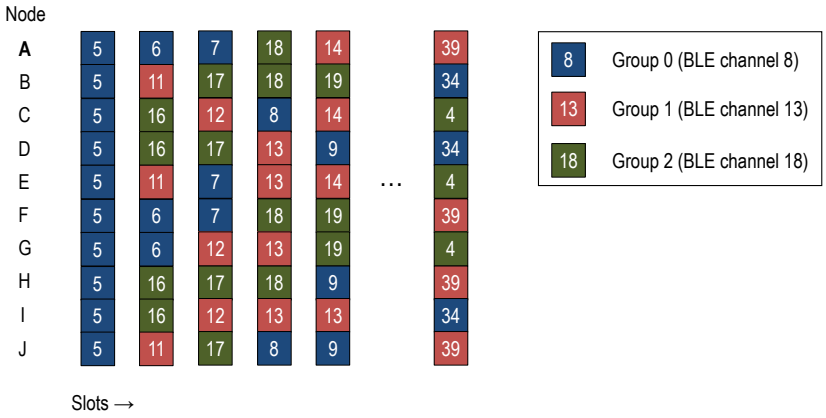


Figure 5.2: **Schematic slot layout for Chaos Multichannel with Channel Hopping.**
The channel depends on the chosen group and the current slot and round.

Basic Operation

We incorporate the same group selection procedure as described in Section 5.1.1. In each slot, the nodes randomly decide in which group they will communicate. The difference lays in the fact that the associated channels of each group are not fixed. Equation (5.3) and Equation (5.2) describe the relation between the groups and BLE channels.

$$ch_{proposed} = ch_{main} + slot + round + group \cdot \Delta ch \tag{5.3}$$

The associated channel is now time-dependent due to the incorporation of the slot and round number. The channel hopping mechanism itself is called blind channel hopping [24] and it uses all channels evenly. Figure 5.2 shows the schematic group and channel usage for a round. In the first slot, all the nodes use group 0 that operates on BLE channel 5 for that slot and round. In the next slot, all nodes randomly pick a group and switch to the associated channel. Even though node A, F, and G stay in group 0, they switch to another BLE channel since the slot number is changed. The group and channel switching continues for the remaining slots in the round. The nodes switch back to group 0 at the end of a round such that they can overhear the initiator at the start of the next round.

Bootstrapping

Synchronising all the nodes in the network can take up a long time. The node willing to join the network needs to listen for some time to pick up a signal on the channel they are listening to. Our Chaos Multichannel

approach increases the probability for such a node to overhear a packet. Chaos Multichannel uses multiple channels simultaneously in a slot, and this can decrease the waiting time. When nodes receive a packet, they will learn what the current round and slot numbers are. The nodes will then know what the next hopping pattern is and thus can operate in the consecutive slots and rounds.

5.2 Implementation

The previous section proposed the design of Chaos Multichannel. In this section we present the algorithms behind this multichannel primitive. First we show the Chaos Multichannel algorithm in Section 5.2.1. Thereafter we do this for Chaos Multichannel with Channel Hopping in Section 5.2.2

5.2.1 Chaos Multichannel Algorithm

The nodes in a network running Chaos Multichannel need to decide independently in which group they will operate during a slot. As we discussed in the previous section, the nodes randomly decide to operate in a particular group. We show in Algorithm 1 when a synchronised node chooses a group and the associated channel.

At the beginning of each round (line 6), a node operates in the default group and on the default channel (line 7 and 8) such that it can overhear the initiator in the first Chaos slot (line 11). After the first slot, the node picks a random group number out of the predefined set of groups in the range of [0 - GroupMaximum] (line 12). The node uses the `GroupToChannel` function to find the associated channel of the chosen group (line 13). The *DefaultChannel* and *ChannelSpacing* are predefined variables such that every node that chooses the same group, ends up on the same channel (line 2). *ChannelMaximum* denotes the maximum number of channels which is 40 for Bluetooth Low Energy.

The slot number increases at the end of every slot (line 14) until it reaches the maximum number of slots (line 10). The random selection of a group and channel happens after every slot until the round ends (line 16). The nodes now wait until a new round starts in which they will run this algorithm as well.

Algorithm 1 Chaos Multichannel Round

```
1: function GroupToChannel(group)
2:     channel ← (DefaultChannel + group * ChannelSpacing) mod ChannelMaximum
3: return channel
4: end function
5:
6: round(): ▷ Chaos round starting point
7: group ← DefaultGroup
8: channel ← DefaultChannel
9:
10: for slot ≤ slotMaximum do
11:     ChaosSlot() ▷ Transmission or reception
12:     group ← RandomGroup ▷ Range [0 - GroupMaximum]
13:     channel ← GroupToChannel(group)
14:     slot ← slot + 1
15: end for
16: end round
```

5.2.2 Chaos Multichannel with Channel Hopping Algorithm

Section 5.1.2 presented the Chaos Multichannel with Channel Hopping variant. Besides changing groups in a slot, the nodes will also change the associated channels per slot. The nodes that operate in the same group, use a different channel for every round and slot. We show in Algorithm 2 when the nodes pick a group and change channels. The algorithm for the Chaos Multichannel with Channel Hopping primitive is similar to Algorithm 1. We highlight the main differences between the two algorithms.

