Diss. ETH Nr. 18209

Reducing Uncertainty in Wireless Sensor Networks

Network Inspection and Collision-Free Medium Access

A dissertation submitted to the
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
Matthias Ringwald
Diplom-Informatiker, University of Karlsruhe
born November 24, 1973
citizen of Germany

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Lothar Thiele, co-examiner

2009

Abstract

Wireless sensor networks consist of a large number of sensor nodes which communicate wirelessly. These nodes are small, battery-powered computing devices equipped with sensors to perceive their environment. Sensor networks are typically deployed in the nature; e.g., to observe the breeding of delicate bird species, to monitor changes of glaciers or to study seismic activities and earthquakes.

Several factors contribute to the fact that wireless sensor networks often do not work as expected when deployed in a real-world setting. Some of them are environmental influences which may lead to non-deterministic behavior of radio transmission, malfunction of the sensors, or even the complete failure of a sensor node. In addition to that, scarce resources and missing protection mechanisms on the sensor nodes may lead to program errors. Fixing these problems and errors at the deployment site is difficult, as the inherent characteristics of sensor networks - autonomous, distributed, and resource constrained - hardly allow to get insight into the inner workings of the network. Another common cause for network problems is an unforeseen high traffic load, a so-called *traffic burst*. Such a traffic burst may occur for example, when a sensor network observes a physical phenomenon, and all nodes in the vicinity will try to report at once, causing the occurrence of packet collisions combined with a high packet loss. All these factors contribute to the uncertainty of the sensor network behavior and function.

To help with the detection of faults in a deployed network, various active inspection tools have been developed where the sensor nodes are instrumented to actively collect statistical data of their state and communication behavior and forward this data over the radio network to a central station. This additional communication, however, not only increases energy consumption but also changes the nodes' behavior. A further limitation is the fact that no information can be gathered from partitioned parts of the network.

With respect to communication, current medium access control protocols use a probabilistic approach to handle concurrent send requests. Al-

though this allows for efficient energy use in the case of infrequent and sporadic traffic, which is typical for wireless sensor networks, probabilistic protocols perform poorly in the presence of synchronous send requests of neighboring nodes.

The goal of this thesis is to facilitate the deployment of wireless sensor networks by reducing the uncertainty about the behavior of the network. In order to reduce this uncertainty, we address two distinct areas, namely *fault detection* and *fault prevention*. To detect faults in the sensor node software, we provide tools for the efficient real-time inspection of deployed networks that overcome the problems of the active inspection approach. To prevent faults caused by traffic bursts, we propose medium access protocols that avoid collisions and handle traffic bursts in a deterministic manner.

With respect to the outlined problems, the contribution of this thesis is threefold. Firstly, we present a *survey* on problems reported during actual deployments and provide a classification of these. Secondly, to address the problem of the active inspection approach, we propose and evaluate the concept of *passive inspection* of sensor networks. Finally, we propose and evaluate a new *collision-free medium access control protocol* capable of handling bursty traffic.

In particular, we show that passive observation of the radio traffic facilitates the development of wireless sensor network applications and improves their deployment. We elaborate on this by supplying appropriate tools which can be adapted for different wireless sensor network applications with low configuration effort. Concretely, we present the *Sensor Network Inspection Framework* (SNIF) which allows to detect faults in a deployed wireless sensor network using passive observation and online analysis.

To tackle problems related to traffic bursts, we evaluate different possibilities for collision-free communication in sensor networks and present new techniques for efficient group communication which leverage the broadcast nature of radio communication. By combining existing and new concepts, we realize a collision-free and energy-efficient protocol (*BurstMAC*) that induces only a low overhead compared to current probabilistic state-of-the-art protocols but can handle traffic bursts efficiently.

Kurzfassung

Sensornetze bestehen aus einer grossen Anzahl miteinander drahtlos kommunizierender Sensorknoten. Dabei handelt es sich um batteriebetriebene Kleinstcomputer mit Sensoren zur Wahrnehmung der Umgebung. Sensornetze werden häufig in die Umwelt ausgebracht; beispielsweise zur Beobachtung des Brutverhaltens gefährdeter Vogelarten, zur Beobachtung von Veränderungen bei Gletschern oder zur Studie von seismischen Aktivitäten und Erdbeben.

Verschiedene Faktoren führen dazu, dass Sensornetze oft nicht wie erwartet funktionieren, wenn sie in der realen Welt eingesetzt werden. Dazu gehört unter anderem die Vielzahl von Umwelteinflüssen, was zu nichtdeterministischem Verhalten der Funkschnittstelle, der Sensoren oder gar zum Ausfall von Knoten führen kann. Darüber hinaus tragen knappe Ressourcen und fehlende Schutzmechanismen auf den Sensorknoten zu häufigen Programmierfehlern bei. Eine anderere häufige Ursache für Probleme in Sensornetzen stellen unvorhergesehene, hohe Sendeaktivitäten, so genannte *Traffic Bursts*, dar. Solch ein Traffic Burst kann zum Beispiel auftreten, wenn ein Sensornetz ein physikalisches Phänomen beobachtet und alle Sensorknoten in der Nähe gleichzeitig versuchen dies zu melden, was zum Auftreten von Paketkollisionen in Kombination mit einem hohen Paketverlust führt. All diese Faktoren tragen zur Ungewissheit über das Verhalten und die Funktionalität des Sensornetzes bei.

Um die Fehlersuche in einem ausgebrachten Sensornetz zu vereinfachen, wurden verschiedene Inspektionswerkzeuge entwickelt, die einen aktiven Ansatz verfolgen. Bei diesen Werkzeugen sammeln Sensorknoten aktiv statistische Daten über ihren Zustand und ihr Kommunikationsverhalten und leiten diese dann über das Funknetz nach aussen. Durch die zusätzliche Kommunikation wird jedoch zum einen zusätzlich Energie verbraucht, zum anderen aber auch das Verhalten der Knoten selbst verändert. Eine weitere Einschränkung ist, dass keine Informationen aus isolierten Teilen des Netzes gewonnen werden können.

Für die Kommunikation nutzen aktuelle Medienzugriffsverfahren probabilistische Mechanismen, um den Sendewunsch benachbarter Knoten

zu regeln. Obwohl dies einen niedrigen Energieverbrauch bei der in Sensornetzen üblichen geringen und sporadischen Kommunikation erlaubt, erfüllen die probabilistischen Verfahren bei gleichzeitigem Sendewunsch mehrerer Knoten ihre Aufgabe nur unzureichend.

Das Ziel dieser Disseration ist, den Einsatz von Sensornetzen dadurch zu vereinfachen, dass die Ungewissheit über das Verhalten des Netzes reduziert wird. Hierzu verfolgen wir zwei unterschiedliche Herangehensweisen, und zwar einerseits *Fehlererkennung* und andererseits *Fehlervermeidung*. Um Fehler in der Software von Sensorknoten zu entdecken, stellen wir Werkzeuge zur effizienten Echtzeit-Inspektion ausgebrachter Sensornetze bereit, welche die Probleme des aktiven Ansatzes vermeiden. Hinsichtlich der Fehler, die durch Traffic Bursts verursacht werden, entwerfen wir Medienzugriffsverfahren, die Kollisionen vermeiden und Traffice Bursts auf deterministische Art und Weise abarbeiten.

Der Beitrag dieser Dissertation im Bezug auf die aufgeführten Probleme ist dreigeteilt. Als erstes stellen wir eine *Klassifikation* der Probleme vor, die sich beim Ausbringen von Sensornetzen ergeben. Zweitens schlagen wir das Konzept des *passiven Belauschens* von Sensornetzen vor und stellen hierzu ein lauffähiges und erweiterbares System bereit. Drittens präsentieren wir ein neues *kollisionsfreies Medienzugriffverfahren*, welches in der Lage ist, mit Traffic Bursts effizient umzugehen.

Insbesondere zeigen wir, dass das passive Belauschen der Kommunikation in einem Sensornetz das Enwickeln und das Ausbringen von Sensornetzen erleichtert und belegen dies durch das Bereitstellen entspechender Werkzeuge, die mit geringem Konfigurationsaufwand an unterschiedliche Sensornetzanwendungen angepasst werden können. Konkret stellen wir das *Sensor Network Inspection Framework* (SNIF) vor, welches es erlaubt Fehler in einem ausgebrachten Sensornetz durch passives Mithöhren und Online-Analyse zu entdecken.

Um Probleme durch Traffic Bursts zu vermeiden, evaluieren wir verschiedene Möglichkeiten zur kollisionsfreien Kommunikation in Sensornetzen und präsentieren neue Basistechniken für kollisionsfreie Gruppenkommunikation, die sich die Tatsache zu Nutze machen, das ausgesandte Funkwellen von allen Nachbarn empfangen werden. Durch eine Kombination existierender und neuer Ansätze realisieren wir ein kolllisionsfreies und energieeffizientes Protokoll (*BurstMAC*). Das neue Vefahren verursacht gegenüber existierenden probabilistischen Lösungen zwar geringe Mehrkosten, vermag dafür aber auch mit länger andauernden Traffic Bursts effizient umzugehen.

Acknowledgements

Thanks to all of my collaborators who provided technical, intellectual, practical, and advisorial help with various aspects of this work. These include but are not limited to Kay Römer, Oliver Kasten, Marc Langheinrich, Harald Vogt, Jan Beutel, Matthias Dyer, Philipp Blum, and, of course, my dissertation committee.

Special thanks to Mila for her constant support.

This work was partially supported by the Swiss National Science Foundation under grant number 5005-67322 (NCCR-MICS).

Contents

1. Introduction

Recent progress in the field of *wireless sensor networks* (WSN) has led to the deployment of sensor networks by scientists from different research areas, such as biology, seismology and agriculture. However, when sensor networks are deployed in the real world, they often do not work as expected for a number of different reasons. The goal of this thesis is to facilitate the deployment of sensor networks by providing new methods and tools to detect faults in deployed networks and new medium access protocols which prevent faults by avoiding typical collisions.

In this first chapter, we introduce the research area of wireless networks, motivate the need for passive inspection and collisions-free medium access, and give a brief overview of the main contributions of our work. We conclude the chapter with an overview of the remainder of this thesis.

1.1. Motivation

Wireless sensor networks consists of a large number of sensor nodes which communicate wirelessly. Sensor nodes are small, battery-powered computing devices equipped with sensors which are able to perceive their environment. They were first envisioned for military surveillance applications where a large number of nodes would be deployed in hostile environments to provide information about the other parties' movements without own risks. Later, sensor networks became a civil tool for environmental monitoring and even wireless indoor power and utility metering. While the cost of sensor installation stayed constant, the falling costs for processing power and wireless communication made wireless sensor networks more feasible and more attractive. In addition, ready-to-use example applications and better commercial packaging of sensor nodes made this technology available to scientists from different research areas, e.g. in biology [61], seismology [86], and agriculture [13].

However, wireless sensor networks often do not work as expected when deployed in the real world. Environmental influences may lead to non-deterministic behavior of radio communication, the sensors, or even the

complete failure of a sensing node. In addition to that, scarce resources and missing protection mechanisms on the sensor nodes often lead to program errors which makes them hard to debug.

To get an insight into the inner workings of a sensor network, several tools have been developed, such as Sympathy [63] or Memento [70]. In these tools, sensor nodes collect statistical data of their state and communication behavior and forward this data over the radio network to a central station. The additional communication, however, not only increases energy consumption but may also involuntarily change the nodes' behavior. A further limitation is the fact that no information can be gathered from isolated parts of the network. Therefore, tools which provide an insight without affecting and disturbing wireless network, are an important tool to facilitate real-world deployments.

A typical communication problem is the sudden occurrence of packet collisions combined with a high packet loss. This might be triggered by a sudden high traffic load, a *traffic burst*. Such a traffic burst may occur if it is the task of a sensor network to detect a certain physical phenomenon. When the phenomenon is observed, all nodes in the vicinity will try to report at once. Current low power media access protocols which are optimized for infrequent and sporadic traffic, use probabilistic protocols which perform poorly in the presence of correlated send requests of neighboring nodes. Thus, collision-free medium access protocols with similar energy efficiency are a promising approach to allow collision-free traffic bursts.

1.2. Contributions

The goal of this thesis is to facilitate the deployment of wireless sensor networks by reducing the uncertainty about the behavior of the network. In order to reduce uncertainty both in already deployed networks and in new deployments, we address two distinct issues, namely *fault detection* and *fault prevention*. To detect faults in the sensor node software, we aim to provide tools for the efficient and real-time inspection of deployed networks that overcome the problems of the active inspection approach. To prevent faults caused by traffic bursts, we aim to design medium access protocols that avoid collisions and handle traffic bursts in a deterministic manner. This thesis provides three main contributions:

1. Classification of deployment problems: As a first contribution, we

provide a classification of faults experienced in sensor networks based on various reports.

2. Passive Inspection: We show that passive observation of the radio traffic facilitates the development of wireless sensor network applications in general and particularly simplifies their deployment. We justify this by supplying appropriate tools which can be adapted for different WSN applications with little configuration effort. Concretely, we present the *Sensor Network Inspection Framework* (SNIF) which consist of several components: (1) a wireless sniffing network, (2) a packet description language, (3) a framework for data stream processing with sensor network specific operators, and (4) a graphical user interface for the observation of the network.

3. Collision-free medium-access: Based on our findings on common error causes in sensor networks, we present a new media access protocol which shows only a low overhead compared to current (probabilistic) state-of-the-art protocols but can handle traffic bursts efficiently. For this, we evaluate different possibilies for collision-free communication in sensor networks. By combining existing and new concepts, we are able to achieve energy-efficient and collision-free communication in our *BurstMAC protocol*.

1.3. Structure

This thesis first discusses general aspects of wireless sensor networks, before providing an analysis of WSN deployments and their problems. To address these problems, we propose, implement, and evaluate passive inspection of WSNs and collision-free medium access. In more detail, the thesis is structured as follows:

Chapter 3 introduces the concept of passive inspection based on live traces of network messages from WSNs and discusses its challenges. We further present passive indicators that allow to detect most of the common problems in WSN deployed so far. We present and discuss related work for network monitoring and WSN inspection.

Chapter 4 is devoted to our Sensor Network Inspection Framework (SNIF), an instance of the passive inspection approach. We first present the general architecture and explain how we realize a flexible network sniffer. The online analysis of the captured network traces is performed by a data-stream processor. We demonstrate WSN-specific operators and

show how these can be combined to detect problems in a deployed WSN and inspect the state of individual nodes via a graphical use interface. We evaluate SNIF in a simulation environment in terms of accuracy and latency.

Chapter 5 provides an overview on WSN medium-access protocols with an emphasis on energy-efficiency and mechanism for collision-free communication. We describe our new physical layer techniques that are based on concurrent transmissions over a shared radio channels. Based on these techniques, we develop an efficient protocol for single-hop networks.

In chapter 6, we present a MAC-protocol based on the new integer operation from the previous chapter to form a spanning-tree multi-hop network. The focus in this protocol lies on efficient network-startup and coloring. We evaluate the protocol in terms of required time synchronization, feasible network density, and energy consumption.

Based on the MAC protocol from chapter 6, in chapter 7, we relax the assumptions and present a MAC protocol which works on common platforms. We provide a detailed protocol overview and describe the implementation. An evaluation in a lab setting confirms its robustness and energy-efficiency and provides insights into further optimization options.

Chapter 8 concludes this thesis by summarizing the results, by discussing limitations, and by providing an outlook on future work.

2. Wireless Sensor Network Deployments

Sensor networks offer the ability to monitor real-world phenomena in detail and at large scale by embedding a wireless network of sensor nodes into the real world. Here, *deployment* is concerned with setting up an operational sensor network in a real-world environment. In many cases, deployment is a labor-intensive and cumbersome task as real-world influences trigger bugs or degrade performance in a way that has not been observed during pre-deployment testing in the lab. The reason for this is that the real world has a strong influence on the function of a sensor network by controlling the output of sensors, by influencing the existence and quality of wireless communication links, and by putting physical strain on sensor nodes. These influences can only be very rudimentarily modeled in simulators and lab testbeds.

Information on the typical problems encountered during a deployment is rare. We can only speculate on the reason for this. On the one hand, a paper which only describes what happened during a deployment seldom constitutes novel research and might be hard to get published. On the other hand, people might tend to hide or ignore problems which are not directly related to their field of research. In this chapter, we first summarize projects for which information on the deployment and the encountered problems have been made available. Then, we analyze the reported problems and classify them according to the number of affected nodes.

2.1. Projects

In the following projects, a variety of sensor nodes have been used. To get an impression on the actual hardware and its characteristics, we give a short overview on the used platforms before presenting the projects and the experienced problems.

The majority of the projects used the Mica2 mote [97] designed at the University of Berkeley and produced by Crossbow Technology Inc. or a

variant of it (Mica2Dot, T-Node [104]). Its main components are an Atmel ATmega128L [92] 8-bit microcontroller and a Chipcon CC1000 [108] radio module for the 433/868/915 MHz ISM bands. Later deployments used the Chipcon CC2420 radio module on the MicaZ and the TMote Sky sensor node. In deployments with special communication requirements such as GlacsWeb and in the described oceanography project, a Microchip PIC microcontroller was used. The key figures for the used hardware and the surveyed projects are given in tables 2.1 and 2.2 respectively. In the latter, the column *yield* denotes the amount of data reported by the sensor network with respect to the expected optimum, e.g., based on the sample rate.

	Mica2(Dot)	T-Node	MicaZ
Microcontroller	ATmega128L	ATmega128L	ATmega128L
Architecture	8 bit	8 bit	8 bit
Clock	7.328 MHz (4 MHz)	7.328 MHz	7.328 MHz
Program Memory	128 kB	128 kB	128 kB
Data Memory	4 kB	4 kB	4 kB
Storage Memory	512 kB	512 kB	512 kB
Radio	Chipcon CC1000	Chipcon CC1000	Chipcon CC2420
Frequency	433 / 915 MHz	868 MHz	2.4 GHz
Data Rate	19.2 kbps	19.2 kbps	250 kbps

	TMote Sky	TinyNode	Oceanography	GlacsWeb
Microcontroller	MSP 430	MSP 430	PIC 18F452	PIC 16LF878
Architecture	16 bit	16 bit	8 bit	8 bit
Clock	8 MHz	8 MHz	<40 MHz	<20 MHz
Program Memory	48 kB	48 KB	32 kB	16 kB
Data Memory	10 kB	10 kB	1536 B	368 B
Storage Memory	1024 kB	512 kB	0.25 kB	64 kB
Radio	Chipcon CC2420	Xemics XE1205	not specified	Xemics
Frequency	2.4 GHz	868 MHz	173 MHz	433 MHz
Data Rate	250 kbps	152.2 kbps	10 kbps	9600 bps

Table 2.1.: Sensor node characteristics. Data for Mica2Dot in parentheses.

2.1.1. Great Duck Island

One of the earliest deployments of a larger WSN was carried out in the summer of 2002 on Great Duck Island [61], located in the gulf of Maine, USA. The island is home to approximately 5000 pairs of Leach's Storm Petrels that nest in separate patches within three different habitat types.

Deployment	Year	#nodes	Hardware	Duration	Yield	Multi-hop
GDI I	2002	43	Mica2Dot	123 days	16%	no
GDI II - patch A	2003	49	Mica2Dot	115 days	70%	no
GDI II - patch B	2003	98	Mica2Dot	115 days	28%	yes
Oceanography	2004	6	Custom HW	14 days	not reported	no
GlacsWeb	2004	8	Custom HW	365 days	not reported	no
SHM	2004	10	Mica2	2 days	up to 50%	yes
Redwoods	2005	33	Miac2Dot	44 days	49%	yes
Potatoes	2005	97	TNode	21 days	2%	yes
Volcano	2005	16	TMote Sky	19 days	68%	yes
Soil Ecologoy	2005	10	MicaZ	42 days	not reported	no
Sensorscope	2006-2008	6-97	TinyNode	4-180 days	not reported	yes

Table 2.2.: Characteristics of selected deployments

Seabird researchers are interested in questions regarding the usage pattern of nesting burrows with respect to the microclimate. As observation by humans would be both too costly and might harm the birds, a sensor network of 43 nodes was deployed for 4 months just before the breeding season. The nodes had sensors for light, temperature, humidity, pressure, and infrared radiation and have been deployed in a single hop network. Each sensor node samples its sensors every 70 seconds and sends its readings to a solar-powered gateway. The gateway forwards the data to a central base station with a database and a WAN connection using a two-way satellite connection to the Internet. During the 123 days of the experiment, 1.1 million readings have been recorded, which is about one sixth of the theoretical 6.6 million readings generated over this time.

In a book chapter [61], the authors analyze the network's behavior in detail. The most loss of data was caused by hardware-related issues. Several nodes stopped working due to water entering the sensor node casing. As all sensors where read out by a single analog-to-digital converter, a hardware failure of one of the sensors caused false readings of other sensors. Due to the transparent casing of the sensor nodes, direct sun light could heat the whole sensor node and thus lead to high temperature readings for nodes which are deployed above ground. Over time, various sensors report false readings such as humidity over 150% or below 0%, or too low or unreasonably high temperature. The temperature sensor of about half the nodes failed at the same time as the humidity sensor suggesting water inside the packaging. Although it did not directly cause packet loss as the gateway was always listening, several nodes did show a phase skew

with respect to their 70 second sending interval. A crash of the database running on the base station resulted in the complete loss of data for two weeks.

After lessons learned from the first deployment, a second deployment was conducted in 2003 [75]. This time, two separate networks, a single-hop network of 49 nodes similar to the one in the first deployment and a multi-hop network with 98 nodes were deployed. The multi-hop network used the routing algorithm developed by Woo [89]. Again, the project suffered from several outages of the base station - this time caused by harsh weather. In the multi-hop network, early battery depletion was caused by overhearing in combination with low-power listening. In the pre-deplomyent calculation, the group did not account for an increased overhearing in the multi-hop network although it could have been predicted.

2.1.2. Oceanography

A small sensor network of 6 nodes was deployed in 2004 on a sandbank off the coast of Great Yarmouth, Great Britain [76] to study sedimentation and wave processes. A node did consist of a radio buoy for communication above the sea and a sensor box on the seabed connected by a wired serial connection. The sensor box had sensors for temperature, water pressure (which allows to derive wave height), water turbidity, and salinity. The authors reported problems with the sensor box due to last-minute software changes which led to cutting and re-fixing of the cable between buoy and sensor, and later, to the failure of one of the sensors caused by water leakage.

2.1.3. GlacsWeb

The GlacsWeb project [57] deployed a single-hop sensor network of 8 glacier probes in Norway. The aim of this system is to understand glacier dynamics in response to climate change. Each probe samples every four hours the following sensors: temperature, strain (due to stress of the ice), pressure (if immersed in water), orientation, resistivity (to detect whether the probe would be in sediment till, water or ice), and battery voltage. The probes were installed in up to 70 m deep holes located around a central hole which did hold the receiver of the base station.

In this deployment, initially, the base station only received data from seven probes and during the course of the experiment, communication

with four probes failed over time. In the end, three probes were able to report their sensor readings. The base station experienced an outage. The authors speculate that the other probes might have failed for three reasons: Firstly, nodes might have been moved out of transmission range because of sub-glacial movement. Secondly, the node casing might have broken due to stress by the moving ice. And thirdly, clock drift and sleeping policy might have led to unsynchronized nodes which hinders communication.

2.1.4. Structural Health Monitoring

To assess the structural health of buildings, the Wisden [58] data acquisition system was conceived. Each node measures seismic motion by means of a three-axis accelerometer and forwards its data to a central station over a multi-hop network. The data samples are time stamped and aggregated in the network to compensate for the limited bandwidth. In the case of an seismic event, the complete data for it is buffered on the node to allow for reliable end-to-end data transmission.