At the beginning of a round, all the nodes operate in the default group (line 9). However, the channel associated with this default group does now also depend on the slot and round number (line 1). The default channel may be interfered and therefore we change its associated channel every round and slot (line 10). At the end of a slot, the nodes randomly pick a group (line 14) and switch to the associated channel (line 15). We use blind channel hopping such that the nodes cycle evenly over all 40 BLE channels (line 2). The associated channel wraps around when the proposed channel is higher than the maximum number of channels.

Algorithm 2 Chaos Multichannel + Channel Hopping Round

```
1:  function GroupToChannel(group, slot, round)
2:      channel ← (DefaultChannel + round + slot +
        (group ∗ ChannelSpacing)) mod ChannelMaximum
3:  return channel
4:  end function
5:
6:  round():                                  ▷ Chaos round starting point
7:  slot ← 1
8:  round ← RoundNumber
9:  group ← DefaultGroup
10: channel ← GroupToChannel(DefaultGroup, slot, round)
11:
12: for slot ≤ slotMaximum do
13:     ChaosSlot()                           ▷ Transmission or reception
14:     group ← RandomGroup                   ▷ Range [0 - GroupMaximum]
15:     channel ← GroupToChannel(group, slot, round)
16:     slot ← slot + 1
17: end for
18: end round
```

In this section we showed when each node randomly picks a group with an associated channel. This associated channel depends in the case of the channel hopping primitive not only on the group number, but also on the current round and slot numbers. In Section 5.3 we evaluate the performance of Chaos Multichannel and Chaos Multichannel with Channel Hopping and we discuss these results in Section 5.4.

5.3 Evaluation

Section 5.3 structures the Chaos Multichannel evaluation into four parts. First, we discuss the network behaviour of the nodes in the network during a typical Chaos Multichannel round in Section 5.3.1. Thereafter, we evaluate the impact of the transmission power on the performance in a 15 node testbed in Section 5.3.2. Section 5.3.3 shows the impact of the group size for Chaos Multichannel in a 15 and 25 node testbed. Finally we have a look at the impact of interference on Chaos Multichannel and Chaos Multichannel with Channel Hopping in Section 5.3.4.

5.3.1 Network Activity

Figure 5.3a shows the activity of each node during a typical Chaos Multichannel round on the 25 node testbed (see Appendix A). We run the

maximum aggregation application and try to find the maximum node id in the network. The nodes operate with a transmission power of $-8dBm$ while using five groups with five associated channels. We have chosen the channels such that they are free from substantial WiFi interference.

Node 1 initiates the round in the first slot and every node in the testbed receives the information. The nodes switch randomly to one of the five channels in the next slot and transmit their merged information. Node 1 listens in Slot 2 because it transmitted in the previous one. In the Chaos BLE single channel variant, the 24 concurrently replying nodes would overcrowd the channel such that Node 1 can not correctly decode a packet. The separation of the nodes into groups results in less concurrent transmitters in a channel and hence, Node 1 receives a packet in Slot 2. Node 1 transmits its merged information in Slot 3 and Node 3, 7, 15, 20, and 23 receive the packet because they all have randomly chosen the same group. These nodes transmit in the next slot and they randomly pick a different group. The new information spreads through the network and through the different groups. This behaviour continues for the consecutive slots until the nodes have completed. The last node that learns new information is Node 18 in Slot 38. This means that all nodes found the maximum node id and hence, the latency for this round is 38 slots.

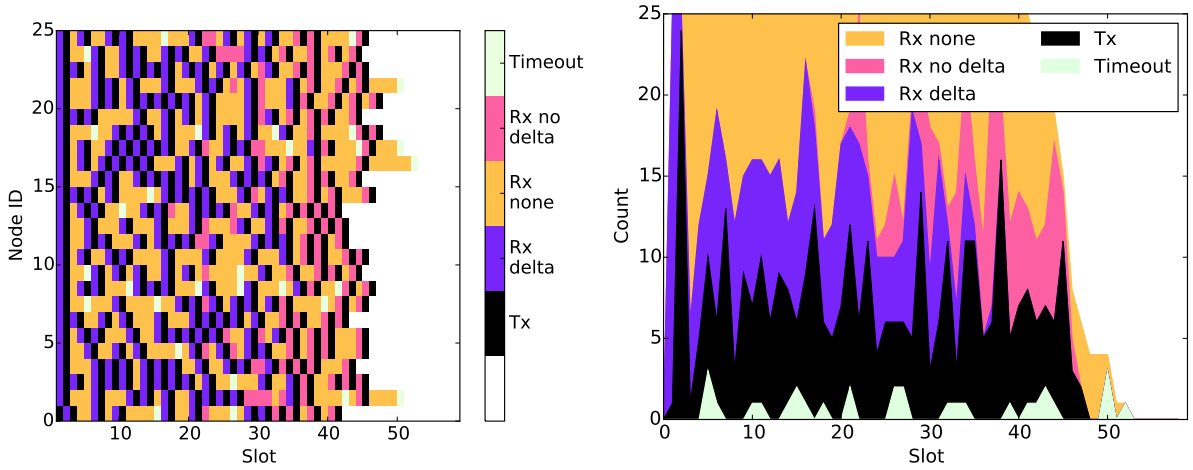
The cumulative activity in Figure 5.3b shows how the total network behaves during a round. Compared to the single channel approach in Figure 4.12b, the activity is more balanced across the nodes. There are only two slots, Slot 1 and 2, in which most nodes have the same activity. In Slot 1, all nodes except for the initiator receive new information and they all transmit in Slot 2. The maximum number of timeouts in a slot is now 3 in a 25 node testbed compared to 7 concurrent timeouts in the 15 node testbed (see Figure 4.12b). This means that the Chaos operation sustains better and does not need to be restarted by a timeout in order to continue.

5.3.2 Impact of Transmission Power

We evaluate Chaos Multichannel on the 15 node testbed for different transmission powers. Figure 5.4 shows a comparison between the single channel Chaos variant and Chaos Multichannel for different group sizes. We calculate the reliability and latency over three 5 minute test runs, in which a Chaos round starts every 2 seconds. We find that:

Finding 9. *Chaos Multichannel performs significantly better and more constant than the single channel variant.*

Figure 5.4a shows the reliability against the transmission power for different group sizes. Note that the y-axis ranges from 40% to 100%. The



(a) Activity of individual nodes.

(b) Activity across all nodes.

Figure 5.3: **Activity of the nodes in a Chaos Multichannel round.**

reliability is higher and less dependent on the transmission power. We observe a reliability between 80% and 100% for Chaos Multichannel while the single channel variant has a reliability between 40% and 85%.

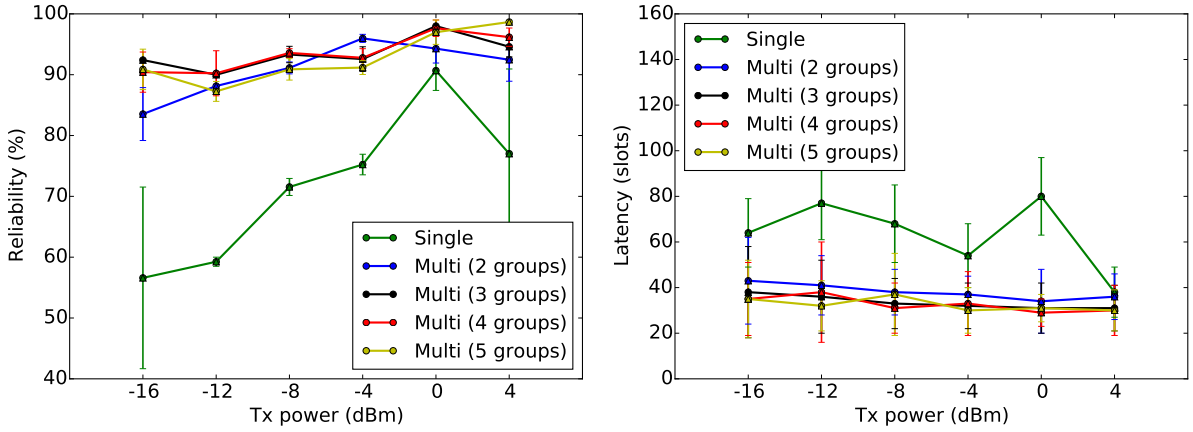
Figure 5.4b presents the latency against the transmission power for different group sizes. The latency is lower for each transmission power and it has less variability across the different transmission powers. The average latency is around 37 slots for different transmission powers and group sizes in the case of Chaos Multichannel. This is 67 slots for the Chaos single channel variant.