The authors report a bug in their system, where an 8-bit counter was used for the number of locally buffered packets and an overrun would cause packets to not be delivered at all. Also, the accelerometer readings showed increased noise when the battery voltage did fall below an certain threshold.

2.1.5. Redwood Tree

To monitor the microclimate of a 70-meter tall redwood tree, 33 sensor nodes have been deployed along a redwood tree roughly every two meters in height for 44 days in 2005 [80]. Each node measured and reported every 5 minutes air temperature, relative humidity, and solar radiation. The overall yield of this deployment has been 49%. In addition to the Great Duck Island hardware and software, the “Tiny Application Sensor Kit” (TASK) was used on the multi-hop network. The TinyDB [105] component included in TASK provides an SQL-like database interface to specify continuous queries over sensor data. In addition to forwarding the data over the network, each sensor node was instructed to record all sensor readings into an internal 512 kB flash chip.

Some nodes recorded abnormally high temperature readings above $40\,^{\circ}\mathrm{C}$ when other nodes reported temperatures between 5 and $25\,^{\circ}\mathrm{C}$. This

allowed to single out nodes with incorrect readings. Wrong sensor readings have been highly correlated to low battery voltage similar to the report for the Structural Health Monitoring. This should have not been surprising as the used sensor nodes, Mica2Dot motes, did not employ a voltage converter and the battery voltage fell below the threshold for proper operation over time. Also in this project, two weeks of data were lost due to a gateway outage. The data stored in the internal flash chip was complete but did not cover the whole deployment. Although it was estimated that it would suffice, initial tests, calibration, and a longer deployment than initially envisioned led to a full storage after about four weeks.

2.1.6. LOFAR-agro

A detailed report on deployment problems was aptly called "LOFAR-agro - Murphy Loves Potatoes" [47]. The LOFAR-agro project is aimed at precision agriculture. In summer 2005, after two field trials, 110 sensor nodes with sensors for temperature and relative humidity were deployed in a potato field just after potatoes have been planted. The field trials and the final deployment suffered from a long list of problems.

Similar to the oceanography project, an accidental commit to the source code revision control system led to a partially working MAC protocol being installed on the sensor nodes just before the second field trial. Later, update code stored in the nodes' external flash memory caused a continous network code distribution which led to high network congestion, a low data rate and thus the depletion of all nodes' batteries within 4 days. The routing and the MAC component used different fixed size neighbor tables. In the dense deployment, where a node might have up to 30 neighbors, not all neighborhood information could be stored, which caused two types of faulty behavior. Firstly, the routing component of most nodes did not send packets to the gateway although the link would have been optimal. Secondly, as both components used different neighbor tables, packets got dropped by the MAC-layer when the next hop destination was not in its neighbor table. To allow nodes to recover from software crashes, a watchdog timer was used. Either due to actual program crashes or due to a malfunction of the watchdog handling, most nodes rebooted every two to six hours. This did not only cause data loss for the affected node but also led to network instability as the entries for rebooting nodes are removed by their neighbors. As in other projects, the LOFAR-agro project suffered from gateway outages. In this case, a miscalculation of the power

requirements for the solar-powered gateway caused a regular outage in the morning when the backup battery was depleted before the sun rises and the solar cells provided enough power again. The sensor nodes were programmed to also store their readings in the external flash memory, but due to a small bug, even this fallback failed and no data was recovered after the deployment. In total, the 97 nodes which ran for 3 weeks did deliver 2% of the measured data.

2.1.7. Volcano

In August 2005, a sensor network of 16 sensor nodes has been deployed on the volcano Reventador in Ecuador [86]. Each node samples seismic and acoustic data at 100 Hz. If a node detects a local seismic event, it notifies a base stations. If 30% of the nodes report an event in parallel, the complete data set of the last minute is fetched from all nodes in a reliable manner. Instead of immediately reporting all data, which would lead to massive network congestion and packet collisions with current low-power MAC protocols, the nodes are polled by the base station sequentially.

The first problem encountered was a bug in the clock component which would occasionally report a bogus time. This led to a failure of the time synchronization mechanism. The team tried to reboot the network but this trigged another bug, which led to nodes continuously rebooting. After manual reprogramming of the nodes, the network was working quite reliably. A median *event yield* of 68 % was reported, which means that for detected events 68% of the data was received. As with other deployments, data was lost due to power outage at the base station. During the deployment, only a single node stopped reporting data and this was later confirmed to be due to a broken antenna.

2.1.8. Soil Ecology

To monitor the soil ecology in an urban forest environment, ten sensor nodes have been deployed near John Hopkins University in the autumn of 2005. The nodes have been equipped with manually calibrated temperature and soil moisture sensors and packed into a plastic waterproof casing. The sensor application was designed to store all sensor readings in the local flash memory which had to be read out every two weeks to guarantee 100% sensor data yield in combination with a reliable data transfer protocol.

However, due to an unexpected hardware behavior, a write to the flash memory could fail and an affected node would then stop recording data. Further parts of the data have been lost due to human errors while downloading the data to a laptop computer. Similar to previous deployments, the software on the nodes had to be updated and for this the waterproof cases had to be re-opened several times which led to water leakage in some cases.

2.1.9. SensorScope

Sensorscope [4, 5] is a system for environmental monitoring with many deployments across Switzerland. Instead of accuracy of individual sensors, the system design strives for generating models by high spatial density of inexpensive sensing stations. Each station is powered by a solar cell, which is mounted onto a flagstaff alongside the sensors and a TinyNode. Different environmental parameters are captured and gathered for later analysis. Most deployments are in the range of days to a couple of months. The environments are typically harsh, especially in the high mountain terrain deployments, e.g., on the Grand St. Bernard pass. Similar to the report of the LOFAR-agro project, the authors provide a detailed overview of issues and lessons learned in a comprehensive guide [4]. They again stress the importance of adequate packaging of the sensor nodes, and especially the connectors. With respect to these particular harsh environments, substantial temperature variations showed considerable impact on the clock drift. However, while the clock drift typically affects the time synchronization of the network, in this case, it induced a loss of synchronization of the serial interface between the sink node and the GPRS modem. In an indoor test deployment, the authors report on packet loss from interference where the interfering source could not be determined. Finally, a change on the querying interval of the wind speed sensor, when moving from the lab to the deployment, caused counter over?ows, which rendered all sensor readings useless.

2.2. Analysis

This section contains a classification of the problems typically found during deployment according to our own experience and as surveyed in the previous section. Here, a problem is essentially defined as a behavior of a set of nodes that is not compliant with the specification.

We classify problems according to the number of nodes involved into four classes: *node problems* that involve only a single node, *link problems* that involve two neighboring nodes and the wireless link between them, *path problems* that involve three or more nodes and a multi-hop path formed by them, and *global problems* that are properties of the network as a whole.

2.2.1. Node Problems

A common node problem is *node death* due to energy depletion either caused by "normal" battery discharge or due to short circuits. An increased amount of network traffic, compared to initial calculations, led to an early battery depletion due to unexpected overhearing (Great Duck Island [74], section 2.1.1) or repeated network floods (LOFAR-agro [47], section 2.1.6). In [75] (Great Duck Island), a low-resistance path between the power supply terminals was created by water permeating a capacitive humidity sensor, resulting in early battery depletion and abnormally small or large sensor readings.

Low batteries often do not cause a fail-stop behavior of the sensor nodes. Rather, nodes may show random behavior at certain low battery voltages. As reported in [80] (Redwood Tree, section 2.1.5), for example, *wrong sensor readings* have been observed at low battery voltage.

Software bugs often result in *node reboots*, for example, due to failure to restart the watchdog timer of the micro controller (LOFAR-agro [47]). We also observed software bugs resulting in hanging or killed threads, such that only part of the sensor node software continued to operate.

Sink nodes act as gateways between a sensor network and the Internet. In many applications they store and forward data collected by the sensor network to a background infrastructure. Hence, problems affecting sink nodes or the gateway must be promptly detected to limit the impact of data loss (GlacsWeb [57], Great Duck Island [74], Redwood Tree [80]).

2.2.2. Link Problems

Field experiments (e.g., [27, 89]) demonstrated a very high variability of link quality both across time and space resulting in temporary link failures and variable amounts of *message loss*.

Network congestion due to *traffic bursts* is another source of message loss. In [74](Great Duck Island), for example, a median message loss of 30% is reported for a single-hop network. Excessive levels of traffic

bursts have been caused by accidental synchronization of transmissions by multiple senders, for example, due to inappropriate design of the MAC layer [64] or by repeated network floods [47](LOFAR-agro). If message loss is compensated for by retransmissions, a *high latency* may be observed until a message eventually arrives at the destination.

Most sensor network protocols require each node in the sensor network to discover and maintain a set of network neighbors (often implemented by broadcasting HELLO messages containing the sender address). A node with *no neighbors* presents a problem as it is isolated and cannot communicate. Also, *neighbor oscillation* is problematic [64], where a node experiences frequent changes of its set of neighbors.

A common issue in wireless communication are *asymmetric links*, where communication between a pair of nodes is only possible in one direction. In a field experiment [27] between 5-15% of all links have been observed to be asymmetric, with lower transmission power and longer node distance resulting in more asymmetric links. If not properly considered, asymmetric links may result in fake neighbors (received HELLO from a neighbor but cannot send any data to it) and broken data communication (can send data to neighbor, but cannot receive acknowledgments).

Another issue is the physical length of a link. Even though if two nodes are very close together, they may not be able to establish a link (*missing short links*). On the other hand, two nodes that are very far away from each other (well beyond the nominal communication range of a node), may be able to communicate (*unexpected long links*). Experiments in [27] show that at low transmit power about 1% of all links are twice as long as the nominal communication range. These link characteristics make node placement highly non-trivial as the signal propagation characteristics of the real-world setting have to be considered [12] to obtain a well-connected network.

Most sensor network MAC protocols achieve energy efficiency by scheduling communication times and turning the radio module off in-between. Clock drift or repeated failures to re-synchronize the communication phase may result in failures to deliver data as nodes are not ready to receive when others are sending. In [53], for example, excessive *phase skew* has been observed (about two orders of magnitude larger than the drift of the oscillator).

2.2.3. Path Problems

Many sensor network applications rely on the ability to relay information across multiple nodes along a multi-hop path. In particular, most sensor applications include one or more sink nodes that disseminate queries or other tasking information to sensor nodes and sensor nodes deliver results back to the sink. Here, it is important that a path exists from a sink to each sensor node, and from each sensor node to a sink. Note that information may be changed as it is traversing such a path, for example due to data aggregation. Two common problems in such applications are hence *bad path to sink* and *bad path to node*. In [47], for example, *selfish nodes* have been observed that did not forward received traffic, but succeeded in sending locally generated messages.

Since a path consists of a sequence of links, the former inherits many of the possible problems from the latter such as *asymmetric paths*, *high latency*, *path oscillations*, and *high message loss*. In [74](Great Duck Island), for example, a total message loss of about 58% was observed across a multi-hop network.

Finally, *routing loops* are a common problem, since frequent node and communication failures can easily lead to inconsistent paths if the software isn't properly prepared to deal with these cases. Directed Diffusion [37], for example, uses a data cache to suppress previously seen data packets to prevent routing loops. If a node reboots, the data cache is deleted and loops may be created [71].

2.2.4. Global Problems

In addition to the above problems which can be attributed to a certain subset of nodes, there are also some problems which are global properties of a network. Several of these are failures to meet certain application-defined quality-of-service properties. These include *low data yield*, *high reporting latency*, and *short network lifetime* [77].

Low data yield means that the network delivers an insufficient amount of information (e.g., incomplete sensor time series). In [80](Redwood Tree), for example, a total data yield of only about 20-30% is reported. This problem is related to message loss as discussed above, but may be caused by other problems such as a node crashing before buffered information could be forwarded, buffer overflows, etc. One specific reason for a low data yield is a *partitioned network*, where a set of nodes is not connected to the sink.

Reporting latency refers to the amount of time that elapses between the occurrence of a physical event and that event being reported by the sensor network to the observer. This is obviously related to the path latency, but as a report often involves the output of many sensor nodes, the reporting latency results from a complex interaction within a large portion of the network.

The lifetime of a sensor network typically ends when the network fails to cover a given physical space sufficiently with live nodes that are able to report observations to the observer. The network lifetime is obviously related to the lifetime of individual nodes, but includes also other aspects. For example, the death of certain nodes may partition the network such that even though coverage is still provided, nodes can no longer report data to the observer.

2.3. Discussion

Orthogonal to the classification in the previous section, the deployment problems in the surveyed literature fall into two categories: implementation and design defects. A majority of the reported problems have been caused by defects in the realized hardware and software, and can be fixed after they have be detected, analyzed, and understood. Here, inspection tools allow to find the defects quicker.

The two most underestimated problems in the surveyed sensor network deployments have been the water-proof packaging of the sensor nodes required for an outside deployment and the provision of a reliable base station which records sensor data and has to run for months and years. This suggests that sensor nodes should be sold together with appropriate packaging. However, due to the variety of sensors used for different applications, a common casing is often not practicable or possible. The provision of a reliable base station is not specific to wireless sensor networks and mostly depends on a reliable power supply and software.

The packet collisions and network congestions caused by traffic bursts are an example for problems caused by improper design, in this case, by an unsuitable medium access control design. In the case of the volcano deployment (see section 2.1.7), the application designer was able to avoid packet collisions by serializing the data transfer from the nodes to the base station. After an event was detected, the base station polls each node individually for its data. However, this constitutes an uncessary “work around” as the MAC layer should be able to handle this.

To address both kinds of problems, we will argue in the first part of this thesis that a majority of deployment problems can be detected by passive observation of the exchanged radio messages, which we call passive inspection. Then, in the second part, we will present our work on collision-free medium access protocols which can handle traffic bursts caused by event-triggered applications.

Part I.

Fault Detection

3. Network Inspection

Deployment of sensor networks in real-world settings is typically a labor-intensive and cumbersome task, as we have shown in the previous chapter.

During the development of a sensor networks application, an array of tools such as simulation, emulation, and testbed is used in the lab. However, these tools cannot applied to an already deployed network. Other tools that have been designed to query the state of a deployed network consume scarce resources and require to modify the nodes.

To deal with these limitations, we propose a *passive* approach for sensor network inspection by overhearing and analyzing sensor network traffic to infer the existence and location of typical problems encountered during deployment.

In this chapter, we first provide an overview on available tools and concepts to help in the development of sensor networks and discuss their shortcomings with respect to the inspection of already deployed networks. Next, we propose a set of requirements which a useful tool to inspect deployed sensor networks has to fulfill. Finally, we present our "passive inspection" approach and show that it on one hand meets the stated requirements and on the other hand allows to detect most of the problems listed in section 2.2.

3.1. Related Work

This section presents and discusses existing tools and methods to help in the development and deployment of wireless sensor networks and related areas such as wireless LANs. We first discuss tools which are used before the actual deployment of sensor networks. Our main focus is, however, on tools that can be used for already deployed networks. In addition, we discuss related tools for analysis of network traces for wireless LAN, and for monitoring and debugging of distributed systems.

3.1.1. Simulation and Testbeds

Simulations and testbeds are a valuable tool as they support the development and test of sensor networks *before* deployment in the field.

Simulation is commonly used for the evaluation of new network protocols and algorithms, and it is also used for wireless sensor networks In the simulator, a complete network of sensor nodes together with a radio model which controls the success of packet transmission is run and analyzed. The abstraction level of simulation tools vary from high-level discrete event simulation (with actions like "send packet", and "receive packet") down to accurate instruction level simulation of a particular sensor node, e.g., with the AVRORA simulator [78]. Simulation tools with a high abstraction level, e.g., general purpose simulators such as ns-2 [10] or OMNet++ [83] and wireless sensor network specific simulators, e.g., SENS [73] require to implement applications in the context of their programming framework. In contrast, other sensor network simulators like TOSSIM [49] and EmTOS [29] allow to simulate a network of TinyOS sensor nodes as used by most of the projects in section 2.2

Testbeds for wireless sensor network such as MoteLab [87], Kansai [25], Mirage [19] or Twist [31] allow to test sensor network applications in a fixed setup, mostly in academic office buildings. They consists of a large number of sensor nodes that are provided with permanent power supply and a back-channel for logging and control. The testbed infrastructure allows for nodes to be programmed and controlled (on/off, reboot) and provides a wired back-channel from each node, such that sensor nodes can be instrumented to send status information to an observer. As the behavior of a node and particularly its radio module is not simulated, testbeds provide a far more realistic behavior than simulations, but do not scale to large numbers of nodes.

Hybrid Simulation integrates simulation and testbed concepts into a common framework. In EmStar [28], the radio communication between sensor nodes can either be simulated using various radio models or handled by real sensor nodes in a testbed. Hence, the sensor nodes in an EmStar testbed need instrumentation and a wired back-channel. Each of the nodes can either act as an autonomous testbed node or only provide an interface to its radio module to be used by an simulated node.

In contrast to EmStar, a "simulation-based augmented reality system" [85] allows to simulate a network of both simulated and real nodes. In such a mixed topology, so-called "super nodes", which consist of a real node that are attached to the simulator, are present in both the simulated and the real wireless network. By this, the behavior of real nodes in a large (simulated) network can be studied in detail.

Deployment-Support Network A second network which helps with the deployment of wireless sensor nodes was proposed in [21]. In this approach, each sensor node is connected physically to a deployment-support node which provides the functionality of a testbed but without a fixed network infrastructure. Instead, the reliable Bluetooth Scatternet of the BTnodes [95] provides a wireless back-channel and enables remote control of the sensor nodes. This effectively creates a wireless testbed and allows the sensor nodes to be deployed without additional restrictions.

Discussion While simulation and lab testbeds are helpful tools to test an application prior to deployment, they fail to provide realistic environmental models (e.g., regarding radio signal propagation, sensor stimuli, chemical/mechanical strain on sensor nodes). Hence, environmental effects in real deployments often trigger bugs or degrade performance in a way that could not be observed during pre-deployment testing.

To track down such problems, a developer needs to inspect the state of network and nodes. While this is easily possible during simulation and experiments on lab testbeds (wired backchannel from every node), access to network and node states is very constrained after deployment.

Although the deployment-support network approach allows to inspect a deployed sensor network, the fact that sensor nodes need to be physically wired to DSN nodes (requiring as many DSN nodes as there are sensor nodes) limits this approach significantly. Also, node software must be instrumented to be used with wired or wireless testbeds as discussed in the following section.

3.1.2. Active Inspection

Current practice to inspect a deployed sensor network requires *active* instrumentation of sensor nodes with monitoring software and monitoring traffic is sent in-band with the sensor network traffic to the sink. The

most suitable tools for deployed networks are Nucleus, Sympathy, and Memento.

Nucleus [79] is a management system for sensor networks. It allows to query sensor node attributes over the network and provides a logging framework that delivers important local events to the sink. By querying for example the neighbor table or the state of the routing module, the nodes' networking behavior can be monitored and inspected in a live setting.

Sympathy [63] is a system for the detection and debugging of faults based on statistical data collected by individual nodes and forwarded to the sink node. It supports a fixed set of statistical metrics related to networking and makes use of the neighbor and routing tables as well as the number of packets received (correctly vs. with bit-errors) and transmitted. In case of a fault, e.g., if no data is received for a node in a certain period of time, the system uses a heuristic decision tree to infer the most likely root cause of the fault.

Memento [70] focusses on the efficient monitoring of nodes' state and in-network failure detection for dead nodes and network partitions. The failures detection algorithm is designed to be robust to packet loss. Besides *node dead*, other binary states are reported, e.g., *low battery* and *network congested*. Compared to Sympathy, it is less flexible but it reduces the network traffic for monitoring significantly.

Discussion The main advantage of active inspection is that it can provide accurate access to the internal state of wireless sensor nodes in their real-world deployment environment. Unfortunately, however, the active inspection approach has several fundamental limitations. Firstly, problems in the sensor network (e.g., partitions, message loss) also affect the monitoring mechanism, thus reducing the desired benefit. Secondly, scarce sensor network resources (energy, CPU cycles, memory, network bandwidth) are used for inspection. In Sympathy, for example, up to 30% of the network bandwidth is used for monitoring traffic. Thirdly, the monitoring infrastructure is tightly interwoven with the application. Hence, adding/removing instrumentation may change the application behavior in subtle ways, causing probe effects. As reported in the previous chapter,

changes to a deployed network should be avoided, if possible, to reduce the risk of failure of the network. Also, it is non-trivial to adopt the instrumentation mechanism to different applications or sensor network operating systems. Memento, for example, assumes a certain tree routing protocol being used by the application and reuses that protocol for delivering monitoring traffic.

3.1.3. Network Sniffing

Passive observation by means of packet sniffing has been applied to wireless (and wired) LANs before [33].

Recently, four systems for passive analysis of WLANs and sensor networks have been proposed, namely WIT and JIGSAW for WLAN, and STNS and LiveNet for sensor networks. JIGSAW processes the traces of multiple WLAN sniffer nodes on-line whereas the other systems record traces collected by disconnected nodes which are later merged and analyzed offline. For the merging and time synchronization of separate traces, all systems rely on messages being overheard by two or more sniffer nodes.

WIT [52] follows an offline approach, merging redundant traces of network traffic collected by distributed sniffers. After merging packet traces of all sniffers, WIT makes use of a detailed model of the 802.11 MAC [107] to infer which packets have actually been received by the respective destination nodes and derives different network performance metrics.

JIGSAW [18], in contrast to WIT, focuses on online inference of link-layer and transport-layer connections and their characteristics for larger networks based on a detailed model of the 802.11 MAC. It utilizes time synchronization on the sniffer nodes to reduce the computation cost for merging the traces. In [18], the authors report on a 150 node sniffer network which generated 2.7 million events in a day and which JIGSAW is able to merge, synchronize and analyze in real time on a single server.

SNTS: Sensor Network Troubleshooting Suite [41] uses distributed sniffer sensor nodes that record overheard traffic in local Flash storage. After an experiment, the nodes are collected and the packet traces are transfered to a central server. In contrast to WIT and JIGSAW, where the underlying 802.11 packet format is standardized, SNTS decodes the raw packet

dumps based on a text file that describes the packet format. As an example for a possible processing of the packet traces, the authors employed machine-learning algorithms to identify bad sequences of events, which lead to an observed bug in the protocol/system, allowing them to fix the problem.

LiveNet [69] is a sensor network tool for network dynamics analysis. It uses passive sniffer nodes that forward overheard packets on the serial port to a connected laptop computer or stores them locally in the flash memory. Using an out-of-band mechanism, traces are collected on a central server and merged based on the WIT approach. The main analysis described in [69] is the reconstruction of the spanning tree routing paths using statistical methods.

Discussion Sensor networks differ substantially from wireless LANs. While typical wireless LANs are single-hop networks that can be observed with one or few sniffers, sensor networks are typically multi-hop networks. As sensor networks do not have standardized packet formats, an approach similar to the one employed by SNTS, which uses a packet description file, is favorable. Also, many of the problems encountered during deployment of sensor networks are not present in WLANs.

In general, offline approaches using sniffer nodes which log packets locally are cumbersome to use for deployed sensor networks, as they would have to be continuously placed within the network and later recollected to get access to the packet logs.

Finally, the use cases of the described works are rather specific and the proposed tools cannot easily be adapted to other protocols and applications, or used as general inspection tools.

3.1.4. Distributed Systems

In the more general context of management and debugging of distributed systems, a large body of work exists. However, here, we will focus on classes of work which are related to the inspection of deployed sensor networks.