5.3.3 Impact of Group Size

In this section we further evaluate of the impact of the number of groups and different group sizes. We do this in the 15 node and 25 node testbed setup and we compare the different number of groups on reliability and latency. The nodes all transmit with a transmission power of $-8dBm$ and use channels free of substantial WiFi interference. We calculate the reliability and latency over three 5 minute test runs, in which a Chaos round starts every 2 seconds. We find that:

Finding 10. *Increasing the number of groups results in a higher reliability in the 15 node testbed, while the 25 node testbed has an optimal reliability at 4 to 6 groups.*

Figure 5.5a shows us the reliability agains the number of multichannel



(a) Reliability against transmission power

(b) Latency against transmission power

Figure 5.4: **Impact of the transmission power for different Chaos Multichannel group sizes in a 15 node testbed.**

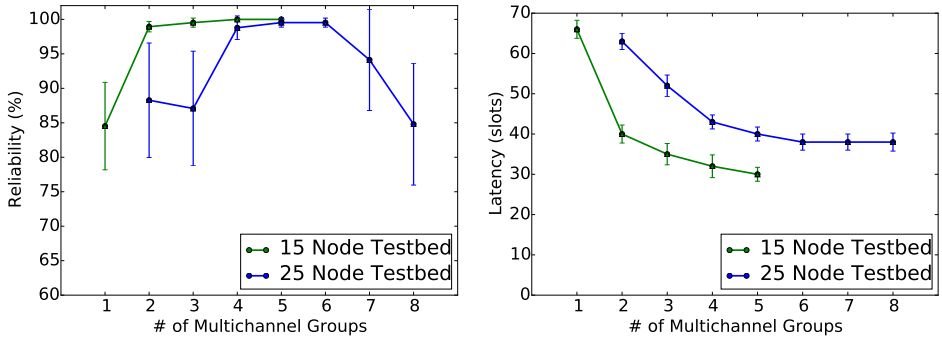
groups. Note that the y-axis ranges from 60% to 100%. We observe that the reliability of the single channel variant is on average 85%, while the introduction of Chaos Multichannel increases the reliability above 98% for 2, 3, 4 and 5 multichannel groups.

The reliability of Chaos Multichannel in the 25 node testbed does not solely increase with an increasing number of groups. Less than 4 groups results in more nodes on average per group and thus too many concurrent transmitters. More than 6 groups results in too little nodes on average per group and hence this results in nodes that can not complete the round. Hence, the reliability is optimal at 4 to 6 multichannel groups and Chaos Multichannel achieves a reliability higher than 98.5% for these group sizes. Figure 5.5b presents the latency against the number of multichannel groups. We find that:

Finding 11. *Increasing the number of groups results in lower latencies in both testbeds.*

The latency drops drastically when we introduce Chaos Multichannel with two groups in the 15 node testbed. Adding more groups decreases the latency further but it stabilises around 30 slots. In the case of the 25 node testbed, we observe that the latency decreases for each added group. The latency stabilises with more than 6 groups and settles around 38 slots.

If we look at both the reliability and latency for the two testbeds we find that the 15 node testbed has the best overall performance for 5 multichannel



(a) Reliability against number of groups

(b) Latency against number of groups

Figure 5.5: **Impact of the number of groups in the 15 and 25 node testbed.** The 15 node testbed shows the best overall results for 5 multichannel groups, the 25 node testbed shows this for 6 groups.

groups. The 25 node testbed operates at best with 6 multichannel groups. Hence, we conclude that the optimal group size with this density is between 4 and 7 nodes per group.

5.3.4 Impact of Channel Interference

Finally, we evaluate the impact of channel interference on the Chaos Multichannel primitive. We run the tests on the 25 node testbed for 5 minutes with a Chaos round every two seconds. The nodes operate with a transmission power of $-8dBm$. Chaos Multichannel uses BLE channel 11 as the main channel and channels 15, 19, 23, and 27 as the four sub-channels. Chaos Multichannel with Channel Hopping uses all 40 available BLE channels. We evaluate the Chaos Multichannel primitive in the presence of three types of interference:

- **Light**: interference on 3 sub-channels by BLE nodes which are spread out over the testbed.
- **Medium**: interference on all 5 channels by BLE nodes which are spread out over the testbed.
- **Heavy**: interference on all 5 channels by BLE nodes which are placed close to a single node.

BLE nodes generate the interference by transmitting a constant carrier onto each one of the interfered channels. The *heavy* interference is more obstructive since the interferers are concentrated around one node, which will suffer the most from the interference. A Chaos round can only complete when all nodes in a network complete. We define the reliability such that a Chaos

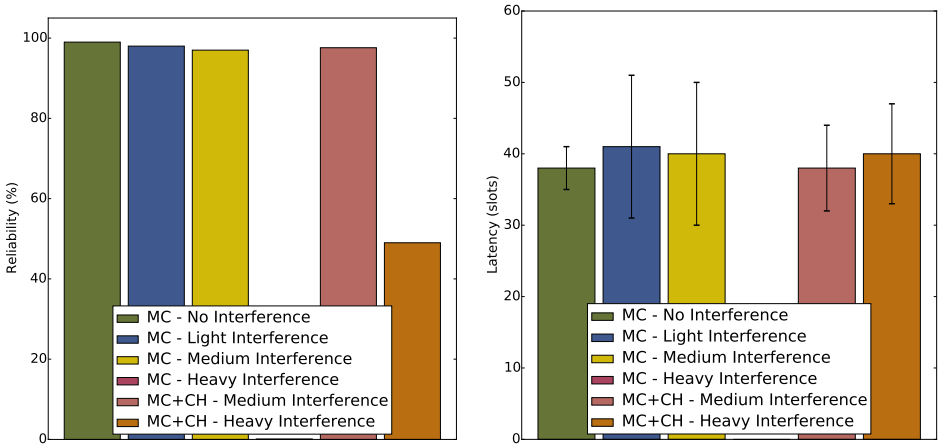
round is only valid if all nodes have the information of every other node. Hence, obstructing a single node impairs the overall performance of the network. Figure 5.6 shows the results and we find that:

Finding 12. *Light and medium interference have little impact on Chaos Multichannel while heavy interference results in impaired performance.*

Figure 5.6a shows the reliability and Figure 5.6b presents the latency for six different situations. We use the Chaos Multichannel performance from Section 5.3.3 as the base case with no interference. Light interference results in a slightly lower reliability and a slightly higher latency for Chaos Multichannel. However, the latencies in completed rounds vary more under light interference.

In the case of medium interference, the reliability is marginally lower while the latencies in the completed rounds still have a larger variation. Chaos Multichannel with Channel hopping has a comparable reliability under medium interference, while the latencies of the rounds are more constant.

Heavy interference has a significant impact on the performance of Chaos Multichannel. The obstructed node can not synchronise with the network and can not contribute to the maximum aggregation application. This results in a reliability of 0% and thus there are no completed rounds. Chaos Multichannel with Channel Hopping attenuates the impact of the heavy interference. Up to 50% of the rounds can complete with a latency that is comparable to Chaos Multichannel without interference. The reliability is lower due to the fact that nodes under interference can lose synchronisation with the network. The nodes need to listen for some time to resynchronise again and this can take some rounds in which the network can not complete the maximum aggregation.



(a) Chaos Multichannel reliability under interference

(b) Chaos Multichannel latency under interference

Figure 5.6: **Impact of channel interference on Chaos Multichannel.**

5.4 Discussion

The performance of Chaos BLE that we showed in Section 4.3.4 has room for improvement. Therefore, we proposed a solution that enables Chaos BLE on multiple parallel channels. By dividing the network into smaller groups, we limit the number of concurrent transmitters and we can increase the average PRR. Chaos Multichannel makes each node randomly pick one of the channels in every slot such that the group compositions change constantly. We added channel hopping in order to increase the frequency diversity even further and to suffer less from channel interference.

The evaluation shows that Chaos Multichannel outperforms the single channel variant in terms of reliability and latency. While running Chaos Multichannel on the 25 node testbed, we observe that the network finds a consensus within 38 slots. It costs the single channel variant 67 slots to find a consensus between 15 nodes.

We show that Chaos Multichannel is more resilient against interference due to the increased channel diversity. However, Chaos Multichannel can not operate under heavy interference. Adding channel hopping to Chaos Multichannel ensures that almost 50% of the rounds can complete at a latency of 40 slots.

Chapter 6

Future Work

In this section we propose future work regarding Chaos Multichannel and Chaos Multichannel with Channel Hopping. We suggest a way to make Chaos Multichannel aware of the density of the network in Section 6.1. Section 6.2 discusses the possibility of making the channel hopping adaptive.

6.1 Density Detection

The number of groups in Chaos Multichannel currently must be set with the network properties in mind. However, in a real world situation, the topology may changes due to mobility of the nodes and hence, a fixed parameter setting would not be optimal. Furthermore, density is not homogenous in most WSN deployments and depends on the location in the network. Chaos Multichannel can be adapted to support a variable number of groups.

The density can be detected in two ways: by the use of statistics during Chaos rounds or with a maintenance round once every few rounds. With the first method, each node keeps statistics of the number of different transmitters it received packets from in one hop. However, this can take some rounds before the nodes have an accurate view on the density. The advantage is that the information can be obtained without extra overhead.

The maintenance round method gives every node in the network an appointed slot in which they can transmit. The nodes that receive the packet correctly can add the transmitting node to their neighbour list and thus detect the local density. A disadvantage is that this method adds overhead and that the maintenance round should be scheduled regularly to have an accurate density detection.