Performance One such class of closely related work is performance debugging of distributed systems (e.g., [1,3]) where message traces are used to reconstruct causality paths and their latencies. Although [1] advocates

passive monitoring to collect the traces of distributed systems, all experiments are performed by tagging client calls with a request ID for the used J2EE components. Both [1, 3] are based on LAN infrastructure and assume that message latency is exclusively caused by the inspected software components and not by concurrent network traffic.

Network Traffic Analysis A number of data stream management systems have been specifically developed for network traffic analysis, e.g. Gigascope [20] and Tribeca [72]. Data streams of networking events are processed online based on queries defined in a SQL-like language [20], or by procedural code [72]. The main focus of this work is on processing speed of the queries to allow monitoring even corporate networks.

Discussion The principal idea of those approaches, namely online processing of network traces, tends to support wireless sensor networks in combination with network sniffing very well. However, while trying to create queries which would allow to detect most of the the problems presented in section 2.2, we found it difficult, if not impossible, to express the required indicators, e.g., to test for a routing loop, using the SQL variants of these systems. Although specifying queries using procedural code allows for more flexible processing, extra care has to be taken to allow for re-use of existing queries. Therefore, a modular approach which allows to combine existing queries with new ones, preferably specified in a procedural language, would be beneficial.

3.2. Problem Statement

In the previous section, we have surveyed existing work with respect to the inspection of wireless sensor networks. Based on our observations, we outline the requirements for a useful inspection tool in this section.

3.2.1. Online Analysis

An effective inspection tool has to work and provide insight into the deployed sensor network in a (near) real-time fashion. Otherwise, if network state information is not provided in a timely manner, the person deploying the network might not be able to learn about potential problems in the network before she might have to leave the deployment area.

3.2.2. No Instrumentation of WSN Nodes

To avoid problems in a deployed WSN, it is desirable to offer an inspection approach that does not require the sensor nodes to be altered, neither in hardware nor in software, to prevent problems resulting from changes on the sensor nodes to manifest. For example, firmware updates over-the-air may fail and leave the network in an inconsistent state, while manual updates often require the sensor node cases to be opened, which might harm the nodes' waterproofing. Clearly, it is beneficial, if no extra resources on the sensor nodes are required for the inspection tool. Hence, no program code should be added. By this, no additional memory is used, no radio traffic is generated, and potential race conditions due to the added code are avoided.

3.2.3. Multi-Hop Networks

Routing in sensor networks is a major research area and multi-hop networks already have be used in real deployments as shown in section 2.1. Because of this, the inspection tool also has to support multi-hop networks.

3.2.4. Flexibility

Paramount to the previous requirements is the assumption that an inspection tool for sensor networks has to be quite flexible, otherwise it will not be used. Rather than being limited to a single sensor network OS (e.g., TinyOS) and its default network stack, it should be applicable to all kinds of sensor node platforms and networking implementations. As both sensor network applications and routing protocols may evolve over time, the inspection tools should be easy to re-configure and update. Furthermore, the inspection tool should support the detection of a wide range of problems and not be focused on a specific one.

3.3. Passive Inspection of WSN

In this section, we first present our approach for the passive inspection of deployed wireless sensor networks which fulfills the requirements for a useful inspection tool. Then, we demonstrate how the problems in section 2.2 can be detected based on overheard network traffic.

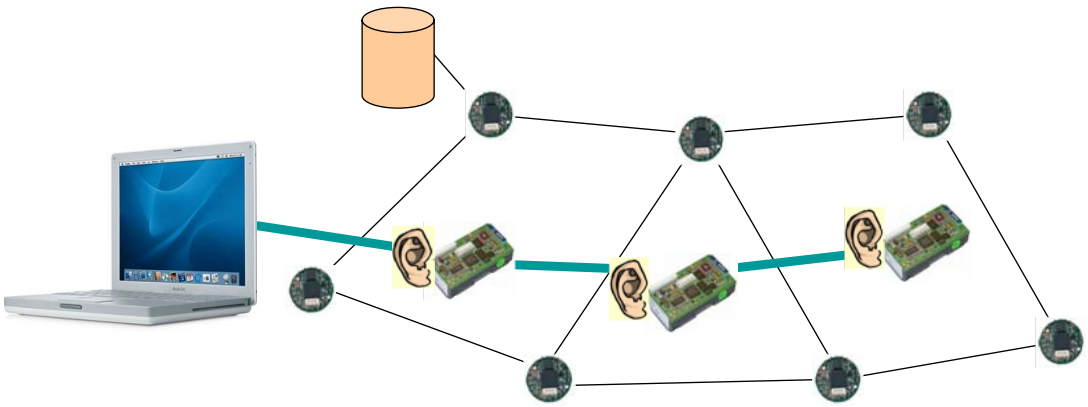


Figure 3.1.: A deployment-support network (rectangular nodes) is a physical overlay network that overhears sensor network (round nodes) traffic and delivers it to a sink using a second radio.

3.3.1. Basic Approach

We advocate to inspect deployed sensor networks using a system that allows to capture the radio traffic on a network wide scale, combine the overhead packets on a central server and provide a flexible framework to define detectors that operate on the captured packets and uncover abnormal behavior.

To overhear the network traffic of a larger multi-hop sensor network, we make use of a so-called *deployment support network* (DSN) [8]: a wireless network that is temporarily installed alongside the actual sensor network during the deployment process as depicted in figure 3.1. The DSN can be removed after the satisfactory operation of the sensor network has been asserted. Each DSN node provides two different radio front-ends. The first radio is used to overhear the traffic of the sensor network, while the second radio is used to form a robust and high-bandwidth network among the DSN nodes to reliably collect overheard packets. The overheard stream of packets is then analyzed to infer and report problems soon after their occurrence.

One might argue that the deployment of the DSN may be as difficult and error-prone as deploying the sensor network itself. However, as the DSN is only used to monitor the network and detect faults for short periods of time (in the order of days), energy and resource constraints are not a primary issue here. This enables the use of more reliable networking

technologies such as Bluetooth or Wireless LAN. For example, Bluetooth has been designed as a cable replacement and employs techniques such as frequency hopping and forward error correction to provide highly reliable data transmission.

3.3.2. Indicators

An indicator is an observable behavior of a sensor network that hints (in the sense of a heuristic) the existence of one of the problems discussed in section 2.2. In the context of passive inspection, we are particularly interested in indicators that can be observed purely by overhearing the traffic of the sensor network as this does not require any instrumentation of the sensor nodes. We call such indicators *passive*. The structure of this section mirrors that of the problem analysis in section 2.2, discussing passive indicators for the problems outlined there.

In fact, passive indicators heavily depend on the protocols used in the sensor network. However, there are a number of protocol elements that are commonly found in sensor network applications that can be exploited. For example, many protocols exchange regular beacon messages, all packets need to contain the per-hop destination MAC address, some packets also contain the per-hop source MAC address to identify the sender of the message, and some packets do contain a monotonically increasing sequence number.

Node Problems

Node death Many commonly used MAC and routing protocols (e.g., [30, 89]) require every node to transmit a beacon message at regular intervals, in particular for the purpose of time synchronization and neighborhood management. Failure to transmit any such message for a certain amount of time (typically a multiple of the inter-beacon time) is an indicator for node death. Also, node death can be assumed if a node is no longer considered a neighbor by any other node (see the next subsection on Link Problems).

Node reboot When a node reboots, its sequence number counter will be reset to an initial value (typically zero). Hence, the sequence number contained in messages sent by the node will jump to a smaller value after a reboot with high probability even in case of lost messages, which can serve as an indicator for reboot. Note that a reboot cannot be detected

this way when the node crashes just before the sequence number counter would wrap around to its initial value. However, a simple fix would be to set the sequence counter to some value other than the initial value at wrap-around, such that a wrap-around could be distinguished from a reboot.

Wrong sensor readings These can only be passively observed when application messages contain raw sensor readings. The decision whether a certain sensor reading is wrong or not strongly depends on the type of the particular application. One could for example exploit the fact that sensor values of nearby nodes are correlated in many applications. For other applications, the range of valid sensor values might be known a priori.

Link Problems

Discovering neighbors Depending on node density, a node in a sensor network may have a large number of other nodes within communication range with largely varying link quality. Most multi-hop routing protocols maintain a small set of neighbors with good link quality. Unfortunately, the set of neighbors chosen by a node cannot be passively observed directly. However, there are two ways to learn about the neighbors of a node. Firstly, by overhearing the destination addresses of messages a node sends we can learn a subset of the neighbors. The second approach exploits link advertisements sent by nodes to estimate link quality. Since links are often asymmetric in sensor networks, link quality estimation must consider both directions of a link. As a sensor node can only measure the quality of one direction of a link directly (e.g., by means of the fraction of beacon messages being received), nodes broadcast link advertisement messages containing the addresses and quality of the incoming links from their neighbors. These messages can be passively observed to obtain information about the neighbors of a node and the quality of the associated link.

Knowing the neighbors of a node, we can detect neighbor oscillation and isolated nodes. If the locations of nodes are known, we can also discover missing short links and unexpected long links.

Message loss Again, it is not directly possible to decide whether or not a node has received a message by means of passive observation. However, in many situations reception of a message by a node does trigger the transmission of another message by that node (e.g., acknowledgment,

forwarding a message to the next hop). If such a property exists, failure to overhear the second message within a certain amount of time after the first message has been overheard is an indicator for message loss. Note that with this approach, it is not possible to tell apart message loss from nodes that receive but fail to forward messages. Another issue with this approach is that the DSN may fail to overhear the second message although it has actually been sent. In this case, one would take the wrong conclusion that message loss occurred. Similarly, even if the messages contain sequence numbers and it can be confirmed that a node did sent a message (by overhearing the next message), it is still not possible to distinguish the forwarding of a message from the sending of a new message.

Latency To estimate the latency of a link, the same approach as for detecting message loss is used. The time elapsed between overhearing the causal and the consequential message gives an estimate of the latency, which includes processing delays in the node.

Congestion The level of link congestion (i.e., frequency of collisions) perceived by a sensor node cannot be passively observed. However, the level of congestion experienced by a deployment support node overhearing the traffic that is being addressed to this sensor node can be used as a rough approximation.

Phase skew Again we can exploit the existence of beacon messages that are sent at regular time intervals. A change of the time difference between receipt of beacons from neighboring nodes indicates phase drift. Averaging over multiple beacon intervals can help eliminate variable delays introduced by, e.g., medium access.

Path Problems

Discovering paths In order to discover the routing path between two sensor nodes (e.g., from node to sink and from sink to node), we would need access to the routing information maintained by sensor nodes. As for the neighbor table, this is not directly possible with passive observation. There are two possible ways around. Firstly, we can reconstruct a path by tracking a multi-hop message as it travels from source to destination. Using the per-hop sender and receiver addresses of overheard messages, we can reconstruct multi-hop paths. However, for this we need a way to

decide whether two messages belong to the same multi-hop transmission. This is easily possible, if the message payload contains a unique identifier such as an end-to-end sequence number. Also, if the message payload is relayed unmodified along the path, we can compare packet contents to decide (with some probability) whether two packets belong to the same multi-hop message transmission. Things get more difficult if message contents are changed along the path, for example due to in-network data aggregation. In this case, one might be able to exploit temporal correlations between messages, assuming that a node will forward a (modified) message soon after it has received the original message.

Another issue is that some protocols do not include the per-hop source address in messages as it is not needed due to the lack of acknowledgments. For example, TinyOS 1.x [106] does not provide a field for the per-hop source address in its standard packet header. One possible heuristic to infer the missing per-hop source address nonetheless exploits the fact that the per-hop source address of a forwarded packet equals the per-hop destination address of the original packet.

Secondly, we can overhear routing messages (if there exist any) to discover paths. While these messages typically indicate that a node has established a route, it is often impossible to reconstruct the route. To construct a spanning tree, for example, it is sufficient that nodes broadcast messages containing their address and distance to the sink, but not their parent in the tree. The latter would be necessary to reconstruct the spanning tree from overheard traffic.

If paths can be discovered, we can also easily detect path oscillations and find missing paths from nodes to sink and vice versa. Using similar techniques as for links, we can estimate message loss and latency along a path.

Loops Like for path discovery, we need a mechanism to decide whether or not two packets belong to the same multi-hop message exchange. If such a mechanism exists, a message that is addressed to a node that previously sent the same message indicates a routing loop. Alternatively, discovered paths can be directly examined for loops.

Global Problems

As discussed in section 2.2, global problems such as low data yield, high reporting latency, or insufficient network lifetime are typically due to a

combination of different node, link, and path problems. Hence, the indicators for the latter problems discussed above can be considered as indicators for these global problems.

Partitions Knowledge of the routing paths as discussed above allows to obtain an approximate view on the routing topology and to detect network partitions.

3.4. Summary

The presented passive inspection approach removes the limitations of active inspection: no instrumentation of sensor nodes is required, no sensor network resources are used, and networking problems in the deployed WSN do not affect the inspection.

The inspection mechanism is completely separated from the application. Thus, it can be more easily adopted to different applications and added or removed without altering sensor network behavior. Even problems in multihop networks can be detected as the DSN covers the whole network. Online analysis (as opposed to long periods of data collection followed by offline analysis) contributes to a more effective deployment process, as it allows an engineer to go out and study affected nodes while a problem is still present. Also, problems can be fixed in an incremental fashion as they occur, thus reducing the chance for complex aftereffects.

Besides these advantages, a number of challenges need to be addressed when implementing passive inspection:

Incomplete information

The DSN may fail to overhear some packets and messages might not contain all information that is needed to infer a problem. To support robust problem detection nonetheless, appropriate loss-tolerant detectors are needed.

Flexibility

There is no established protocol stack for sensor networks – a large variety of radio configurations, MAC, routing, and application layer protocols are in use. To support this open protocol space, a packet capturer that works with a large variety of MAC protocols and radio configurations is required.

Reliability

The DSN should provide reliable wireless communication. For this, an established wireless technology such as Bluetooth, which has been designed as a cable replacement, employing frequency hopping and other techniques to minimize loss, or wireless LAN should be used.

4. SNIF: Sensor Network Inspection Framework

The previous chapter motivated the need for passive inspection. This chapter will focus on a concrete instance of this approach called Sensor Network Inspection Framework (SNIF), which is – as the name suggests – intended as a widely applicable framework for passive inspection.

We first present SNIF's architecture and then explain the framework components in more detail. For a concrete application, we evaluate SNIF in a simulation framework in terms of accuracy and latency.

4.1. Architectural Overview

In this section, we present the architecture of SNIF as depicted in figure 4.1. The network traffic of the wireless sensor network is overheard by a distributed sniffer based on a Deployment Support Network as illustrated in figure 3.1. Each DSN sniffer node captures WSN traffic with one radio receiver and relays the captured raw packets over the second radio. In order to correctly order packets received by different sniffer nodes, the reception time of each packet has to be recorded, which requires the DSN nodes to be time synchronized. The DSN nodes form a collection tree and forward all captured packets to a laptop computer running the SNIF sink. On the SNIF sink, the overheard raw packets are decoded by the packet decoder. Then, the packets are processed by a data stream processor to detect the root causes of failures. The data stream processor makes use WSN-specific operators that implement passive indicators such as those described in section 3.3.2. Finally, information on the state of individual nodes and the network can be accessed by the person who deploys the WSN via a graphical user interface.

As shown in figure 4.1, all parts of the system can be configured by the user. Sniffer nodes and the packet decoder are configured by a text file, the data stream processor and the root cause analysis are configured by Java code.

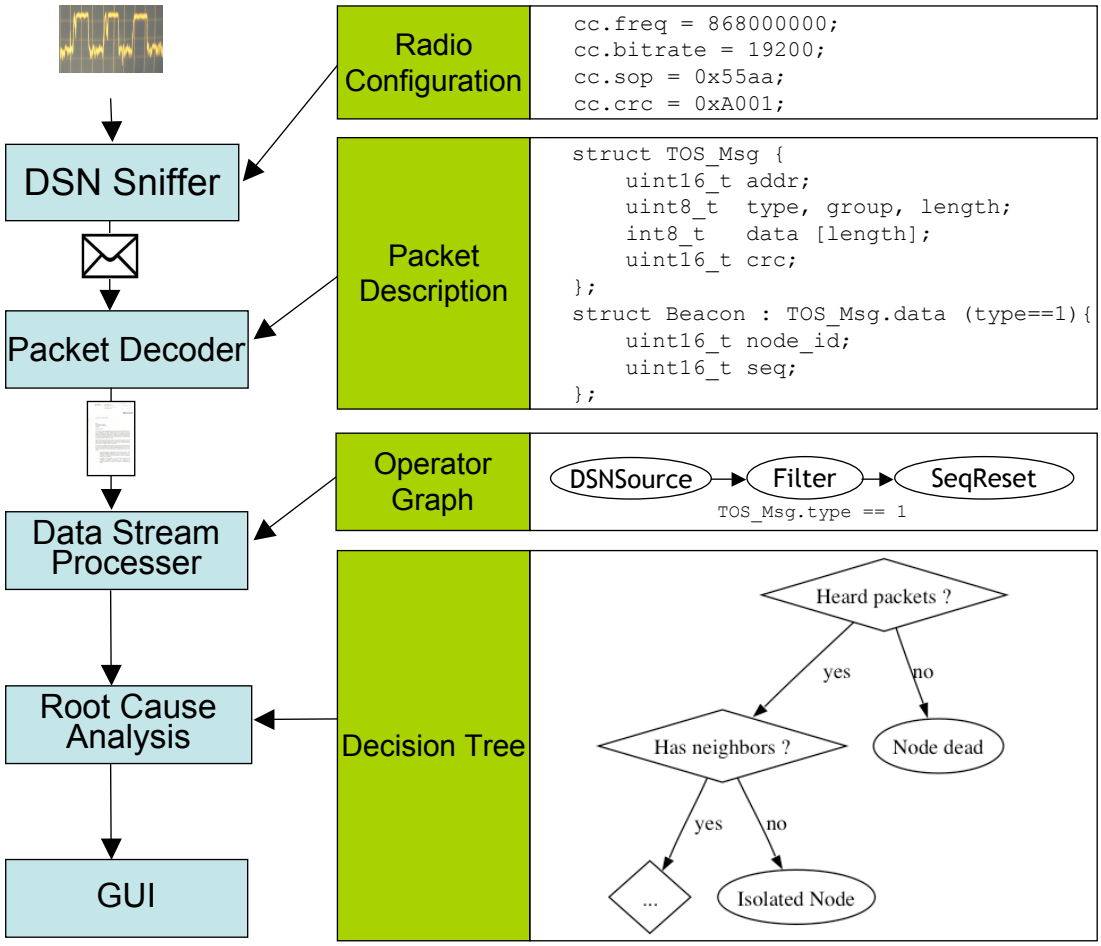


Figure 4.1.: SNIF architecture with parts of an example configuration to detect node reboots, dead and isolated nodes.

For the realization of the distributed sniffer, we make use of the BTnode platform, which we describe in section 4.2. To forward overheard WSN traffic in a reliable manner, the Bluetooth radio is used. However, Bluetooth does not provide time synchronization as a service. Therefore, we developed a new time synchronization technique for Bluetooth networks that we present in section 4.3. In section 4.4, we present our generic distributed sniffer based on the time synchronized DSN. All parts beside the distributed sniffer are implemented in Java and run on a laptop computer or a common PC. We presented these components in detail in sections 4.5 until 4.8.

4.2. Deployment Support Network

The distribued sniffer builds upon a Deployment Support Network to overhear and forward WSN traffic to the SNIF sink. In our implementation, we chose the BTnode platform as the base of our distributed sniffer, as it already provides both required communication primitives for the distributed sniffer, namely network broadcast to disseminate the sniffer configuration and route-to-sink to collect overheard traffic, and provides two independent radio interfaces.

The BTnode platform has been used as a DSN before by others [8, 21] and we did not contribute significantly to these works. However, to provide for a complete overview of SNIF, we summarize the hardware and software of this platform in this section.

4.2.1. Hardware: BTnode rev3

The BTnode rev3 depicted in figure 4.3 is the third incarnation of a Bluetooth-enabled embedded system designed at ETH Zurich. Similar to the sensor network nodes used in the various deployments in chapter 2.1, it mainly consists of a microcontroller and a radio. Its most distinct feature, compared to those nodes, is the presence of two radio modules, a Bluetooth module and the Chipcon CC1000 low-lower radio. Furthermore, an additional 256 KB memory chip provides amble processing memory which alleviates the memory scarcity of the 4 KB provided by the ATmega128L microcontroller alone.

Microcontroller and Memory

Central to the BTnode rev3 architecture depicted in figure 4.2 is the Atmel ATmega128 8-bit microcontroller [92]. It provides 128 KB of program flash, four KB of EEPROM and four KB of internal RAM. An additional 256 KB SRAM module is connected via an 16-bit address and an 8-bit data bus. As the 16-bit address bus only allows to address 64 KB of RAM, two additional I/O lines are used for bank switching between the four available 64 KB RAM banks. The ATmega128L microcontroller provides various standard busses for aynchrous/synchronous serial communication, I2C and SPI, eight analog inputs, and various digital I/O pins.

It runs with a system clock of 7.3278 MHz, a rate commonly found with microcontrollers as it is an integral multiple of the standard serial

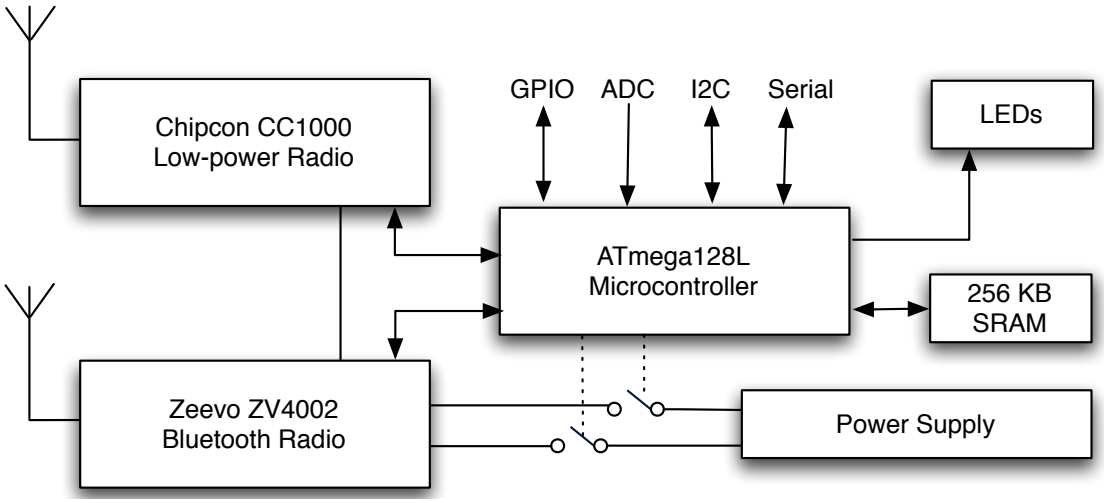


Figure 4.2.: BTnode rev3 architecture. Both radio modules can be operated and powered independently.

communication baud rate of 9600 bps. A 32768 Hz real-time clock allows to keep the time while the processor is in an energy-saving sleep mode.

Communication

Low-Power Radio The Chipcon CC1000 [108] low-power radio can be tuned in software to communicate on a frequency between 400 and 1000 MHz. However, the antenna interface has to be roughly configured for a particular ISM band such as 433, 868 or 915 MHz, where 868 and 915 use the same configuration. On the BTnode, the European 868 MHz ISM band is used. The CC1000 uses frequency-shift keying for sending data at up to 76.8 kbps. If the build-in Manchester-encoding is used, the CC1000 can synchronize itself to incoming data, otherwise bit synchronization has to be performed in software by the microcontroller which does not allow for the microcontroller to handle other tasks in parallel. On all sensor board designs based on an ATmega128 and the CC1000, the CC1000 is controlled via three I/O lines and radio data is transferred using the SPI bus as a synchronous serial bus. Due to the fact that the SPI component of the ATmega128 does not allow to buffer an outgoing byte ahead of time, all ATmega128-CC1000-based sensor nodes only communicate at 19200 baud.

Bluetooth Bluetooth operates in the unlicensed 2.4 GHz ISM band and uses frequency hopping to achieve reliable communication even in noisy

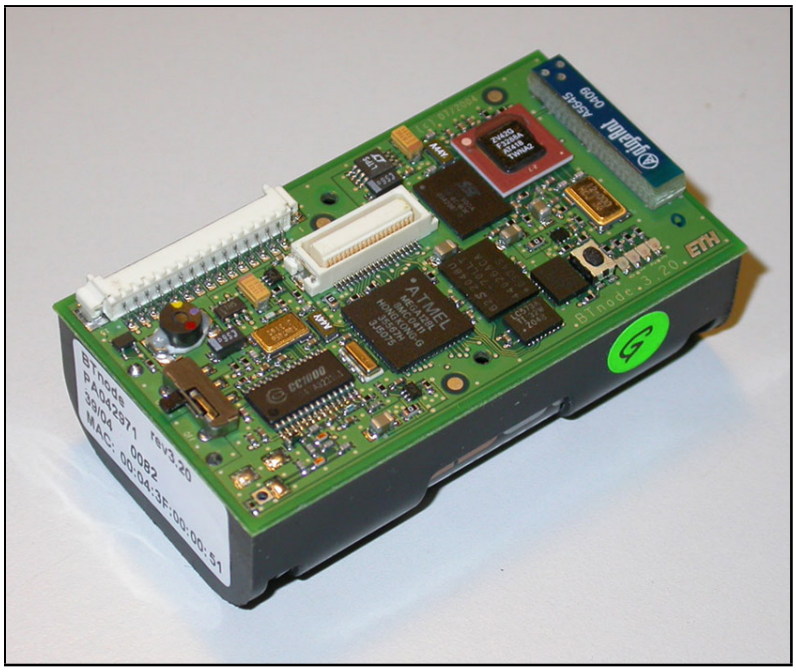


Figure 4.3.: BTnode Rev3.20. The ATmega128L microcontroller is in the center. The Chipcon CC1000 low-power radio and its support circuitry are situated on the lower left, the Bluetooth module in the upper right corner.

environments. A group of devices which are synchronized to a common clock and frequency hopping pattern is called a *Piconet*. The device which provides the reference time for synchronization is called *master*, all other devices are referred to as *slaves*. Piconets have a star topology with the master at the center, that is, direct communication is only possible between a master and a slave, but not between slaves. A *Scatternet* consists of several inter-connected Piconets in which some nodes are part of more than one Piconet at the same time as illustrated in figure 4.4, where large circles indicate a Piconet.

The Zeevo ZV4002 Bluetooth module contains a powerful ARM7 core which handles all Bluetooth baseband communication and provides a serial HCI interface as specified by the Bluetooth Special Interest Group [93]. As it supports participation in up to 4 Piconets, it support Scatternets. The ZV4002 is connected to the ATmega128 by an asynchronous serial port at 115200 baud.

Power

The BTnode runs on a single internal voltage level of 3.3 V which is either provided by a DC-DC step-up convertor powered by two AA batteries or

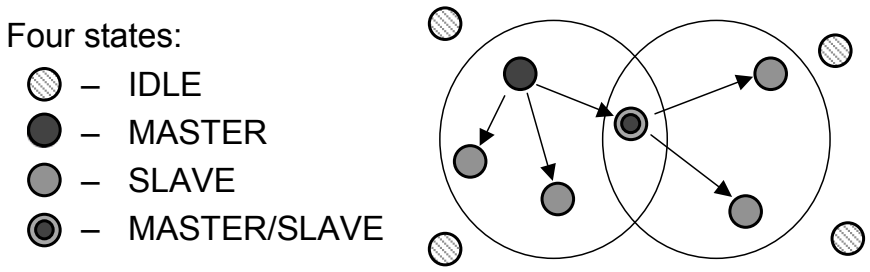


Figure 4.4.: Illustration of Bluetooth Scatternets.

by a linear dropout regulator powered from an external DC input. The power for both radio modules can be switched independently by the microcontroller as depicted in figure 4.2.

4.2.2. Software: BTnut

The additional 256 KB RAM of the BTnode allows for the use of a multi-threading OS. In addition to normal OS services such as threading, memory management, and device drivers, the BTnode requires a radio stack for the Bluetooth and the CC1000 radios as well as support for the banked memory.

Nut/OS

For the OS of the BTnode, we use Nut/OS [98] which is a cooperative multi-threading OS. Its main use is the development of embedded networked Ethernet devices for the Atmel AVR family of microcontrollers. It provides a POSIX-like device driver interface and already supports all internal ATmega128 busses such as SPI, I2C, and serial communication.

BTnode Support

As the ATmega128 can only access 64 KB of RAM directly, we have provided low-level functions that allow to copy memory blocks from one of the hidden banks into the main memory and vice versa. In addition, the init routines are modified to take care of the Bluetooth and the CC1000 module.

Bluetooth Stack

The main component in BTnut is its Bluetooth stack which supports Scatternet networking. Figure 4.5 depicts the lower layers of the standard Bluetooth protocol stack. The component that implements the physical

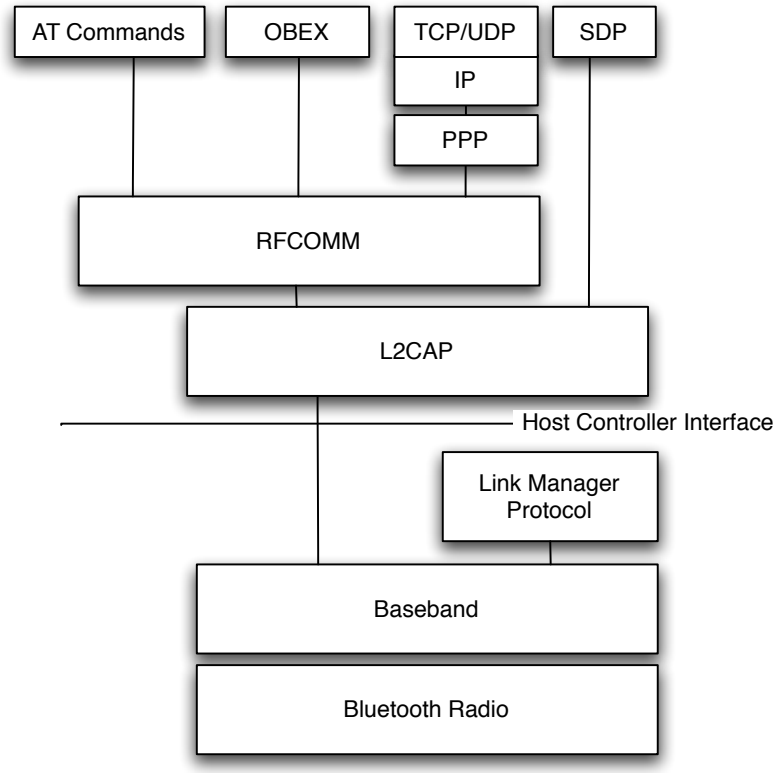


Figure 4.5.: Lower layers of the standard Bluetooth Protocol Stack. Components above the Host Controller Interface (HCI) have to be implemented in software. All but SDP and OBEX are provided in BTnut.

Bluetooth functionality is called Bluetooth Host Controller and the entity that uses it is defined as Bluetooth Host. Accordingly, the interface between the Host and the Host Controller is the Host Controller Interface (HCI). All components above this interface are implemented on the Host, which in our case is the BTnode. Out of those depicted, all but the Service Discovery Protocol (SDP) and the Object Exchange (OBEX) protocol are implemented by BTnut.

Host Controller Interface The HCI implementation [59] of BTnut contains all common HCI commands. A HCI receiver thread is used to handle packets received from the Bluetooth module. A synchronous command call results in the calling thread beeing put into a sleeping state and being woken up by the HCI receiver thread upon command completion.

L2CAP The L2CAP layer provides TCP-like communication over Bluetooth. In addition to this connection-oriented mode, it also provides a connection-less mode which allows to send datagrams without an explicit connection establishment between a master and a slave node in a Piconet.

RFCOMM The RFCOMM layer is used for the emulation of serial ports and is mainly useful for communication with mobile phones. BTnut supports sending of AT commands but does not implement the Object Exchange Protocol (OBEX) yet. Furthermore, RFCOMM allows for Internet access by means of the Point-to-Point protocol (PPP). TCP/IP over PPP over RFCOMM has not been tested on the BTnode, although the combination of Nut/OS and the BTnut Bluetooth Stack provides all required parts.

Scatternet Networking In addition to the standard Bluetooth protocols, BTnut provides multi-hop networking using Scatternets. The components of the BTnut Bluetooth Stack relevant for this are depicted in figure 4.6.

Connection Manager The connection manager is responsible for the formation of Scatternets (see figure 4.4) over all active BTnodes and supports different network topology construction policies for this. Implementations for the XTC algorithm [84], which forms a mesh network, and our own TreeNet [8], which maintains an acyclic tree, are available.

Multi-hop Data Transfer On top of the Scatternet, a simple pro-active flooding-based routing algorithm in the Multi-hop (MHOP) component allows to forward data to a central sink. Upon reception of a broadcast packet, a node stores the source address of the packet together with the ID of the neighbor from which it received the packet in a routing table. Later, a packet can be forwarded towards the destination according using the routing table.

Summary The BTnode platform with the BTnode Rev3 node can be used as a Deployment Support Network and is suitable to construct a distributed sniffer. Its low-power radio is compatible with all WSN nodes which use the CC1000 radio and it might even be possible to decode radio messages sent by a different radio that also uses FSK modulation. The 256 KB memory provides ample space to reliably buffer overheard network traffic until it can be delivered over the Bluetooth multi-hop network.

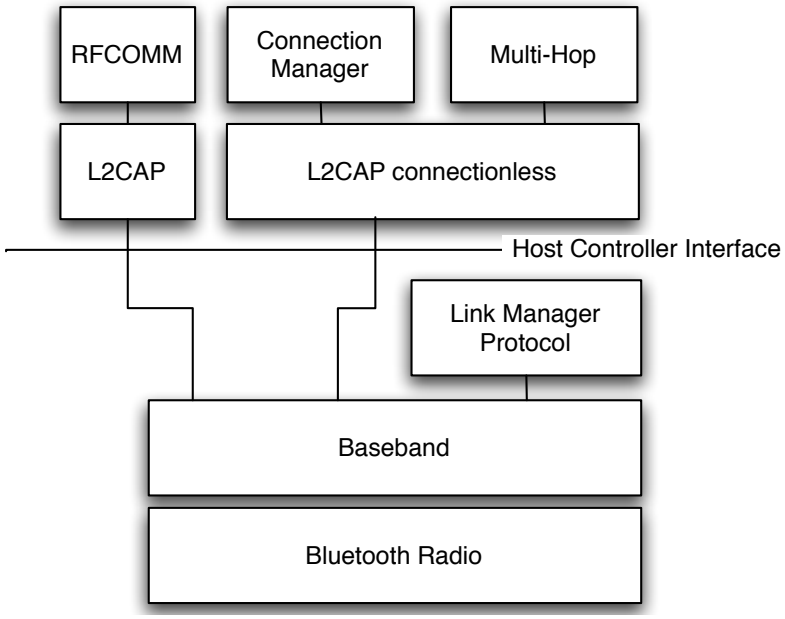


Figure 4.6.: Bluetooth Protocol Stack used for Scatternets in BTnut. The Connection Manager handles topology control and the Multi-Hop (MHOP) component provides basic routing support. Both make use of the connection-less L2CAP standard which provides datagram-like packet delivery over baseband connections.

4.3. Bluetooth Time Synchronization

By means of so-called *Scatternets*, Bluetooth provides the ability to construct robust wireless multi-hop networks. In this chapter we present a practical protocol for time synchronization of such Bluetooth multi-hop networks.

Time synchronization is a fundamental service in almost any computer network, including Bluetooth networks. Surprisingly, Bluetooth does not provide time synchronization as a service to applications even though time synchronization is needed internally, as its medium access is based on time-division multiple access (TDMA). However, the Bluetooth API provides a few functions that allow limited access to the internal clock that is maintained to control medium access. In this section, we propose and evaluate a practical algorithm for synchronizing multi-hop Bluetooth Scatternets which makes use of these functions. The algorithm provides a synchronization accuracy of few milliseconds across multiple hops with minimal communication overhead.

In the following, we first discuss related work on Bluetooth time synchronization before the core of our protocol is described. We then present implementation aspects and evaluate our proposal.

4.3.1. Related Work

There exists a large body of work on time synchronization in wireless networks and in particular wireless sensor networks [68]. However, all of these approaches send explicit synchronization messages, whereas our approach uses the synchronization primitives provided by Bluetooth to build a global synchronization protocol, thus minimizing additional communication overhead. Protocols based on the exchange of explicit synchronization messages would suffer from highly variable messages latencies of typical Bluetooth implementations. In [65], for example, round trip times in a simple two-node network varied between 30 and 230 ms with an 76 ms average. In [55], round-trip times between 20 and 120 ms have been observed in a similar setup.

Specific protocols for time synchronization with Bluetooth are rare. [16] describes an experiment in which a Piconet master sends broadcast messages to synchronize slaves among each other. As the broadcast message arrives almost simluatneously at all slaves, the reception event can be used to accurately synchronize the clocks of slaves. They report very good results for the accuracy among slaves in the order of 10 us. However, this approach can only synchronize slaves among each other, but not the master with the slaves as only the master can send broadcast messages. Hence, a Bluetooth Scatternet cannot be synchronized with this approach. A similar technique is used in [7] to synchronize the slaves of a Piconet. However, in addition they modify the firmware of the Bluetooth module to precisely measure the point in time when the master sent the broadcast message. Using this information, they can also synchronize the master with the slaves. Overall, they obtain a precision of few micro seconds in a single Piconet. While their approach could be extended to Scatternets, they do not consider this option. However, modifying the Bluetooth firmware is often impracticable or even impossible.

IEEE 802.11 also uses an internal clock to control medium access and specifies a Timing Synchronization Function (TSF) to synchronize the clocks of different nodes. In both infrastructure and ad hoc modes, 802.11 only supports single-hop networks. Therefore, TSF is a rather simple protocol. In infrastructure mode, the base station regularly broadcasts timestamped beacon messages and receivers adjust their clocks to the received time stamp. In ad hoc mode, every node broadcasts such beacon messages and receivers adjust their clocks to the sender with the latest time stamp. Huang et. al. [35] propose a simple extension to this procedure to im-

prove the scalability of TSF. Unfortunately, this scheme is not applicable to Bluetooth as precise time stamps can neither be sent or received.

4.3.2. Protocol Overview

Bluetooth is implemented as a *radio modem* with a well-defined command interface, the so-called "Host Controller Interface" (HCI), that is connected to the main processor via a serial interface (e.g., USB or RS232). The Bluetooth radio modem is itself a complex embedded computing device that contains, among others, a separate processor that runs parts of the Bluetooth protocol stack and a real-time clock to control medium access. Access to the internal state of the modem (e.g., the real time clock) is only possible via commands that are sent to the modem via the serial connection. In fact, Bluetooth provides two commands related to the real time clock: `HCI_Read_Bluetooth_Clock` to read out the current value of the real-time clock, and a second command `HCI_Read_Clock_Offset` to read some (but not all) bits of the current offset of the clock to the clock of a connected node.

Even though Bluetooth provides the above two commands as a foundation for time synchronization, the implementation of Bluetooth as a radio modem has far-reaching implications on the design of a synchronization protocol. Firstly, each network node has two clocks: the system clock and the Bluetooth clock. Typically, the operating system and applications use the (unsynchronized) system clock, whereas we intend to use the Bluetooth clock for synchronization among different nodes. That is, we need to synchronize the system clock with the Bluetooth clock in some way. Secondly, reading the Bluetooth clock (offset) is a costly and lengthy operation as it involves exchange of protocol messages between the main processor and the Bluetooth modem over a serial connection (in contrast, reading the system clock is cheap and fast). This implies that the Bluetooth clock (offset) should be read rarely. Moreover, execution time of a Bluetooth command is highly variable, as the reply to a command may be delayed by arriving data messages. That is, accurate synchronization of the system clock with the Bluetooth clock is non-trivial.

Figure 4.7 illustrates the design of our protocol which was inspired by the above observations. Three nodes are shown which are connected in a chain topology using Bluetooth Scatternets. Each node is indicated by a circle that contains the current values of the system clock and of the Bluetooth clock. All clocks are unsynchronized – they advance freely

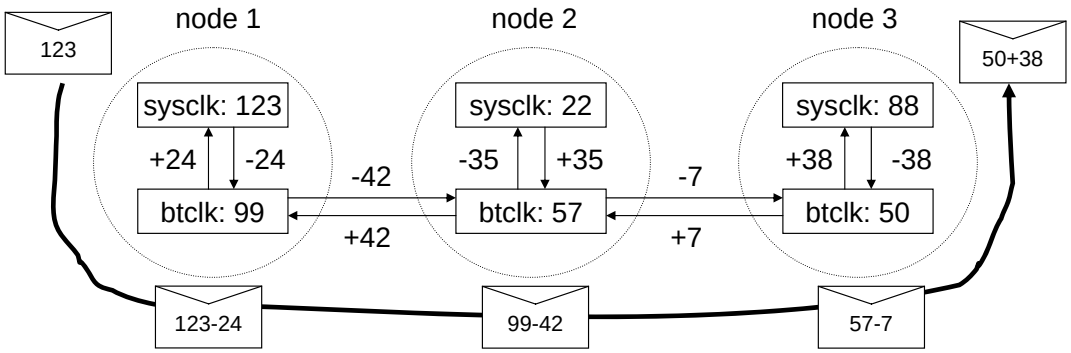


Figure 4.7.: Illustration of the synchronization protocol.

at their respective rates without being disciplined. However, each node maintains offsets between its system clock and the Bluetooth clock (and vice versa) and its Bluetooth clock and the Bluetooth clocks of connected nodes. These offsets are illustrated in figure 4.7 by tagged arrows that point from clock A to another clock B. By adding the arrow tag (i.e., the clock offset) to the value of clock A, one can obtain the corresponding value on clock B. We use the above Bluetooth commands to obtain the offsets between Bluetooth clocks of connected nodes and to obtain the offset between the system clock and the Bluetooth clock. As the offsets change infrequently (the clock drift of the various clocks is small compared to the required synchronization accuracy), these commands are invoked infrequently to update the offset values.

The thick arrow in figure 4.7 illustrates the exchange of a time-stamped message between node 1 and node 3 via node 2. When generating a new message, node 1 reads its system clock (i.e., 123) and includes this value as a time stamp in the message. Next, the time stamp is converted to the Bluetooth clock of node 1 by adding the respective clock offset -24. Then, the time stamp is converted to the Bluetooth clock of node 2 by adding the respective clock offset -42 and the modified message is sent to node 2. There, clock offset -7 is added to the time stamp to convert to Bluetooth clock of node 3. The message is then sent to node 3, where the clock offset +38 is added to transform to the system clock of node 3. The resulting time stamp 88 equals the value of node 3's system clock at the time when the message was generated in node 1. With this approach, all time stamps a node receives from different nodes will be synchronized in the sense that they refer to the time scale defined by its local system clock.

The above approach is sufficient for many applications (including our

distributed sniffer) and has two important advantages. Firstly, it does not require a designated node that acts as a time reference for other nodes and thus our protocol can easily deal with node failures and topology changes. Secondly, the difficulties of disciplining clocks are completely avoided.

Bluetooth Clock

The Bluetooth clock is a 28-bit counter with 0.3125 ms resolution and a mandatory maximum drift of $\pm$ 20 ppm. This results in an overrun every $2^{28} \times$ 0.3125 ms $\approx$ 1 day.

Each Bluetooth device has a unique 6-byte baseband address (BD_ADDR) similar to the medium access control address of Ethernet devices. The hopping sequence is a pseudo-random sequence of communication frequencies seeded with the BD_ADDR of the Piconet master device. Because of the frequency hopping, a special procedure called *inquiry* is required to discover other devices (i.e., their address and hopping sequence). During an inquiry, a device uses a special inquiry hopping sequence and doubles its hopping rate to rendezvous with other devices. As a result of an inquiry, the BD_ADDR and the difference between the local Bluetooth clock and the clock of the remote device are acquired. Based on this information, a node can calculate the hopping sequence of discovered nodes and is thus able to connect to these devices.

The clock offset to discovered devices is defined as bits 2-16 of the difference between the clock of the discovered device and the local clock. Similarly, the clock offset between two connected devices is specified as bits 2-16 of the difference between the clock of the slave node (CLKslave) and the clock of the master node (CLKmaster). With the reduced range (only bits 2-16) of these clock offsets, a maximum time interval of $2^{17} \times$ 0.3125 ms = 40.96 s with a resolution of 1.25 ms can be specified.

Offset between System and Bluetooth Clock

Measuring the offset between the system clock and a Bluetooth clock is non-trivial as reading the Bluetooth clock requires sending a command message to the Bluetooth modem over the serial interface between the main processor and the Bluetooth modem and receiving a reply message (a so-called event) over the serial interface that contains the requested clock value.

This protocol is illustrated in figure 4.8 (a). First, the command is sent over the channel connecting the main processor (MCU) with the Blue-

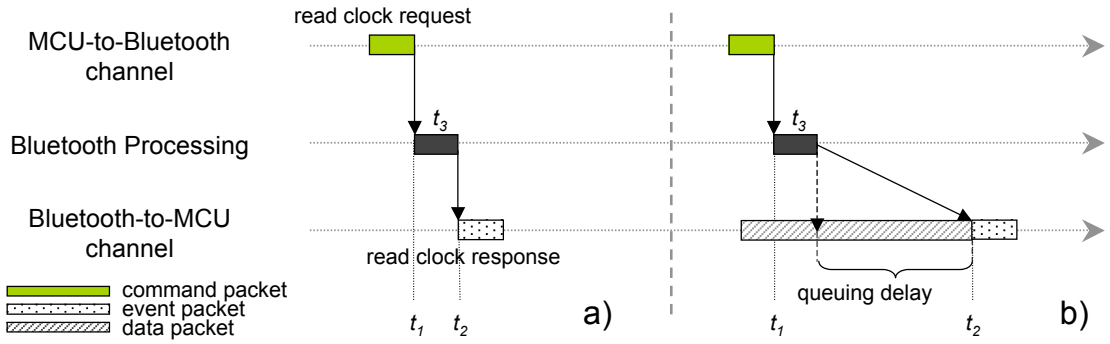


Figure 4.8.: Reading Bluetooth clock over Host Controller Interface. a) no parallel data traffic, b) incoming data packet.

tooth modem ("MCU-to-Bluetooth channel"). After the Bluetooth modem has received the last bit of this message over the serial line, the command will be processed. At some (unknown) point in time, the actual readout of the clock will be performed, before the reply message is generated and sent to the main processor via the "Bluetooth-to-MCU channel".

We could use a typical round-trip-time measurement to compute the clock offset between the system clock and the Bluetooth clock. Here, we would use the system clock to measure the point in time t_1 when the last bit of the command has been sent and the point in time t_2 when the first bit of the reply has been received. The clock offset could then be approximated by the returned Bluetooth clock value t_3 minus $(t_1 + t_2)/2$.

Unfortunately, the reply from the Bluetooth module may be significantly delayed if the Bluetooth module has received a data message via radio and is sending this message to the main processor as illustrated in figure 4.8 (b), such that the communication channel ("Bluetooth-to-MCU channel") is blocked. The resulting highly variable delay $t_3 - t_2$ results in significant errors when using the above approach.