Subsequently, the information from the density detection mechanism can help to make a decision on the maximum number of groups in the network. This maximum could be set to the same value for each node in the network or variable according to the node density. Figure 6.1 shows what a topology

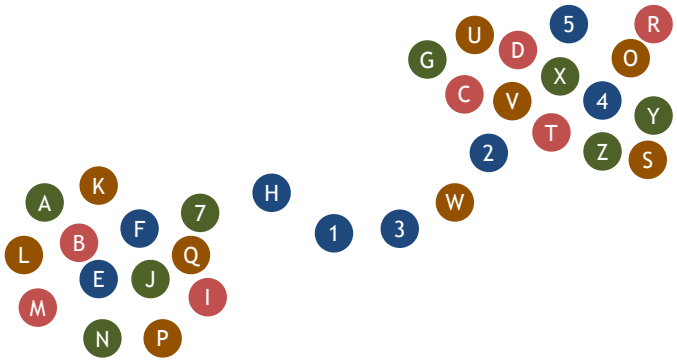


Figure 6.1: **Topology overview for Chaos Multichannel with different group sizes.**
Nodes that are in a denser part of the network should have more group diversity than nodes in a less denser part of the network.

overview looks like when density detection is added. The nodes in the denser areas have set their maximum number of groups higher such that there is more group diversity in a small area. The nodes in the middle have a low density and will use only one or two groups in order not to lose connectivity with the network.

6.2 Adaptive Channel Hopping

The current Chaos Multichannel with Channel Hopping implementation makes use of blind hopping. This means that all 40 BLE are evenly used to decrease the probability of operating on an interfered channel. However, systematically interfered channels should be avoided. Watteyne et al. show that whitelisting of channels on a link-by-link basis improves the estimated transmission count and network churn [24]. Since Chaos does not do any routing, whitelisting on a link-by-link basis is not suitable. Meanwhile, whitelisting locally is relevant because channel interference could also appear in a part of the network.

Bad channels can be detected by keeping statistics during the Chaos rounds. Channels that have a far worse packet reception ratio (PRR) compared to other channels should be blacklisted. A moving average could account for the temporal properties of the interference.

After the good and bad channels are found, the nodes in the network must agree onto which channels they will operate in the consecutive rounds. The Chaos network can then use its own mechanisms to make all nodes agree on the channels that should be used.

Chapter 7

Conclusions

The Chaos communication primitive does not follow the traditional approach on networking. It leverages the capture effect and concurrent transmissions to enable all-to-all communication and in-network processing in an efficient and reliable way. Chaos is currently implemented on the IEEE 802.15.4 radio standard. This radio standard is used more in science and industry, but less in everyday devices. We want to bring Chaos to the real world and therefore, evaluate the feasibility of Chaos on Bluetooth Low Energy

We did this by first characterising one of the foundations of Chaos, the capture effect on BLE. We ported Chaos onto a commonly used Bluetooth Low Energy platform and we achieved a time synchronisation of less than $2.5\mu s$. This enables concurrent transmissions and increases the probability of capture. We build a Bluetooth Low Energy testbed and we evaluated Chaos BLE on it. A reliability of 80% at a 67 slot latency was achieved for 15 nodes. This is rather weak compared to the original Chaos implementation and we account this due to the fact that the capture characteristics are much harsher on BLE than on the IEEE 802.15.4 standard. Channel interference and too many concurrent transmitters result in impaired Chaos BLE performance on our testbed.

We proposed a multichannel approach for Chaos in order to divide the network into smaller groups and limit the number of concurrent transmitters. Our multichannel primitive increases the frequency diversity and lowers the probability of operating solely on an interfered channel. Furthermore, we added channel hopping to our Chaos Multichannel primitive to improve resilience against interference. The evaluation of Chaos Multichannel shows significant improvements in terms of reliability and latency. We achieve a reliability over 98% with a latency of 38 slots on a 25 node testbed. Future work regarding this thesis is to make the channel hopping adaptive and make Chaos Multichannel aware of the network density to improve performance.

Appendix A

Bluetooth LE Testbed

In order to perform the evaluation of Chaos BLE, we setup a testbed with 25 Bluetooth Low Energy (BLE) compatible nodes. We discuss the testbed in this section, which is structured as follows. Appendix A.1 discusses the used hardware, Appendix A.2 the software, and Appendix A.3 presents the setup of the testbed.

A.1 Hardware

The hardware we use for this testbed, are the Bluetooth Low Energy compliant RedBearLabs BLE Nano boards. The BLE SoC is a Nordic Semiconductor nRF51822 as we described in Section 4.1.1. The RedBearLabs BLE Nano boards contain an extra ARM Cortex M4 that makes the nodes programmable through USB. The ARM CMSIS-DAP protocol allows us to copy a binary file to the USB board which is flashed automatically on the nRF51. Each node is connected to a central server through USB extension