Likewise, processing of the command in the Bluetooth modem may be delayed if the modem is busy receiving data. However, we performed experiments that confirmed that the offset $t_3 - t_1$ is much more stable than $t_3 - t_2$ even under heavy communication load. Therefore, we use $t_3 - t_1$ as the offset between the system clock and the Bluetooth clock in our protocol. Still, there are occasional outliers that are substantially larger than the average clock offset. To remove these outliers, we apply a simple median filter. Instead of using $t_3 - t_1$ as the clock offset, the median filter remembers the last n measured offsets and returns the median value among them. The evaluation in section 4.3.4 will show the effectiveness

of this approach for small n.

Note that the above approach results in a systematic error which equals the time interval between transmission of the last bit of the command message and the actual readout of the Bluetooth clock. Assuming that the error is mainly a function of the Bluetooth implementation, it will cancel out in the end-to-end synchronization protocol illustrated in figure 4.7 if both sender and receiver use identical Bluetooth hardware. The reason for this is that we transform each time stamp twice between system clock and Bluetooth clock: once at the sending node and once at the receiving node. Since these transformations are in reverse direction (i.e., system clock to Bluetooth clock on the sending node and Bluetooth clock to system clock on the receiving node), the error will cancel out.

Offset between Bluetooth Clocks

To obtain the offset Δ between the Bluetooth clocks of two connected nodes, the `HCI_Read_Clock_Offset` command can be used. However, the clock offset returned by this command only contains bits 2-16 of the clock difference. We therefore need to reconstruct the complete clock difference. For this, a local Bluetooth time stamp is sent over an established connection as shown in figure 4.9. The method for reconstructing the missing bits of Δ described below assumes that the transmission latency d of the message is less than the clock offset range of 40.96 s, which is a reasonable assumption.

In the following, we will refer to the difference between two Bluetooth clocks as *clock difference* Δ and refer to the lower part of this clock difference Δ returned by Bluetooth commands as *clock offset*. Also, we assume that all variables hold integer multiples of a Bluetooth clock tick of 0.3125 ms. We can therefore express the assumption that d is below 40.96 s as follows

$$0 \leqslant d < 2^{17} \tag{4.1}$$

In the following we will use the notation V_{b_c} to refer to the integer value of V where bits with index b-c in the binary representation are preserved and all others are set to zero. The least significant bit has index 0. For example, $15_{1\_2} = 6$.

From the `HCI_Read_Clock_Offset` command, we obtain bits 2-16 of the clock difference Δ:

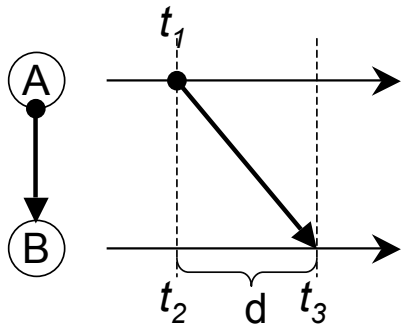


Figure 4.9.: Sending a Bluetooth time stamp from node A to node B.

$$\Delta_{2\_16} = (CLKslave - CLKmaster)_{2\_16}$$

Let us assume that node A in figure 4.9 is in the slave role and sends the current value t_1 of its clock CLKslave to node B which records the time t_3 of its local clock CLKmaster at reception.

Node B can calculate an approximate clock difference Δ' as:

$$\Delta' := t_1 - t_3 \tag{4.2}$$

As t_3 represents the time when the message was sent plus the (unknown) transmission delay d, (4.2) can be reformulated as follows:

$$\Delta' = t_1 - (t_2 + d) = (t_1 - t_2) - d = \Delta - d \tag{4.3}$$

As we know Δ' and Δ_{2_16}, we can use (4.3) to calculate Δ_{17_27} as follows. Since $0 \leq d < 2^{17}$ by assumption, (4.3) implies that either $\Delta'_{17\_27} = \Delta_{17\_27}$ or $\Delta'_{17\_27} = \Delta_{17\_27} - 2^{17}$. $\Delta'_{17\_27} = \Delta_{17\_27}$ can only hold iff $\Delta'_{2\_16} \leq \Delta_{2\_16}$, otherwise it would follow that $\Delta' > \Delta$ in contradiction to (4.3). In summary, we can compute the missing bits of Δ using the following equation:

$$\Delta_{17\_27} = \begin{cases} \Delta'_{17\_27} & \text{if } \Delta'_{2\_16} \leqslant \Delta_{2\_16}, \\ \Delta'_{17\_27} + 2^{17} & \text{otherwise.} \end{cases} \tag{4.4}$$

If node A would have been in master mode, the following analogous equation has to be used:

$$\Delta_{17\_27} = \begin{cases} \Delta'_{17\_27} & \text{if } \Delta'_{2\_16} \geqslant \Delta_{16\_2}, \\ \Delta'_{17\_27} - 2^{17} & \text{otherwise.} \end{cases} \tag{4.5}$$

4.3.3. Implementation

We implemented our time synchronization protocol on the BTnode Rev. 3 nodes described in section 4.2. Unfortunately, the implementation of the commands `HCI_Read_Clock_Offset` and `HCI_Read_Bluetooth_Clock` on the Zeevo Bluetooth module suffers from several bugs, which we had to work around when implementing our protocol. While the `HCI_Read_Clock_Offset` command was always present in the Bluetooth specification, the `HCI_Read_Bluetooth_Clock` command was introduced by Bluetooth specification 1.2 in 2003. The Zeevo Bluetooth module on the BTnode was sold as a pre-1.2 version. Although it supports the required commands for our time sync approach, some commands do not follow the specification or are not properly implemented. We provide details on the bugs and our work-arounds below.

`HCI_Read_Bluetooth_Clock`

This command is supposed to return the value of the local Bluetooth clock or the value of the Bluetooth clock of a connected Piconet master (depending on the command parameters). However, our Zeevo modules always return the value of the local Bluetooth clock, the clock of the master could not be read. Further, the returned Bluetooth clock value was expressed as a multiple of 1.25 ms instead of the specified 0.3125 ms. Finally, we noticed occasional significant outliers when analyzing the computed offsets between system clock and Bluetooth clock The reason for this is a bug in the implementation of the `HCI_Read_Bluetooth_Clock` command, which returns the same Bluetooth clock reading twice in 1% of all cases. As the command execution takes about 10 ms in our configuration, two `HCI_Read_Bluetooth_Clock` commands cannot have been issued and answered within the clock resolution of 1.25 ms. As it turned out, the second value was a duplicate of the first, and we resorted to reading the Bluetooth clock twice and using the second value only if it was different from the first.

`HCI_Read_Clock_Offset`

This command is supposed to return the current clock offset to a connected device. However, once a connection between two BTnodes has been established, the offset returned by this command never changes even though the two clocks drift apart. Only after closing and re-opening a con-

nection the returned clock offset changed. That is, the command always returns the clock offset at the time when the connection was established.

As periodic disconnects are no option in many applications, another way to read the clock offset was needed. After some experimentation, we noticed that a connected device is also reported by an Bluetooth inquiry. This comes as a surprise as inquiries are generally used to find new devices and not to collect information about already connected neighbors. As inquiries also return the current clock offset, periodic inquiries can be used to update the clock offsets to connected neighbors. However, the clock offset returned by an inquiry is slightly different from the value returned by `HCI_Read_Clock_Offset`. The latter always returns CLKslave - CLKmaster no matter if the Bluetooth device is a master or a slave. The inquiry returns the same value only if the invoking Bluetooth device is a master. If invoked by a slave, the returned value is $2^{17}-$(CLKslave - CLKmaster).

4.3.4. Evaluation

To evaluate our approach, we first study the accuracy of synchronization between system and Bluetooth clock in the presence of parallel data transmissions. Then we measure the synchronization error within an 8 node multi-hop Scatternet. We also show some preliminary results of this approach running on a linux laptop computer with a built-in Bluetooth module.

Reading the Bluetooth Clock under Load

We evaluate the accuracy of the approach to measure the offset between the system clock and the Bluetooth clock. We consider a Bluetooth Piconet of two node A and B. The speed of the serial connection between the main processor and the Bluetooth modem was set to 115200 baud. After B has connected to A, A will start reading its Bluetooth clock 1000 times at regular intervals. A will then signal B to start sending data messages. A will discard these messages, but continues to read out the Bluetooth clock. After 1000 readouts, A signals B to stop sending data. A continues to read out its Bluetooth clock for another 1000 times.

At each readout, A records the time of the system clock after the last bit of the command has been sent to Bluetooth and the returned value of the Bluetooth clock. That is, A records a data point (system clock, Bluetooth clock) for each readout. We then fit a line to these 3000 data

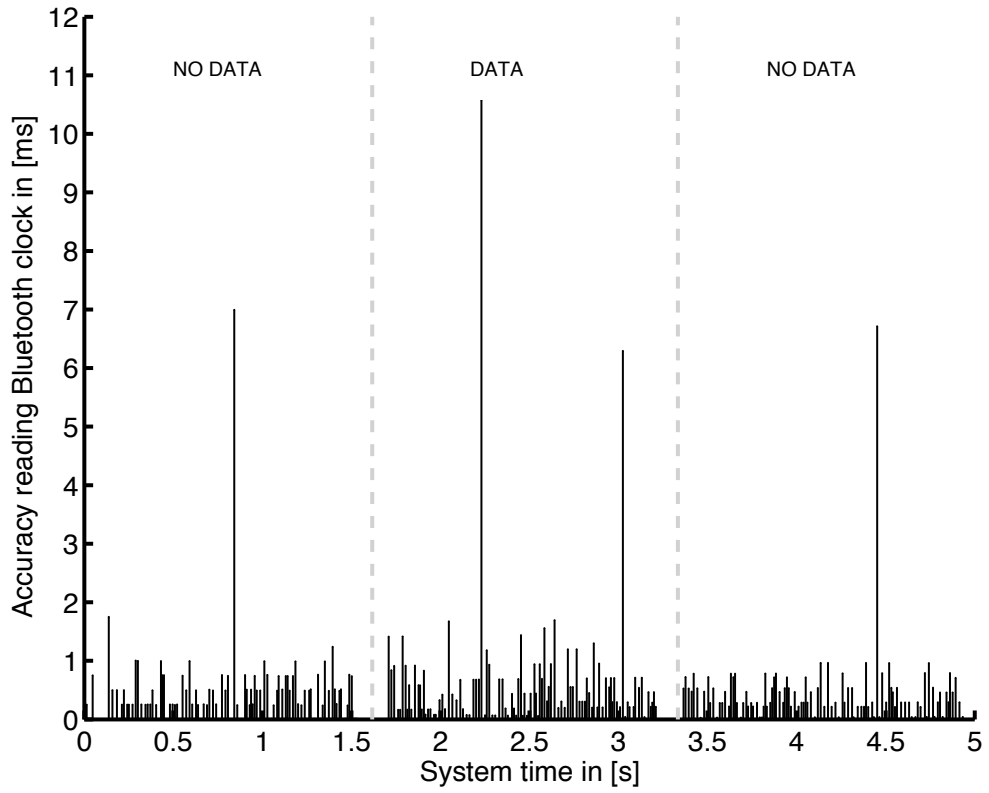


Figure 4.10.: Accuracy of Bluetooth clock read out with and without data traffic.

points using linear regression. This line approximates the ground-truth offset between the two clocks. We consider the distance of a data point from the regression line as a measure of the accuracy of that data point.

Figure 4.10 shows the accuracy for the first 100 readings of each block (no data, data, no data). The maximum error on accuracy was 15 ms. To analyze the impact of data transmission, we plotted the empirical cumulative distribution function (ECDF) for ”data“ and ”no data“ separately as shown in figure 4.11. The distribution for ”no data“ shows smaller errors than for the ”data“ section, but this results mainly from having less outliers compared to the ”data“ section.

To further improve accuracy, we employed a median filter as described in section 4.3.2, which outputs the median of the last n samples. Figure 4.12 shows the average and the maximum error for different values of n over all samples. For all 3000 samples with and without parallel data traffic, the maximum error for n=5 is below 2 ms.

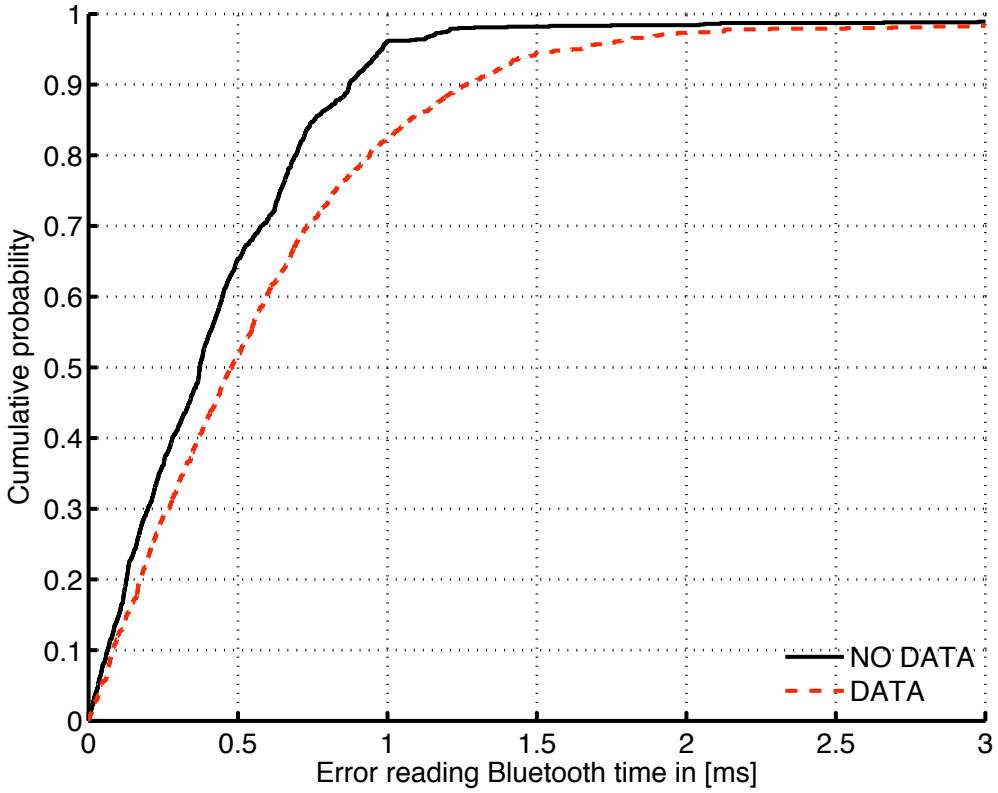


Figure 4.11.: ECDF of accuracy with and without data traffic.

Scatternet Synchronization Error

As the Bluetooth clocks run unsynchronized and the clock offset is only available with a 1.25 ms resolution, we expect the error between two connected nodes to be less than this value. We set up 8 BTnodes in a chain topology in which each node but the end nodes act as a master/slave bridge, effectively forming a Scatternet of 7 inter-connected Piconets. All nodes are connected to an 8-channel logic analyzer with 1 us time resolution. After the Scatternet has been established, the first node in the chain periodically sends a time-stamped message along the chain of nodes, applying the synchronization algorithm described in section 4.3.2 to synchronize the time stamp. When receiving the message, a node will set a timer to expire at system time $t + C$, where t is the (synchronized) time stamp contained in the message and C is a constant offset. When the timer expires, the node toggles the I/O pin which is connected to the logic-analyzer. Ideally, all nodes should toggle their pins at exactly the same point in time. However, synchronization errors will cause nodes to toggle their pins at slightly different points in time. Using the logic analyzer, we measure the time between the first node in the chain toggling its

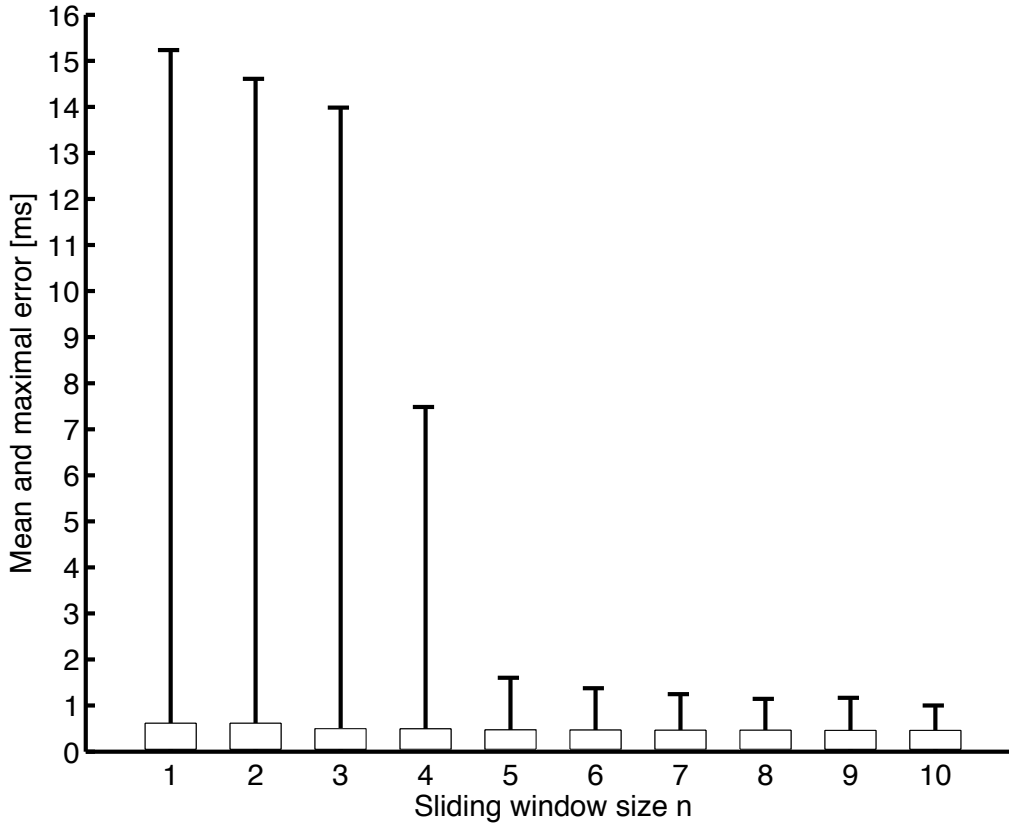


Figure 4.12.: Mean and maximum clock reading error for median filter with window size n.

pin and every other node in chain toggling its pin. This amount of time is the synchronization error.

In the experiment, the BTnodes update their clock offsets every 5 minutes using a Bluetooth discovery. The experiment runs for 2 hours, resulting in 712 accuracy measurements for all 8 nodes. Figure 4.13 shows the average and maximum synchronization error for each node. The mean error for the last node in the chain is 5.47 ms $\pm$ 2.25 ms and the maximum error is 11.35 ms.

Other Platforms

The BTnodes used in the evaluation are only a single example for devices which support our time synchronization protocol. Often, a more powerful device such as a laptop computer is used to store collected data and to provide a time reference for a whole Scatternet. For our protocol to work, a device needs to support the `HCI_Read_Bluetooth_Clock` and `HCI_Read_Clock_Offset` commands which are available, e.g., in the BlueZ [94] Linux Bluetooth Stack. We tested an Apple PowerBook

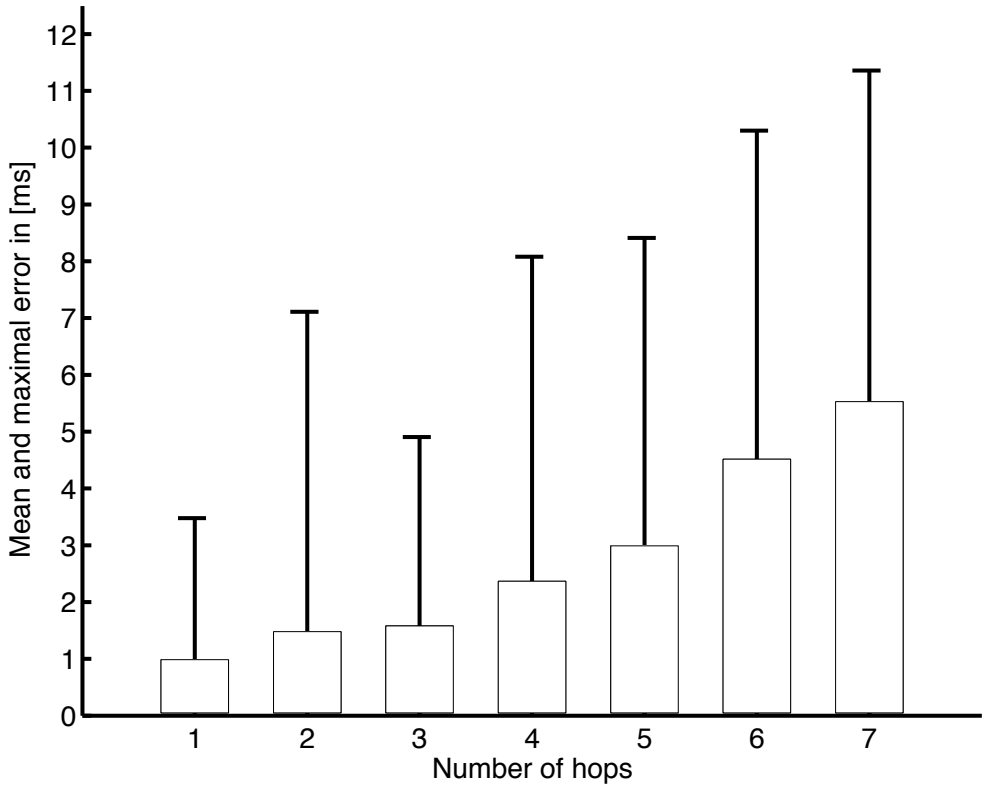


Figure 4.13.: Time synchronisation error for a 7-hop Scatternet.

G4 12" with an embedded Cambridge Silicon Radio (CSR) chipset connected over USB, running the Ubuntu 6.10 Linux distribution with the default 2.6.17 kernel. Both commands are available on the CSR Bluetooth module. However, the `HCI_Read_Clock_Offset` only works correctly if the device is in a slave role. In master role, the clock offset was not updated similar to the bug with the Zeevo module (see section 4.3.3). We repeated the read Bluetooth clock test and calculated the accuracy as in section 4.3.4. In this configuration without median filter, the maximum error on accuracy was 4 ms. The mean error was 0.15 ms with a standard deviation of $\pm$ 0.24 ms.

4.3.5. Discussion

The proposed time synchronization results in an average time sync error in the order of several milli seconds over multiple hops. For SNIF, to correctly order two packets received by two neighboring DSN nodes, the time synchronization error should be lower than half the duration of a WSN packet. An example TinyOS packet for the Mica2 node with a 10 bytes payload results in a total of 25 bytes (6 bytes preamble, 2 bytes start-

of-packet indicator, 5 bytes header and a 2 bytes CRC). At 19200 baud, the total packet time is 10.4 ms which is more than twice the mean time synchronization error between nodes with five hops in between. Note that with low-power listening protocols such as WiseMAC [23] or B-MAC [60], a significantly longer preamble has to be used which reduces the required accuracy for the time synchronization.

4.4. DSN Sniffer

The first step in the passive inspection approach is the collection of radio traffic. For this, a distributed generic sniffer is needed that can be configured for different MAC protocols used by WSN applications. The BTnode platform provides the functionality for this: The BTnodes are equipped with two independent radios, can form a deployment support network with a reliable Bluetooth Scatternet for packet forwarding, and allow for Bluetooth-based time synchronization.