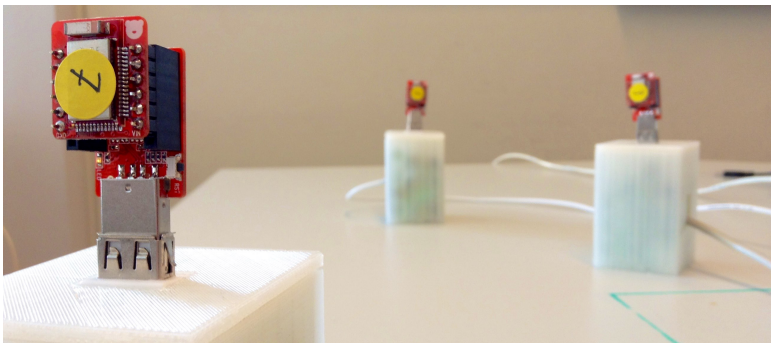


Figure A.1: **Closeup of the Bluetooth Low Energy hardware that we use for the evaluation of Chaos BLE.**

cords and USB hubs. The USB connection powers the devices and is used to log data onto the central server.

A.2 Software

The nodes are connected to a central server running Xubuntu 14.04. The server is responsible for uploading the binaries to the nodes and logging the data. We made it possible in the server that nodes are always connected to a specific device name by using the USB serialnumber. This allows us to program the binary files for particular nodes always onto the same node. Moreover, the UART must be connected to the same serial adapter to connect the log files to a specific node. We use the USB serialnumbers for this as well.

After compiling the Chaos BLE code to a binary, it needs to be modified before the server can upload it to a specific node. The binary does not yet incorporate a Node ID and hence, we make 15 or 25 different binaries with the specific node IDs using `tos-set-symbols`. This open-source software replaces a default value in the binary with the Node ID.

While performing a Chaos test run on the testbed, we gather logs through the UART connection between the nodes and the server. The central server saves these logs on its hard drive and we process them after the run, using a Python script. This script parses the logs and plots various graphs such as statistics on the time synchronisation and a timeline of the finished run.

A.3 Setup

Figure A.2 shows the schematic layout of our testbed. It is deployed in a small office space and nodes are placed on the wall, windows, and table. We use the nodes that are coloured red for the 15 node testbed setup. We add the 10 remaining black coloured nodes for the 25 node testbed.

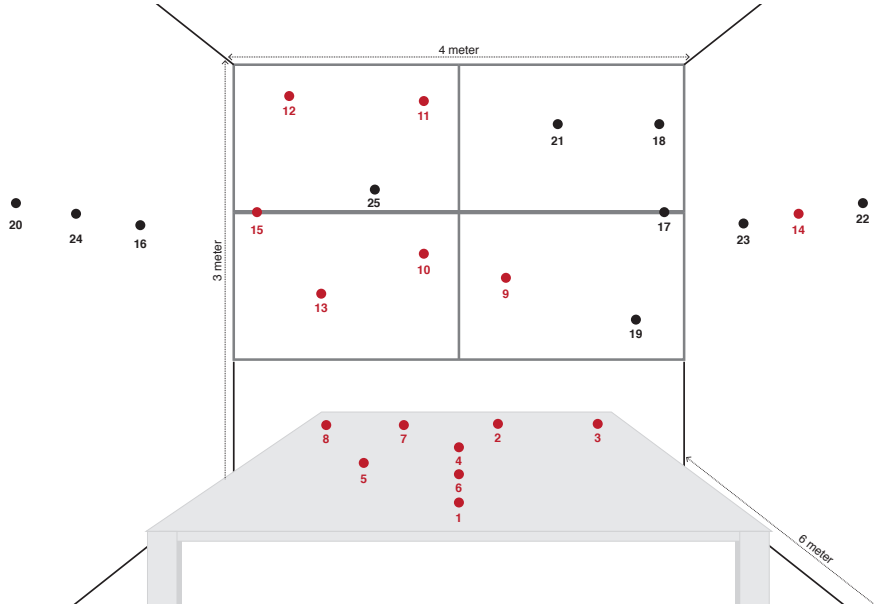


Figure A.2: **Schematic layout of the Bluetooth Low Energy testbed.** The red nodes constitute the 15 node testbed. The 25 node testbed combines the red nodes with the black nodes.

Appendix B

Glossary

List of Acronyms

BLE Bluetooth Low Energy

CPS Cyber-Physical Systems

DSSS direct-sequence spread spectrum

FHSS frequency-hopping spread spectrum

GFSK Gaussian frequency-shift keying

IEEE Institute of Electrical and Electronics Engineers

LWB Low-Power Wireless Bus

MiM message-in-message

NOPs *no operations*

nRF51 Nordic Semiconductor nRF51822

O-QPSK off-set quadrature phase-shift keying

PRR packet reception ratio

RSSI received signal strength indicator

SFD start of frame delimiter

SNR signal-to-noise ratio

SoC System on a Chip

TI Texas Instruments

WSN wireless sensor networks

Bibliography

[1] Beshr Al Nahas, Simon Duquennoy, Venkatraman Iyer, and Thiemo Voigt. Low-power listening goes multi-channel. In *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*, pages 2–9. IEEE, 2014.

[2] Bluetooth SIG. Bluetooth core specification version 4.0. *Specification of the Bluetooth System*, 2010.

[3] Behnam Dezfouli, Marjan Radi, Kamin Whitehouse, Shukor Abd Razak, and Hwee-Pink Tan. Cama: Efficient modeling of the capture effect for low-power wireless networks. *ACM Trans. Sen. Netw.*, 11(1):20:1–20:43, Aug. 2014.

[4] Prabal Dutta, Stephen Dawson-Haggerty, Yin Chen, Chieh-Jan Mike Liang, and Andreas Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM, 2010.

[5] Prabal Dutta, Razvan Musaloiu-e, Ion Stoica, and Andreas Terzis. Wireless ack collisions not considered harmful. In *Proceedings of the 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*, pages 1–6, 2008.

[6] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14. ACM, 2012.

[7] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 73–84, April 2011.

[8] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[9] Zonglin Guo, Ian G Harris, Lih-feng Tsaur, and Xianbo Chen. An on-demand scatternet formation and multi-hop routing protocol for ble-based wireless sensor networks. In *Wireless Communications and Networking Conference (WCNC), 2015 IEEE*, pages 1590–1595. IEEE, 2015.

[10] Naresh Gupta. *Inside Bluetooth low energy.* Artech house, Norwood, MA, USA, 2013.

[11] Youngmin Kim, Hyojeong Shin, and Hojung Cha. Y-mac: An energy-efficient multi-channel mac protocol for dense wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, pages 53–63. IEEE Computer Society, 2008.

[12] Olaf Landsiedel, Federico Ferrari, and Marco Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 1. ACM, 2013.

[13] Joonsoo Lee, Young-myoung Kang, Suchul Lee, and Chong-kwon Kim. Opportunities of mim capture in ieee 802.11 wlans: analytic study. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, page 66. ACM, 2011.

[14] K. Leentvaar and J. Flint. The capture effect in fm receivers. *Communications, IEEE Transactions on*, 24(5):531–539, May 1976.

[15] Jiun-Ren Lin, Timothy Talty, and Ozan Tonguz. On the potential of bluetooth low energy technology for vehicular applications. *Communications Magazine, IEEE*, 53(1):267–275, 2015.

[16] Jiakang Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *INFOCOM 2009, IEEE*, pages 2491–2499, April 2009.

[17] Nordic Semiconductor. nrf51822 product specification 3.1, 2014.

[18] Naveen Santhapuri, Justin Manweiler, Souvik Sen, Romit Roy Choudhury, Srihari Nelakuduti, and Kamesh Munagala. Message in message (mim): A case for reordering transmissions in wireless networks. In *Seventh ACM Workshop on Hot Topics in Networks*, 2008.

[19] Wilson So, Jean Walrand, Jeonghoon Mo, et al. Mcmac: A parallel rendezvous multi-channel mac protocol. In *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE*, pages 334–339. IEEE, 2007.

[20] Andrew Tinka, Thomas Watteyne, and Kris Pister. A decentralized scheduling algorithm for time synchronized channel hopping. In *Ad Hoc Networks*, pages 201–216. Springer, 2010.

[21] Joost Van Velzen and Marco Zuniga. Let's collide to localize: Achieving indoor localization with packet collisions. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 336–339. IEEE, 2013.

[22] Yin Wang, Yuan He, Xufei Mao, Yunhao Liu, and Xiang-Yang Li. Exploiting constructive interference for scalable flooding in wireless networks. *Networking, IEEE/ACM Transactions on*, 21(6):1880–1889, Dec 2013.

[23] Thomas Watteyne, Steven Lanzisera, Ankur Mehta, and Kristofer SJ Pister. Mitigating multipath fading through channel hopping in wireless sensor networks. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[24] Thomas Watteyne, Ankur Mehta, and Kris Pister. Reliability through frequency diversity: why channel hopping makes sense. In *Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 116–123. ACM, 2009.

[25] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pages 45–52, May 2005.

[26] Matthias Wilhelm, Vincent Lenders, and Jens B Schmitt. On the reception of concurrent transmissions in wireless sensor networks. *Wireless Communications, IEEE Transactions on*, 13(12):6756–6767, 2014.

[27] Dingwen Yuan and Matthias Hollick. Let's talk together: Understanding concurrent transmission in wireless sensor networks. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 219–227. IEEE, 2013.

[28] Dingwen Yuan, Michael Riecker, and Matthias Hollick. Making âĂŸglossyâĂŹnetworks sparkle: Exploiting concurrent transmissions for energy efficient, reliable, ultra-low latency communication in wireless control networks. In *Wireless Sensor Networks*, pages 133–149. Springer, 2014.