In addition, the DSN nodes need a receive-only implementation of the physical (PHY) and MAC layers in order to overhear sensor network traffic. Due to the lack of a standard protocol stack, many variants of PHY and MAC are in use in sensor networks. Hence, we need a flexible implementation that can be easily configured for the sensor network under inspection.

Our generic PHY implementation supports configurable carrier frequency, baud rate, and checksumming parameters. We assume that the sensor network uses a single frequency for communication (which is the case with most current implementations) such that a single-channel radio is sufficient to overhear WSN traffic.

Regarding MAC, we exploit the fact that – regardless of the specific MAC protocol used – a radio packet always has to be preceded by a preamble and a start-of-packet (SOP) delimiter to synchronize sender and receiver. In our generic MAC implementation, every DSN node has its WSN radio turned to receive mode all the time, looking for a preamble followed by the SOP delimiter in the received stream of bits. Once an SOP has been found, payload data and a CRC follow. Also, a timestamp is recorded when the SOP is detected. This way, DSN nodes can receive packets independent of the actual MAC layer used. Received packets are then forwarded by the Multi-Hop service of BTnut (see section 4.2.2) to the SNIF sink.

Figure 4.14 shows an excerpt of a sample configuration file for inspect-

ing a TinyOS 1.x application running on MICA2 motes. The first five lines set the carrier frequency of the WSN radio to 868.000 MHz and a data rate of 19200 bits/second, and instruct the packet sniffer to check for a start-of-packet sequence of 0xcc33. The 16 bit CRC-CCITT polynomial $x^{16} + x^{12} + x^5 + 1$ (0x1021) is used as checksum algorithm.

4.5. Packet Decoder

Since no standard protocols exist for sensor networks, we need a flexible mechanism to decode overheard packets. As most programming environments for sensor nodes are based on the C programming language or a dialect of it (e.g., nesC for TinyOS), it is common to specify message contents as (nested) C structs in the source code of the sensor network application. Our packet decoder uses an annotated version of such C structs as a description of the packet contents. This way, the user can copy and paste packet descriptions from the source code.

The configuration of the packet decoder consists of some global parameters (such as byte order and alignment), type definitions, and one or more C structs. One of these structs is indicated as the default packet layout. Note that such a struct can contain nested other structs.

Consider figure 4.14 for an example, which describes link advertisement packets used by the Multihop routing service implemented in ESS [30]. Line 20 defines the struct `TOS_Msg` as the default packet layout. The LinkAdv PDU used by ESS, is encapsulated in the field `TOS_Msg.data`, but only if the `TOS_Msg.type` is equal to the constant `LinkAdvType`. Line 36 specifies the contents of `TOS_Msg.data` depending on the value of the field `TOS_Msg.type`. Note that an encapsulated structure can itself also contain another encapsulated structure.

Our description language allows to specify variable sized arrays. If an element of a message is used as array size (e.g., `TOS_Msg.length`), then the value of this field in the overheard message denotes the number of elements in the array to decode. If an array without a size is given (e.g., `LinkAdvertisement.links`), the size of the array is inferred from the total packet size.

At startup of SNIF, the configuration file is parsed and the default packet type is investigated. If the default packet type is of fixed size, the packet size is computed. Otherwise, size and position of the packet length indicator (e.g., `TOS_Msg.length` in the example) is computed. This information, along with the parameters for the physical layer are then broadcast

```
// PHY + MAC parameters
cc.freq = 868000000;
cc.baud =19200;
cc.sop = 0xcc33;
cc.crc = 0x1021;

// encoding: endianness + alignment
enc.endianness = " little ";
enc.alignment = 1;

// type definitions and constants
typedef uint16_t mote_id_t;
typedef uint8_t quality_t ;
struct link_quality_t {
    mote_id_t id;
    quality_t quality ;
};

// Default packet type
default.packet = "TOS_Msg";

// TOS_Msg definition
struct TOS_Msg {
    uint16_t addr;
    uint8_t type;
    uint8_t group;
    uint8_t length;
    int8_t data[length]; // variable payload size
    uint16_t crc;
};

// LinkAdvertisement packet type
const int LinkAdvType = 2;
// LinkAdvertisement packet definition

struct LinkAdv : TOS_Msg.data (type == LinkAdvType) {
    mote_id_t id;
    struct link_quality_t links []; // var. size
};
```

Figure 4.14.: A SNIF configuration file.

to all DSN nodes, allowing them to correctly receive WSN traffic.

4.6. Data Stream Processor

The DSN sniffer outputs a stream of overheard packets that needs to be analyzed to detect problems in the WSN. To enable an efficient deployment process, this analysis should be performed *online*, allowing an engineer to go out and study and fix affected nodes while the problem is still present.

Given these preconditions, we decided for a data stream processor [2] to perform online analysis of packet streams. In the remainder of this section we present the particular data stream model we use, the representation of packets as elements of a data stream, and specific operators provided by the framework for analyzing packet traces.

4.6.1. Data Streams

A data stream is an unbounded sequence of records. Various data stream management systems (DSMS) have been proposed as generic frameworks to process data streams. Mainly motivated by practical considerations (Java as implementation language, stability, open-source availability) we chose the PIPES data stream processor [15] for use with SNIF.

Three basic abstractions are provided: *sources* that produce data streams, *sinks* that consume data streams, and *operators* that modify data streams. An operator is essentially both a source and a sink. Sinks and operators can *subscribe* to sources and operators, such that a data stream output by the subscribee acts as input for the subscriber. That is, sources, operators, and sinks form a directed *operator graph* with data streams flowing from sources through operators towards sinks. In SNIF, we model each DSN node as a data stream source. An operator graph (being executed on the DSN sink) processes these data streams to detect indicators for problems, and sink nodes inform the user of detected indicators.

4.6.2. Records

A data stream record (i.e., an element of the data stream) is a *typed* and *time-stamped* list of attribute-value pairs. The type of a record essentially indicates what attributes can be found in that record. The time stamp indicates when that record was generated according to a global time scale.

In our implementation, an attribute is a string and each value an object. Two built-in attributes holding record type and time stamp are always available. The DSN produces records of type `Packet` which contain additional attributes holding the contents of an overheard packet (compare with section 4.5). The syntax of the latter attributes follows C syntax for accessing the field of a structure (e.g., `TOS_Msg.addr` in Figure 4.14). The fields of the encapsulated structures can be accessed recursively (e.g., `TOS_Msg.data.id`). The length of an array can be accessed by appending `.length` to the array field as in `TOS_Msg.data.links.length`.

All attributes referring to packet contents are implemented as *virtual attributes*. Whenever an attribute referring to packet contents is accessed, the packet decoder is invoked to extract the requested field from the raw packet as captured by the DSN.

4.6.3. Basic Operators

Our framework provides a number of basic data stream operators that are sufficient to implement SQL with time windows, but without joins. These operators can be configured by parameters that are either attribute names (prefixed by *attr*), predicates (prefixed by *pred*), or functions over record(s) (prefixed by *func*).

Mapper(attr1Old, attr1New, attr2Old, attr2New, ...) Renames attribute *attrXOld* to *attrXnew* in each record.

ArrayIterator(attrArray) Provides access to array elements by iterating over the array contained in attribute *attrArray*. The operator creates N copies of each input record, where in the i-th copy the array is replaced with array element i. When applied to `LinkAdvertisement.links` in figure 4.14, for example, we obtain one record for each pair of neighbors.

Union Merges the records of all subscribed data streams into one, such that the output data stream has non-decreasing time stamps.

Filter(pred) Drops all records from the data stream for which the given predicate evaluates to false.

TimeWindowAggregator (window, funcAggr, funcHash, attrGroup) Computes an aggregate value over a time window of size *window*. Within the window, records are grouped by the contents of the attribute *attrGroup*. In each group, duplicate records are removed by applying the collision-free hash function *funcHash* to records. If records hash to the same value, only the one with the latest time stamp is kept. Whenever window contents change, the aggregation function *funcAggr* is applied to the window contents. It creates an output record for each group, containing at least an attribute holding the group id and an attribute holding the aggregated value for that group.

Besides common aggregation functions such as counting the number of records as well as computing the sum, average, variance, minimum, and maximum of a given attribute, we provide an aggregation function *ratio(attr) = count / (max(attr) - min(attr) + 1)* with two notable applications. Firstly, when applied to a packet sequence number attribute, *ratio* computes an indicator of observation quality as the fraction of messages sent by a sensor node that can be overheard by the DSN. Secondly, when applied to the time stamp attribute, *ratio* can also be used to estimate congestion (i.e., packets per time unit).

4.6.4. Sources

DSNSource This data stream source is SNIF's interface to the DSN. The individual streams from the DSN nodes are merged into a single stream using the Union operator. Also, duplicate packets (resulting from two or more DSN nodes overhearing the same sensor node) are removed, using a Filter operator with the *predDistinct(window)* predicate that drops a packet if a copy has been observed before within a time window of size *window*.

EmSource This source uses link dump files or the EmStar framework [28] as input, but it is otherwise identical to DSNSource.

4.6.5. Application-specific Operators

This section presents specialized operators that assist in detecting the problems described in section 2.2. Although the basic operators described in the previous section allow to specify some of the indicators from section 3.3.2 - e.g. *TimeWindowAggregator* can be used to count the number

of packets per node for a certain amount of time and if a node does not send any packets, it can be consider dead - other indicators require more complex logic, e.g., the detection of a loop in the routing path. Such operators can be specific to the domain of wireless sensor networks or even for a concrete WSN application. We call such operator application-specific.

Development of such application-specific operators is the core task to customize SNIF for detection of a specific deployment problem. The primary challenge here is to deal with incomplete information due to i) the DSN failing to overhear packets, and due to ii) information that would be needed to detect a problem not being explicitly included in messages. For example, the SeqReset operator in the following section implements heuristics to tell apart reboots from sequence number wrap-around even in case of lost messages.

Albeit such operators might be combined from other basic operators, we also allow to specify operators in an imperative programming language, which is Java in our implementation and allows to handle state in an intuitive way.

SeqReset(attrSrc, attrSeq, maxSeq) This operator detects node reboots exploiting the fact that the sequence number contained in beacon messages will be reset after reboot. Parameters are the attribute name holding the source address *attrSrc* and the sequence number *attrSeq*, as well as the maximum sequence number value *maxSeq* before a wrap around. The main challenge here is to tell apart a wrap-around of the sequence number from reboot in case of lost beacon messages. The algorithm in Fig. 4.15 maintains a data structure *n* that holds for each node *i* the last sequence number *n[i].seq*, last time stamp *n[i].t*, and minimum interval *n[i].ival* between successive beacons. Whenever a beacon with source address *src*, sequence number *seq*, and time stamp *t* is received, the algorithm checks if *seq* is smaller than the last sequence number *n[src].seq* seen for this node. If the last sequence number is far apart from maximum sequence number *maxSeq* (parameter C must be selected such that loss of C consecutive beacon messages is highly unlikely), then *src* has rebooted. Otherwise, we apply an additional check to distinguish reboots from wrap-arounds with lost messages. In case of a wrap-around, the time between the last and current beacon messages *t - n[src].t* must be greater than or equal to the minimum beacon interval *n[src].ival* times the number of beacon messages that were lost plus one *(seq - n[src].seq) % maxSeq*. If a node reboot is detected, a record of type `SeqReset` is emitted containing the

```
on receive beacon(src, seq, t):
  if (exists n[src]) {
    if (seq < n[src].seq) {
      if (n[src].seq < maxSeq − C)
        emit reboot(src, t);
      else if (t − n[src].t < (seq − n[src].seq) % maxSeq ∗ n[src].ival)
        emit reboot(src, t);
    }
    n[src].ival ← min(n[src].ival, (t − n[src].t) / (seq − n[src].seq));
  } else {
    n[src].ival ← ∞;
  }
  n[src].seq ← seq;
  n[src].t ← t;
```

Figure 4.15.: SeqReset operator

address of the node.

PacketTracer(attrOrigin, attrDst, predSameFlow) In sensor networks, messages are often not acknowledged at the link layer. In these cases, the sender MAC address is not included in the message. However, as noted in section 3.3.2, some indicators require the per-hop source address. The operator PacketTracer reconstructs the per-hop source address for the case that a message is relayed across multiple hops (e.g., a sensor reading being transmitted from a node to the sink). We assume that all messages of such a flow contain the address of the originator (attribute *attrOrigin*) (e.g., as context information on where a sensor reading has been generated). Also, it must be possible to decide whether two given messages belong to the same flow (e.g., using message contents or an end-to-end sequence number) as implemented by the predicate *predSameFlow*. The per-hop destination address (attribute *attrDst*) must be included in any message to identify the receiver.

To recover the missing source MAC address, the fact is exploited that the per-hop destination address of a message will equal the per-hop source address of the next message in the flow. The operator maintains a table with the last message in each active flow. When a message cannot be associated to a flow in the table using *predSameFlow*, then a new flow is created and the packet is stored. The per-hop source address equals the originator address in this case. Otherwise, if the flow already exists, the per-hop source address of the message equals the per-hop destination address in the packet stored for that flow in the table. A special case are

```
on receive data(dst, seq, orig, t):
  if (exists p[seq|orig]) {
    if (p[seq|orig].dst = dst) {
      emit retransmission(dst, seq, orig, t);
    }
    src ← p[seq|orig].dst;
    p[seq|orig].dst ← dst;
  } else {
    src ← orig;
    p[seq|orig].dst ← dst;
  }
  emit data(src, dst, seq, orig, t);
```

Figure 4.16.: PacketTracer operator.

retransmitted packets, where the destination address in the packet equals the destination address stored in the table. The packet in the table is then replaced with the new packet. The operator copies incoming records to the output but appends an attribute holding the discovered source address. Another attribute is added indicating whether or not this packet is a retransmission.

Note that if the DSN fails to overhear one or more of these messages in a row, the destination address of the last overheard packet will be used as the source of the next overheard packet. As a positive side effect of this, we obtain a continuos message flow (i.e., a sequence of messages where the destination address of a message equals the source of the next message in the flow) even if the DSN fails to overhear messages. The operators PathAnalyzer and TopologyAnalyzer operators described below rely on this feature to deal with missing messages.

PathAnalyzer(attrSrc, attrDst, sinkAddr) This operator finds sensor nodes that have a routing path to the sink with MAC address *sinkAddr*. We assume that messages are routed from nodes to the sink along the edges of a spanning tree. Here, a path between a node and the sink exists if a sequence of packets $p_1, ..., p_n$ with increasing time stamps has been observed, such that the source address of p_1 equals the address of the node, the destination address of p_n equals *sinkAddr*, and the destination address of p_i equals the source address of p_{i+1}. The latency of this path is defined as the difference of the time stamps of p_1 and p_n. The names of the attributes holding the source and destination MAC addresses must be given by *attrSrc* and *attrDst*. Note that the above notion of path existence

```
on receive data (src, dst, t):
  if (dst ∈ n[src].desc) {
    emit routingloop (src, dst, t);
    remove dst from n[src].desc;
  }
  desc ← (src, t) ∪ n[src].desc;
  foreach (dn, dt) ∈ desc {
    if (dst = sink) {
      if (dn ∉ n[sink].desc) {
        emit goodpath (dn, t);
      } else if (dt > n[sink].desc[dn]) {
        emit goodpath (dn, t);
      }
    }
    n[dst].desc ← n[dst].desc ∪ (dn, max (n[dst].desc[dn], nt));
  }
```

Figure 4.17.: PathAnalyzer operator.

does not imply that packets are actually successfully delivered, but packet loss will result in increased path latency.

To implement this approach, a data structure is maintained that contains for each node A a set $\{(j, t_j)\}$, where j is a node that has a path to A according to the above definition and t_j is the time stamp of the latest message sent by j. We call these nodes j descendants of A. When a message p is observed with source address A, destination address B, and time stamp t, node B is inserted into the data structure if it did not exist before. If B is already listed as a descendant of A then a routing loop exists and B is removed from the A's descendants. Then, the set of descendants of B is updated to include (A, t) and all descendants (j, t_j) of A. Whenever a new descendant is added to the sink or the time stamp of an existing descendant of the sink is updated with a later value, a record with type `GoodPath` is emitted containing the MAC address of the descendant and the latency of the path from the descendant to the sink. In case of a routing loop, a record of type `RoutingLoop` is emitted holding the addresses of sender and receiver of the message causing the loop.

TopologyAnalyzer(window, sinkAddr) This operator implements a heuristic to detect network partitions. Note that partition detection is a non-trivial problem as we do not know the exact set of neighbors of each node. We assume that messages are sent from nodes to the sink (with address *sinkAddr*) along the edges of a spanning tree. Here, a partition is a

```
on receive data (src, dst):
  n[dst].nb ← n[dst].nb ∪ src;
  reset timeout (dst, src);

on timeout (dst, src):
  remove src from n[dst].nb;

on receive nodestate (src, state):
  if (state = ''dead'') n[src].nb ← ∅;

periodically:
  DFS (n, sink);
  foreach unvisited node nn
    emit partitioned (nn);
```

Figure 4.18.: TopologyAnalyzer operator.

special case of "no path to sink", where node failures lead to a separation of the node from the sink. In fact, only after a "no path to sink" error has been reported for a node, should the output of TopologyAnalyzer be used to decide whether the reason for this error is caused by a network partition.

To detect such partitions, we construct an approximate view of the network topology. For each node, we maintain a list of recently used (in terms of a time window with length *window*) tree parent nodes by extracting sender and receiver addresses from overheard packets. Also, when a node failure is observed, the respective node is marked as failed in this data structure. A node is considered partitioned if there is no path from that node to the sink in this data structure that uses only nodes that have not failed. In practice, a depth-first search is performed whenever the data structure is modified.

The operator requires two input data streams: a stream of overheard packets and a stream of "node failure" events generated by another operator graph. The output of the operator consists of `Partition` records indicating whether or not a node is currently partitioned from the sink. These records contain a node address, a flag indicating if that node is partitioned or not, and the addresses of any failed upstream nodes that caused the partition.

StateDetector(attrGroup, funcEval) While the above operators are used to detect a variety of passive indicators, it is often not trivial to infer the original problem from these indicators. To help with the root cause anal-

ysis described in the following section, SNIF provides the StateDetector operator.

This operator groups records by record type and by the value of the attribute given by *attrGroup*. For each group, the operator stores the latest record. Whenever a new record is inserted into a group, the function *funcEval* is invoked for that group, with the set of stored records as parameter. The evaluation function outputs a record of type `State`, containing the value of the grouping attribute as well as the current state of the group. A typical application of this operator is to compute the current state of each node (e.g., ok, crashed, no route, ...). Here, records are grouped by node address.

predStateChange(attrGroup, attr1, attr2, ...) This predicate is used with the Filter operator to remove duplicate records. Records are first grouped by *attrGroup*. In each group, a record is dropped unless it differs from the previous record in that group in at least one of the attributes *attrX*. When applied to the output of StateDetector, (node) state changes can be computed.

4.7. Root Cause Analysis

The next step in the inspection of a sensor network is to derive the state of each node, which can be either "node ok" or "node has problem X". Note that the indicators mentioned above may concurrently report multiple problems for a single node. In many cases, one of the problems is a consequence of another problem. For example, a node that is dead also has a routing problem. In such cases, we want to report only the primary problem and not secondary problems. For this, we use a decision tree, where each internal node is a decision that refers to the output of an operator graph, and each leaf is a node state. In the example tree depicted in figure 4.1, we first check (using the output of an operator graph that counts packets received during a time window) if any messages have been received from a node. If not, then the state of this node is set to "node dead". Otherwise, if we received packets from this node, we next check if this node has any neighbors (using an operator graph that counts the number of neighbors contained in link advertisement packets received from this node). If there are no neighbors, then the node state is set to "node isolated". Here, the check for node death is above the check for

isolation in the decision tree, because a dead node (primary problem) is also isolated (secondary problem).

In general, the decision tree for the inspection of a WSN is constructed in a heuristic manner, based on a basic cause-effect analysis for the expected problems. In section 4.9.3, the decision tree used in our evaluation of a data gathering application is explained in detail.

4.8. Visualization

To display problems in the sensor network that have been detected by the data stream processor, SNIF provides a configurable user interface, which allows to display a real-time view of the network topology graph, where nodes and links can be annotated with application-specific information (e.g., state of a node, packet loss of a link) using a simple API. The core abstraction implemented by the user interface is a network graph, where nodes and links can be annotated with arbitrary information. Also, logging and later replay of execution traces is supported. Figure 4.19 shows an instance of this user interface for a typical data gathering application as discussed in the next section. Here, node color indicates state (green: ok, yellow: warning, red: severe problem, gray: not covered[1] by DSN -), detailed node state can be displayed by selecting nodes. Thin arcs indicate what a node believes are its neighbors, thick arcs indicate the paths of multi-hop data messages.

4.9. Evaluation: Data Gathering Applications

So far, most existing non-trivial deployments are data gathering applications (e.g., [47, 74, 80]), where nodes send raw sensor readings at regular intervals along a spanning tree across multiple hops to a sink. In this evaluation, we will therefore consider how SNIF can be applied to this application class. We first characterize the application in more detail and define the problems we want to detect. We then describe application-specific data stream operators to detect these problems and how they are used to form an operator graph. Finally, we evaluate the resulting inspection tool.

[1] We consider a node covered, if the DSN overhears a certain amount of beacon messages, e.g., 70%

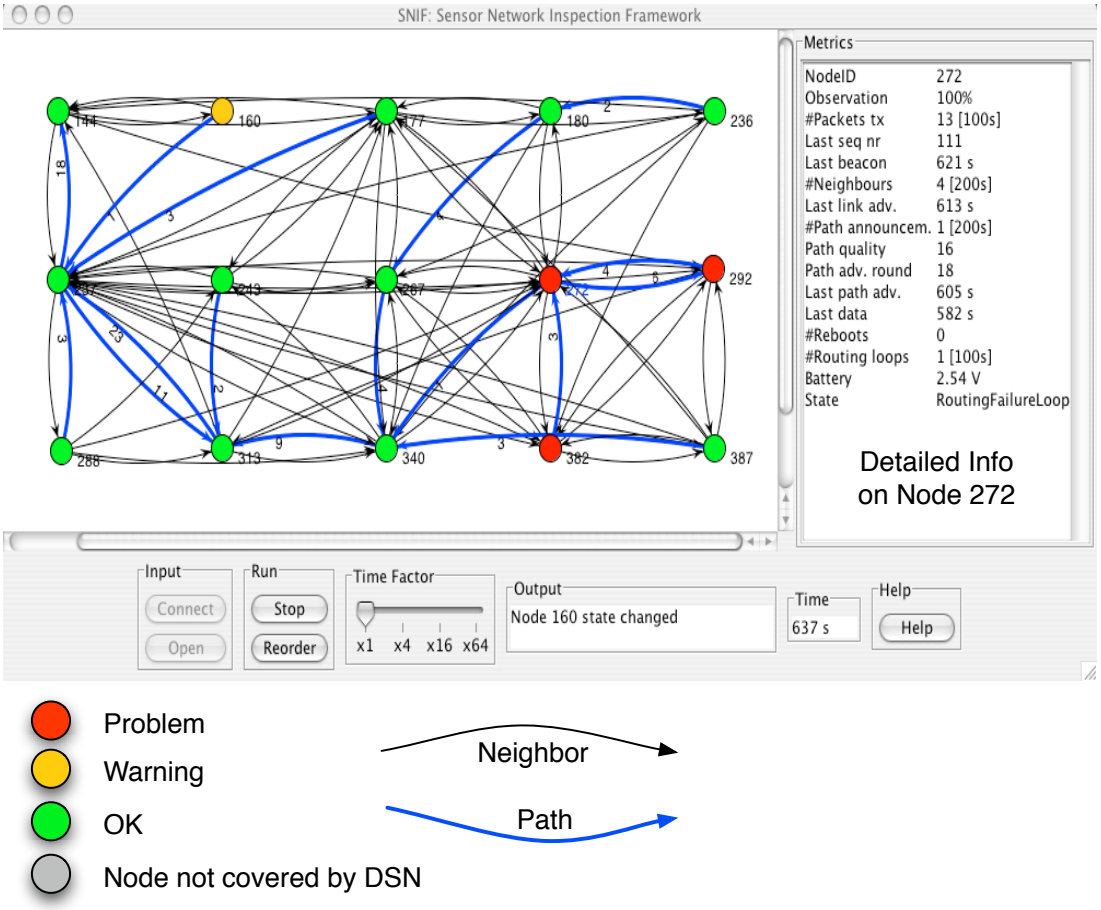


Figure 4.19.: An instance of SNIF's user interface.

4.9.1. Application Model

Two prominent implementations of data gathering applications are the Extensible Sensing System (ESS) [30] using beacon-based multi-hop routing for data collection, and Surge using MintRoute [89] for data collection. Both implement a similar multi-hop tree routing scheme as described below. We will use ESS as an example throughout the paper, but our approach can be readily applied to other, similar implementations.

In ESS, all nodes broadcast *beacon messages* at regular intervals. To discover neighbors, nodes overhear these messages and estimate the quality of incoming links from neighbors based on message loss. Nodes then broadcast *link advertisement messages* at regular intervals, containing a list of neighbors and link quality estimates. Overhearing these messages, nodes compute the bidirectional link quality to decide on a good set of neighbors. To construct a spanning tree of the network with the sink at the root, nodes broadcast *path advertisement messages*, containing the quality of their current path to the sink. Nodes overhearing these mes-

sages can then select the neighbor with the best path as their parent and broadcast an according path advertisement message. All this is executed continuously to adapt neighbors and paths to changing network conditions. Finally, *data messages* are sent from nodes to the sink along the edges of the spanning tree across multiple hops.

In ESS, beacons are sent every 10 seconds, path advertisements and link advertisements every 80 seconds, data message are generated every 30 seconds. All messages except data messages are broadcast messages and contain per-hop source address. Data messages contain the address of the originator of the sensor data and the per-hop destination address, but not the per-hop source address. In addition, beacon messages and data messages contain a sequence number.

4.9.2. Problems and Indicators

In section 2.2, we studied existing deployments to identify common problems and described passive indicators that allow to infer the existence of a problem from overheard network traffic in section 3.3.2. Below, we summarize the problems that are considered in the evaluation and give passive indicators for their detection.

Node death (fail stop) An affected node will not send any messages.

Node reboot After reboot the sequence number contained in beacon messages will be reset.

Node has no neighbor The node does not send link advertisement messages or they are empty.

Node has no parent The node fails to send path advertisement messages.

No path from node to sink Data messages sent by the node are not forwarded to the sink.

Node's path to sink loops A data message originating from the node is sent twice to the same destination by different senders. Note that this is a special case of "no path from node to sink".

Node partitioned from sink A node on the path from the node to the sink died and there is no alternate path available. Note that this is a special case of "no path from node to sink".

4.9.3. Decision Tree

In the following, we describe the decision tree used for the root cause analysis to detect the problems defined in the previous section. We first explain how the tree has been constructed and then describe each test in the decision tree and its corresponding part of the used operator graph.

The structure of the decision tree in figure 4.20 is motivated by the desire to find and report the *root cause* of a failure. For example, a dead node (root cause) also has a routing problem (consecutive fault). Here, we want node death to be reported, but not the routing problem. Hence, in the decision tree, the check to detect node death is located above the checks to detect a routing problem. Actually, the "Heard any packets?" test for a dead node is the very first one in the tree. For the remaining tests in the decision tree, we motivate why they are ordered in the presented way.

If a node reboots, the routing component will be reset and it will take awhile until the node can participate in the network again. Therefore, the "Sequence number reset?" test which detects reboots follows the dead node test. The most important requirement for a node to communicate is to have neighbors, hence, the "Has neighbors?" test is performed next. Having neighbors allows a node to learn of a routing path to the sink and select a routing parent to whom it sends data for the sink. If it does not have a parent, this can either be caused by a recent network partition or there might have not been a path to the sink before. If the node has a parent and it is working properly, it should periodically send data packets to the sink in our data gathering application. If no data from a node reaches the sink, this can again be caused by a recent network partition or by a routing loop. If neither a network partition nor a routing loop can be detected, we classify this as a general routing failure.

The tests in the decision tree make use the operator graph depicted in figure 4.21. We now explain these tests in detail:

Heard any packets? This test succeeds if any packet from a sensor node could be overheard. Since data messages do not contain the per-hop source address, DSNSource is filtered for data packets (multiHopFilter) and PacketTracer is applied to reconstruct

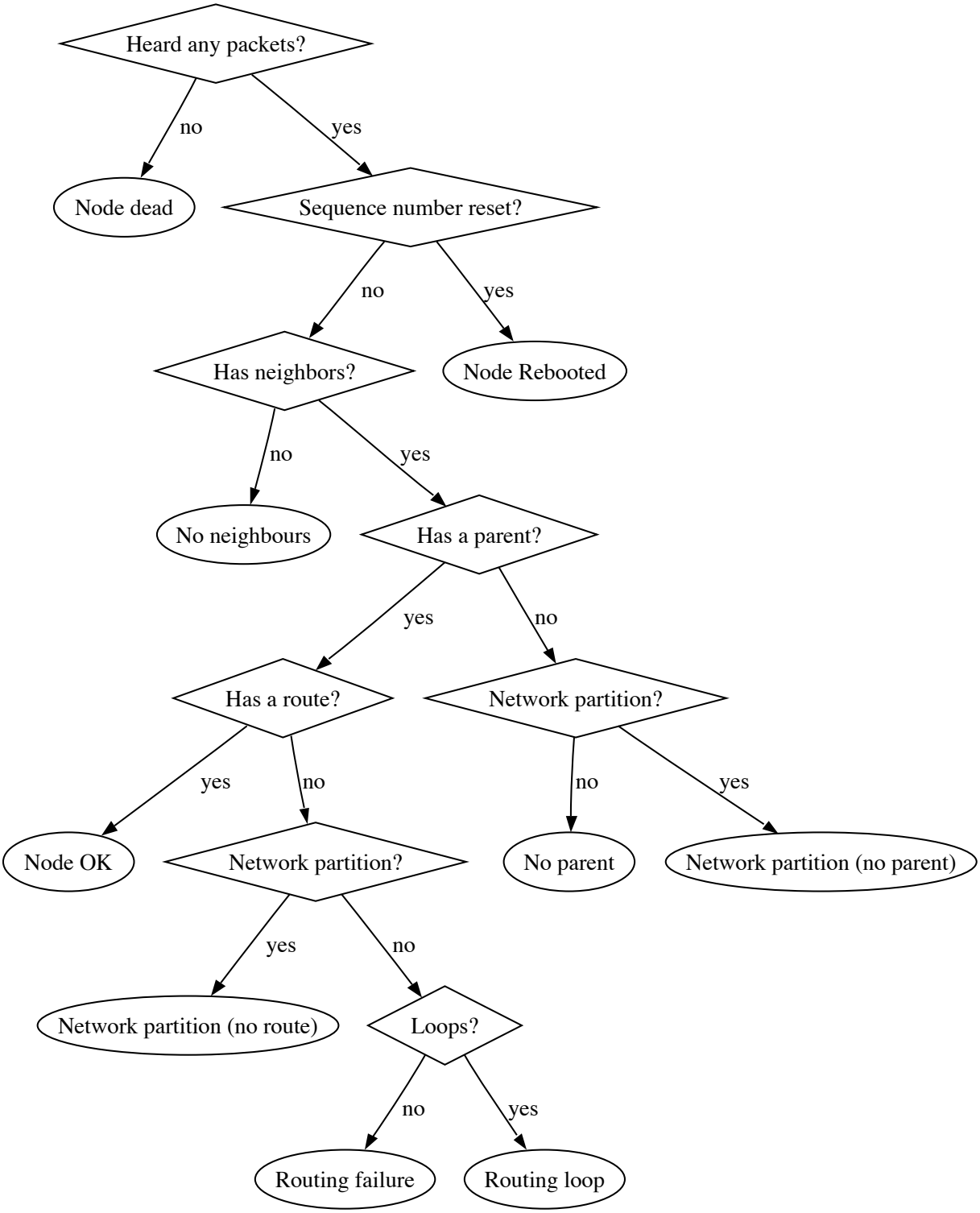


Figure 4.20.: Node state decision tree.

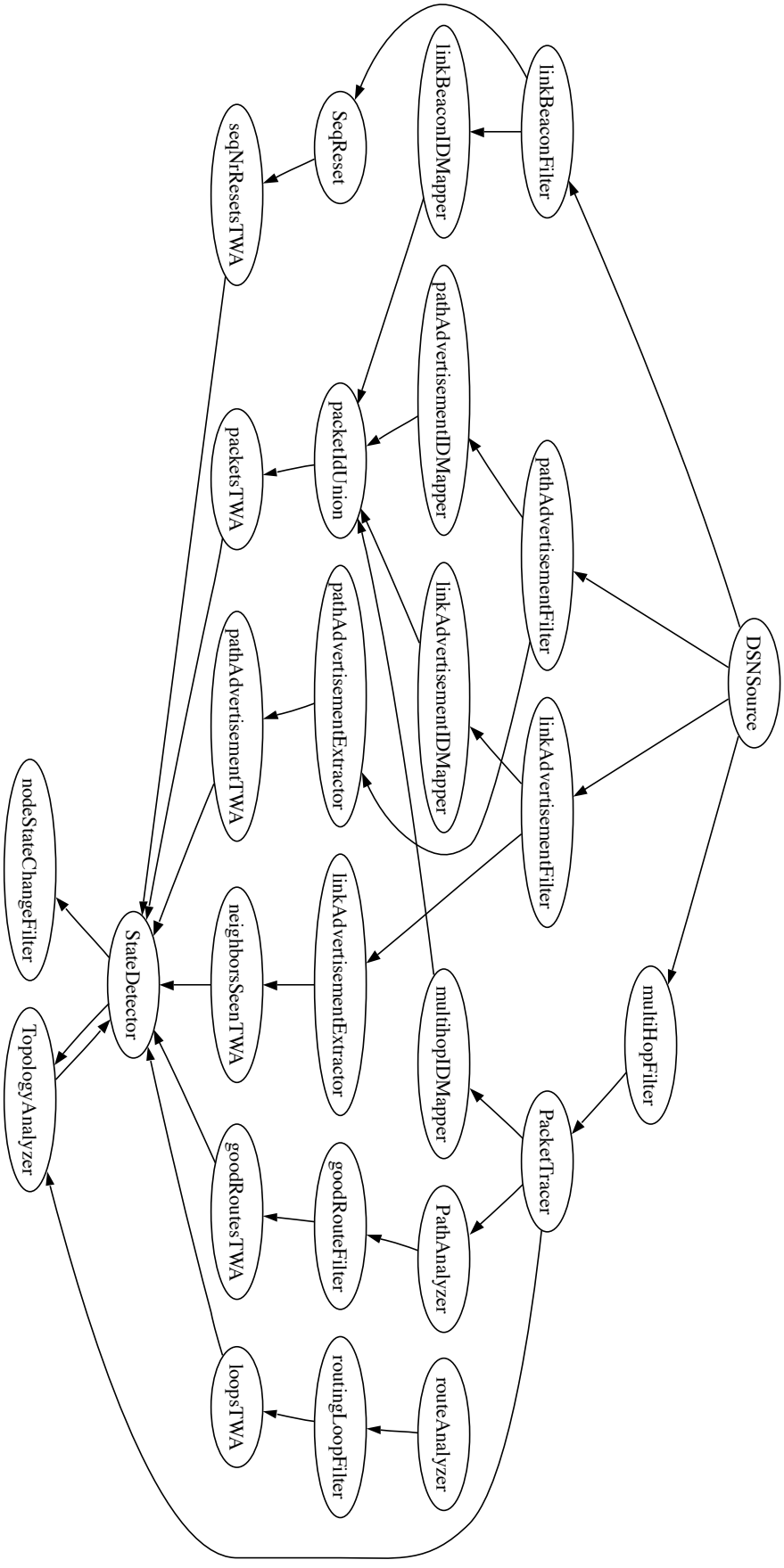


Figure 4.21.: Operator Graph for Indicators.

the source address. Also, DSNSource is filtered for the remaining packet types (beacon (linkBeaconFilter), link and path advertisements(linkAdvertisementFilter, pathAdvertismentFilter)) that do already contain the per-hop source address. A Mapper is used to normalize the name of a particular field to the generic srcAddr attribute (linkAdvertisementIDMapper, linkBeaconIDMapper, pathAdvertisementIDMapper, multihopIDMapper). The resulting data streams are then merged with the Union operator (packetIdUnion) to obtain a stream of all packets containing source addresses. This stream is then fed to a TimeWindowAggregator (packetsTWA) to count the number of packets per node using the count aggregation function.

Sequence number reset? This test succeeds if the node rebooted. To implement this test, DSNSource is filtered for beacon packets (linkBeaconFilter) and SeqReset is applied to the resulting data stream. To allow for a node reboot warning to be temporary, a TimeWindowAggregator (seqNrResetTWA) is used to count the number of resets per node.

Has any neighbors? This test examines whether a node has any neighbors. DSNSource is filtered for link advertisement packets (linkAdvertisementFilter). An ArrayExtractor (linkAdvertisementExtractor) is use to create one record for each neighbor. Using TimeWindowAggregator (neighborsSeenTWA) the number of such advertisements per node is computed. The test succeeds for a node if at least one non-empty link advertisement was heard from this node.

Has a parent? This test examines whether a node has a parent in the tree. DSNSource is filtered for path advertisement packets (pathAdvertisementFilter). Using an ArrayExtractor (pathAdvertisementExtractor) and TimeWindowAggregator (pathAdvertisementsTWA), the number of such advertisements per node is computed. The test succeeds for a node if at least one path advertisement was heard from this node.

Has a route? This test checks whether a node recently had a routing path to the sink. DSNSource is filtered for data messages (multihopFilter). PacketTracer is applied to reconstruct the source address. PathAnalyzer is applied and its output filtered for good route reports (goodRouteFilter). Using TimeWindowAggregator (goodRoutesTWA), the number of good

route reports per node is counted. The test succeeds for a node if at least one data packet from this node was sent to the sink.

Loops? This test checks whether the path from a node to the sink recently had any loops. DSNSource is filtered for data messages (multihopFilter). PacketTracer is applied to reconstruct the source address. PathAnalyzer is applied and its output filtered for routing loop reports (routingLoopFilter). Using TimeWindowAggregator (loopsTWA), the number of routing loop reports per node during a time window is counted. The test succeeds for a node if a routing loop was reported more than once for this node. This allows to mask temporary routing loops that may occur after a route change.

Network partition? This test checks if a bad path from a node to the sink was caused by a network partition. DSNSource is filtered for data messages (multihopFilter). PacketTracer is applied to reconstruct the source address. TopologyAnalyzer is applied to detect partitions. TopologyAnalyzer is also subscribed to the output of StateDetector in order to obtain *node death* events. The test succeeds for a node if the last record received from TopologyAnalyzer says that this node is partitioned.

Time window configuration In the above operator graphs, the time windows for TimeWindowAggregator are set to W times the interval of the packets they consider. For example, the time window in *Has a parent?* is set to $W \times$ 80 seconds, since path advertisement messages are considered which are sent every 80 seconds. That is, W is a global parameter and we will study its performance impact in the next section.

4.9.4. Results

To evaluate our data gathering application case study, we used the same experimental setup as described in [63], where the Extensible Sensing System (ESS) [30] is executed in the EmStar emulator [28]. The reason for choosing EmStar instead of the real DSN as a data source for evaluation is the ease of injecting failures in a reproducible way with EmStar.

As depicted in figure 4.22, we consider a network of 21 nodes forming a multi-hop topology with a diameter of 7 hops. Node 2 acts as the sink. We added three DSN nodes (nodes 31, 33, and 35 marked with squares in figure 4.22). The link dump files of the DSN nodes generated by EmStar

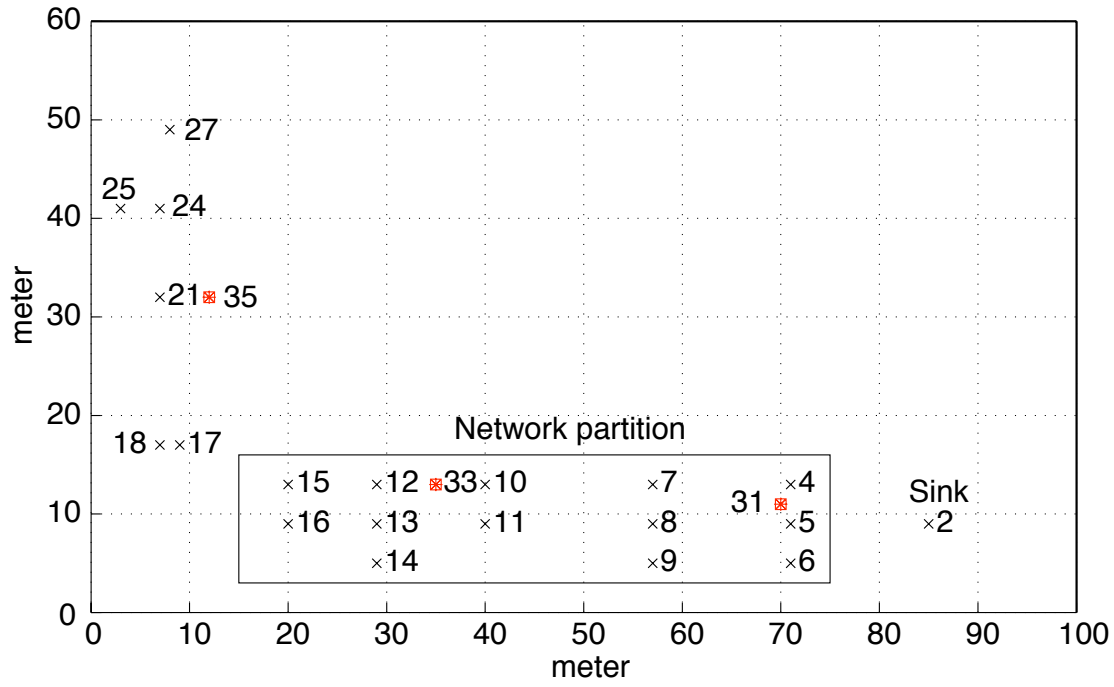


Figure 4.22.: Experiment setup: WSN (2-27) and DSN (31-35)

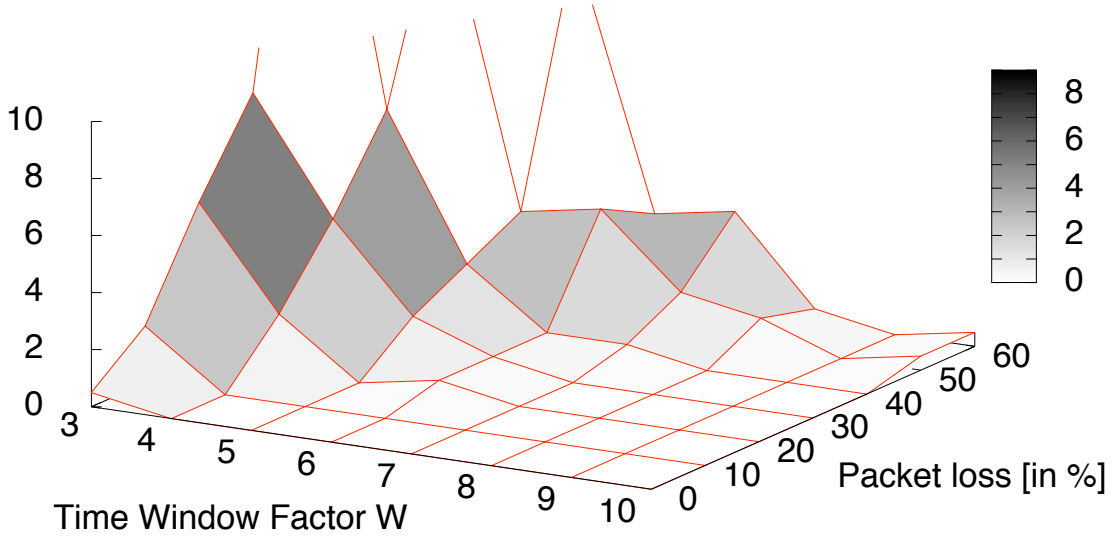


Figure 4.23.: Number of false reports as a function of packet loss and time window factor W.

were used as input to the inspection tool. Since some sensor nodes could be overheard by more than one DSN node, the DSN received 1.3 ± 0.5 copies of each sensor network message during the experiments, while 4% of the beacon messages were lost (i.e., not overheard by any DSN node).

Accuracy and Latency

We study the accuracy (number and type of false error reports) and latency (time between failure injection and report) of our inspection tool. These metrics mainly depend on two parameters: the size of time windows used in the operator graph (i.e., the value of the time window factor W) and the amount of packet loss (i.e., fraction of sensor network messages that were not overheard by DSN nodes).

As most decisions regarding node state are based on packets received during a fixed time window, increasing W should improve accuracy (as operators then have more packets to base their decision on) and increase latency linearly (as more packets need to be collected before a decision is made). Increasing packet loss should degrade accuracy (as operators with fixed time windows then have less packets to base their decision on) and decrease latency (e.g., since node death is reported when no packets are received from a node during a time window, loss of the last packets sent by a node before death will decrease latency).

In general, the latency to detect a problem is determined by the path of decisions leading to this problem in the binary decision tree depicted in figure 4.20. For example, the decision *Network partition?* leading to state *Network partition (no parent)* can only be made when the previous decision *Has a parent?* has been made with a result of *no*. That is, the latency for detecting a given problem is a function of the maximum latency of the decisions in the decision tree on the path from the root to the leaf denoting this problem. In turn, the latency of a decision is determined by the size of time window(s) in the associated operator graph.

In order to assess the impact of W and packet loss on accuracy and latency, we ran a set of experiments injecting three types of faults into the network: node failure, network partition, and no data. The duration of each experiment was 30 minutes with faults being injected randomly between 10 and 15 minutes after experiment begin. In addition to the (small) packet loss of the DSN, we introduced additional packet loss by uniformly dropping a given fraction of the overheard packets. We report averages and standard deviation over multiple runs.

To guide the selection of W for a given amount of packet loss, we ran a first experiment without injecting any faults, varying both W and packet loss, counting the number of (false) error reports for each parameter choice. The averaged results over 10 runs are depicted in figure 4.23. The flat area of the graph shows feasible values for W given a certain packet loss. For a packet loss of 30% (a common value in single-hop sensor networks [74]), no errors were reported for $W \geq 7$, motivating our choice of $W = 8$ to study the impact of message loss in more detail as depicted in figure 4.24. Similarly, we chose a packet loss of 30% for a more detailed study of the impact of W as depicted in figure 4.25.

In the first experiment, we performed 40 runs and injected a single node failure per run, such that all nodes but the sink failed twice. All node crashes were correctly detected and no false errors were reported. The latency of the reports is mainly determined by the size of the time window used to implement the *Heard any packets?* test which is $W\times$ 10s. For $W = 8$ and a beacon period of 10 seconds, we expect the latency to be between 70 and 80 seconds, which is confirmed by the experiments. Increasing packet loss does not have a significant impact on latency. The number of false positives is negligible until 30% of packet loss and raises significantly with more than 50% as depicted in figure 4.24. We analyzed the generated error reports and observed that for up to 70% of packet loss, we only observed *no neighbor* and *no parent* reports. These reports are caused by missing link and path advertisements, respectively, which are rarely sent (every 80s). For higher packet loss, we found *node dead* reports for working nodes. We never observed any false negatives. When varying W, we find (as expected) a linear increase of latency and an improvement of accuracy as depicted in figure 4.25.

In the second experiment we made nodes 4-16 fail at random times to partition nodes 17-27 from the remainder of the network. We would expect a *network partition* error for nodes 17-27. We report the latency until the first node was classified as partitioned. As explained above, the latency of partition detection is bounded by the latencies of preceding decisions in the decision tree, namely *Has a parent?* and *Has a route?*, which both use a time window of $W\times$ 80 seconds. As *Has a route?* basically tracks multi-hop data packets which are sent often (every 30s by all nodes), it reacts shortly before 640 seconds. The *Has a parent?* test fails, if no path announcements were observed during the time window. As explained above, increasing packet loss results in reduced detection latency.

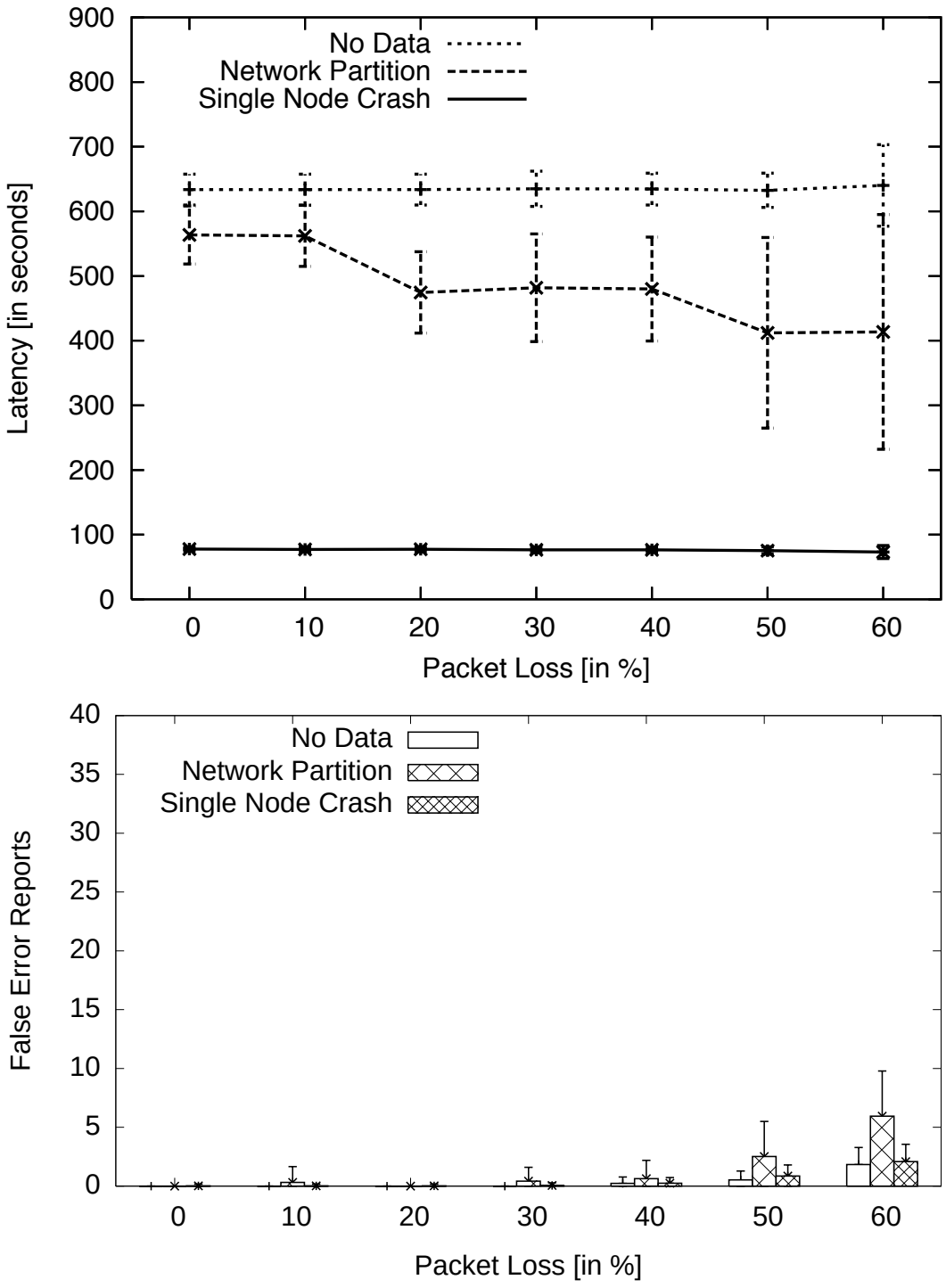


Figure 4.24.: Reporting latency (top) and number of false reports (bottom) as a function of packet loss for W=8.

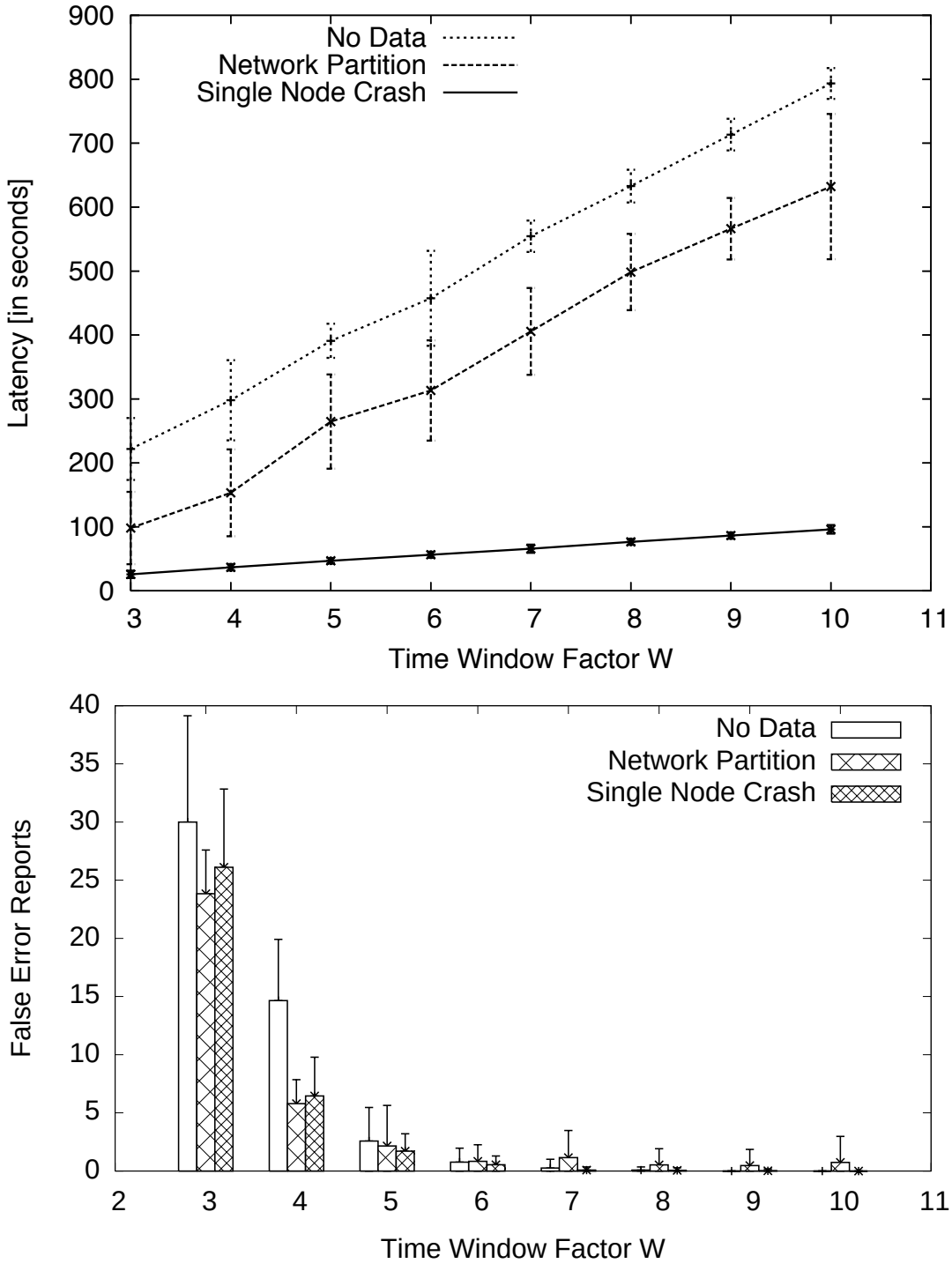


Figure 4.25.: Reporting latency (top) and number of false reports (bottom) as a function of time window factor W for 30% packet loss.

In the third experiment, we injected faults into the Multihop routing component of single nodes such that an affected node stops sending data messages, while still broadcasting beacons and advertisements. We would expect a *no route* error for the affected node and all other nodes whose paths contain the former. We report the time until the affected node is marked with *no route*. In this experiment, the latency is determined by the window size of the *Has a route?* test which is set to $W \times$ 80 seconds. As most nodes in the network forward packets for other nodes and data packets are sent every 30 seconds, the DSN should observe data packets until the fault is injected and the average latency should be close to the window size. The average of 633 ± 24 seconds for $W = 8$ and no packet loss confirms this. Again, in figure 4.25 bottom, the accuracy improves and latency increases linearly with W as expected.

SNIF Performance

We also studied the performance overhead of SNIF itself. During one 30 minute experiment run without any fault injections, the DSN collected 261 kB of data, resulting in an average data rate of 1.2 kbps including duplicate packets. Note that this equals about 0.3% of the effective Bluetooth 1.2 bandwidth of 400 kbps. SNIF was executing on a 2 GHz PC using Java 1.5. The total CPU time for processing the above amount of data was about 13 seconds, which equals about 0.7% of the experiment duration of 30 minutes.

A Bug in the ESS Multihop Component

In the course of our experiments, we encountered a bug in the ESS Multihop component. At one point we decided to upgrade to a new version of EmStar that fixed a bug with collision handling. After the upgrade, we suddenly observed a large number of *no parent* error reports without injecting any faults. As SNIF was still receiving close to 100% of all beacon packets and link advertisements, we concluded that this problems was caused solely by the path advertisement component. By examining the source code of Multihop, we learned that nodes react to receipt of a path advertisement message by updating their parent selection and broadcasting their updated path advertisement immediately without any additional delay. Hence, the original path advertisement broadcast results in an implicit synchronization of all receivers, such that the secondary path advertisements collide with high probability without being retransmitted.

By adding a random jitter, we were able to fix this problem.

Although SNIF was neither constructed nor configured for this particular problem, its ability to report both the close to 100% reception rate of beacon packets while receiving almost no path advertisement reduced the time to trouble-shoot this problem significantly.

4.10. Summary

In this chapter, we presented a framework for passive inspection of deployed sensor networks, consisting of a distributed network sniffer, data stream processor, and user interface. The key advantage of this framework is that sensor networks need not be instrumented for inspection. The framework has been specifically designed to support different protocol stacks and operating systems. We showed how this framework can be applied to data gathering applications, demonstrating that our approach can detect typical problems encountered during deployment timely and accurately even in case of incomplete information. Using this tool, we found a bug in the ESS application. SNIF has been fully implemented and demonstrated at the EWSN conference in 2007 [67].

Part II.

Fault Prevention

5. Medium Access Protocols for Wireless Sensor Networks

In the second main part of the thesis, we proactively address problems caused by the unsuitable design of current medium access control (MAC) protocols for *event-triggered* applications by providing a new MAC protocol specifically designed for this kind of application.

Many early applications of sensor networks were *time-triggered*, where sensor nodes sample their sensors at regular intervals and report these readings to a sink. In event-triggered applications, in contrast, sensor nodes do not transmit any data unless a relevant real-world event occurs. In the volcano monitoring application described in section 2.1.7, for example, sensor nodes detect volcanic eruptions by sampling their sensors. Only when a node detects an eruption, it sends a lengthy time series of sensor values to the sink, generating a *traffic burst* in the network. Because an eruption typically triggers many nodes simultaneously, the occurrence of traffic bursts produced by different nodes are highly *correlated* in time. Here, it is important that all data is collected in a reliable and timely manner with low energy overhead. However, it is equally important that the energy overhead during idle periods between eruptions is very low.

Existing medium access (MAC) protocols are not well-suited for such event-triggered applications with correlated traffic bursts as they are efficient either in idle mode or during correlated traffic bursts, but not both. Contention-based protocols such as SCP-MAC are very efficient in idle mode because they minimize control overhead required for node coordination. However, during correlated traffic bursts when many nodes compete for the radio channel, their efficiency is rather low due to control traffic required for contention. In contrast, scheduled protocols such as LMAC eliminate contention overhead during correlated traffic bursts, but exhibit a high overhead in idle mode due to the control traffic required for node coordination.

The goal of the second main part of this thesis is the design of a MAC protocol suitable for event-triggered applications with correlated traffic

bursts. It should have a low energy consumption for idle situations and be able to transfer the data during bursts efficiently. In this first chapter of the second part, we lay the foundation for the MAC protocols presented in the subsequent two chapters. We first give a general overview on the principles underlying MAC protocols for wireless sensor networks and then present related work with an emphasis on energy-efficiency and collision-free communication. Then, we specify the goal of the second part of this thesis in more detail. We present basic mechanism that will be used by the protocols introduced later.

5.1. Background

In contrast to existing wireless MAC protocols (e.g., for WLAN), sensor network MAC protocols stand out by their need for energy efficiency. Therefore, we first highlight the main sources of energy waste. Also, whereas WLAN communication generally is infrastructure-based, in which a number of entities communicate directly with a single access point, WSN increasingly make use of multi-hop communication, in which data is routed from sensor nodes to a central sink node over multiple intermediate nodes. Based on an understanding of the sources of energy waste and the need for multi-hop protocols, several design strategies for WSN MAC protocols have emerged, which we review next.

5.1.1. Sources of Energy Waste

Ye et. al. identified four main sources of energy waste in their seminal work on MAC protocols for WSN [90]:

- **Idle Listening:** If a node does not know when the next message for it arrives, it has to listen to the medium also when no messages are sent. This is called idle listening. Idle listening constitutes the major source for energy consumption in idle networks.
- **Overhearing:** Overhearing refers to the undesired reception of messages which are not addressed to oneself.
- **Collisions:** Collisions occur when a node transmits a packet in the vicinity of a node that is receiving another packet. Such a collision may result in a corrupted packet that has to be retransmitted. Retransmissions increase energy consumption for both the sender and the receiver.

- **Protocol Overhead:** Protocol overhead refers to the signaling protocol and the packet headers that are required to implement the medium access control protocol.

Over time, various MAC protocols have been proposed that address these sources of waste differently and are often named X-MAC, where X stands for a group of letters of the alphabet. [45] provides an overview of this "MAC Alphabet Soup".

5.1.2. Design Strategies

MAC protocols differ in the way how they address the sources of energy waste listed in the previous section. In general, all nodes in a WSN implement the same MAC protocol. On each node, the protocol has control over the operation mode of the radio transceiver and can use timers. Common radio transceivers for WSN work in half-duplex mode as they cannot receive and transmit at the same time and support three different modes of operation: sleep, receive and transmit. In addition, most radio transceivers allow for the measurement of the received radio signal strength (RSSI). This measurement can be used to perform a so-called *clear channel assessment* (CCA) [60], which tests if there is an ongoing transmission. Finally, some radio transceivers are able to set the frequency and are able to communicate on different channels, but only one at a time.

The main classification of MAC protocols is whether communication is scheduled in a *time-division multiple-access* (TDMA) fashion or not. Scheduling the communication helps to avoid idle listening, overhearing, and collisions, but incurs additional protocol overhead for the scheduling and time synchronization. If the communication is not scheduled, a *contention-based* approach has to be used, in which nodes compete for the medium. With such protocols, the probability for collisions depends on various factors: protocol design, traffic pattern, network load, and the network topology.

Independent of the scheduling of communication over time, communication can also be scheduled in the frequency domain, which is called *frequency-division multiple-access* (FDMA). The use of multiple, orthogonal frequency channels is possible in scheduled protocols as well as in contention-based protocols and can help to increase the throughput in a network and allows to communicate in overlapping clusters without interference. Although WSN are dominated by the strategies introduced here, others certainly exist for different kinds of wireless networks.

In the following, we further discuss the benefits and drawbacks of scheduled and contention-based protocols with an emphasis on two aspects: energy efficiency, especially the ability to reduce idle listening, and collision avoidance.

Contention-based Protocols

In this section, we discuss the main technique used in contention-based WSN protocols called *low-power listening* and the effectiveness of *carrier-sense multiple-access* (CSMA) for collision avoidance in sensor networks.

In the standard IEEE Wireless LAN 802.11 protocol [107], nodes listen continuously for incoming packet transmissions. For an otherwise idle network, this results in a dominant idle listening overhead. An efficient way to deal with this overhead is to let nodes only periodically poll the medium for on-going transmissions and keep the radio transceiver turned off most of the time. If a node detects a transmission, it keeps the radio turned on and tries to receive the packet. If there is no transmission, it turns the radios off until the next polling event. This scheme is called low-power listening.

However, as the polling is not scheduled, the sender does not know at which time the intended receiver will poll the medium the next time. A first practical solution to this problems was to extend the duration of the preamble preceding a packet to be long enough to assert that the receiver will poll the channel during the packet preamble and, hence, will be able to receive the packet. By this, the cost for idle listening is reduced, whereas the cost for actually transmitting a packet is increased. For applications with very low data rates, this already gets close to the optimum. For applications with low data rates, several improvements of this scheme further reduce the cost for a rendezvous between sender and receiver and allow to achieve idle radio duty cycles below 1%.

As contention-based protocols do not explicitly coordinate the send requests of multiple nodes, they attempt to avoid collisions by listening to the channel before sending, which is called carrier-sense multiple access. If the medium is found busy, a node stops its transmission attempt and tries later. This constitutes a *back-off mechanism*. When the node will try to send again is defined by a protocol's *back-off strategy*. In most WSN protocols, a node will try again after a fixed amount of time. A more advanced strategy is the *binary exponential back-off* algorithm specified by

the IEEE 802.11 standard, whereby a node will choose to wait for a random time between 0 and 2^i times a constant c before trying to send again. For this, i is initialized with 1 and is incremented with each failed try.

The problem with CSMA in wireless networks is that it is not possible to send and transmit at the same time and that switching between polling the channel and sending takes a non-neglibile amount of time. If two or more nodes poll the channel at nearly the same time, all of them will sense a free channel, switch to transmit mode, and instantly start to transmit, which leads to a collision.

Another issue not addressed by CSMA is the *hidden terminal problem* which occurs when two nodes A and B are in the communication range of node R, but nodes A and B cannot detect each other's transmission. Assume that node A is already transmitting a packet to node R. If node B senses the medium, it will not detect node A's transmission and will also start sending its packet. This results a in collision at node R. To reduce collisions caused by the hidden terminal problem, the 802.11 protocol provides the RTS/CTS (Request to Send / Clear to Send) mechanism. A node wishing to send data first transmits a request-to-send (RTS) packet to the destination node. The destination then replies with a clear-to-send (CTS) packet. Any other node that receives either the RTS or CTS packet should refrain from sending for the duration of the transmission. However, this mechanism is rarely used in wireless sensor networks for two reasons. Firstly, the CTS and RTS packets may also collide due to the hidden terminal problem similar to ordinary WSN messages, and secondly, the size of CTS and RTS packets including the long wake-up preamble is similar to ordinary WSN messages. Therefore, it is often cheaper to just retransmit the actual data packet than to make use of CTS/RTS signaling.

Because of the problem of accurately detecting a free medium and the hidden terminal problem, contention-based protocols are not suited for applications with concurrent transmissions where the probability for two or more nodes trying to send at the same time is especially high.

Schedule-based Protocols

Schedule-based MAC protocols coordinate the communication between neighboring nodes. This implicitly avoids collisions, and thus allows to handle traffic bursts. For this, time is divided into frames with a fixed number of slots and a schedule specifies for each node in which slots its has to listen and in which slots it is allowed to transmit. As nodes have to

listen only during certain slots, this leads to a reduced idle listening overhead. In the rest of this section, we discuss different ways how scheduling can be performed and their energy consumption.

If the traffic pattern is constant or predictable (e.g., if there are recurrent traffic flows), a global schedule can be constructed and disseminated, which results in very efficient communication with low idle listening, low overhearing, no collisions, and low protocol overhead.

However, if the traffic is not predictable, the schedule has to be constantly updated, which causes a high protocol overhead. For idle traffic situations, the scheduling even dominates the energy consumption. To reduce the scheduling overhead, MAC protocols can adapt two approaches: fixed slots or dynamic local scheduling.

In traditional TDMA protocols for single-hop networks, each node is assigned a fixed slot in which it can send. With such a fixed scheduling, all nodes have to listen in all slots, as they do not know, whether the owner of this slot will send a packet. To reduce the energy consumption for idle listening, the slots can be made rather long, as less slots per time unit have to be checked for transmissions. Although this saves on energy, it causes a high latency.

Instead of using a fixed schedule, the schedule can be dynamically calculated, but this introduces a chicken-and-egg problem. Let's consider the case of a TDMA protocol with a single receiver and multiple senders where the traffic is not predictable. If the receiver would know which of its neighbors want to send at the moment, it could construct a schedule and broadcast it. The senders could then send their message in turn without further coordination. The problem with this dynamic scheme is that in order to let the receiver know that a node wants to transmit something, it has to communicate with the receiver. So instead of the actual data transmission, the send requests have to be coordinated. For this, either a contention-based approach can be used or send request slots have to be assigned to the nodes. Similar to the non-scheduling MAC protocols, the contention-based approach may lead to collisions. On the other hand, reserving send request slots avoids collisions, but is inefficient, as the transmission of a complete packet is required to communicate the single bit send request over the radio. Even though the send request consists only of the sender ID, which can be encoded as a single bit of information if each slot is associated with a particular sender ID, it has to be framed into a complete packet, consisting of a preamble, the start-of-packet symbol and the actual payload data.

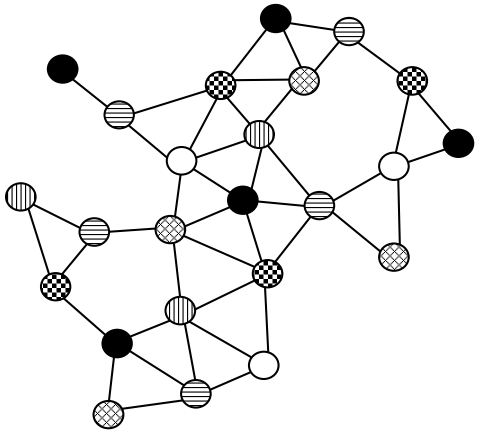


Figure 5.1.: 2-hop coloring of multi-hop network. Two nodes with a common neighbor must have different colors (represented by filling patterns here).

In addition to efficient scheduling, a scheduling MAC protocol also has to prevent collisions between nodes of separate scheduling domains and therefore has to construct a conflict-free schedule. A common approach is to pose the assignment of slot numbers to nodes as a *2-hop-coloring* problem, in which each node has to be assigned a color such that no two nodes with the same color also have a common neighbor as depicted in figure 5.1. Then, nodes with the same color cannot use the same slot for transmissions. This approach effectively prevents collisions for neighboring nodes and also avoids hidden-terminal situations.

5.2. Related Work

A comprehensive discussion and comparison of MAC protocols for sensor networks can be found in [46]. Here, we focus only on the latest and most relevant results and follow roughly the outline of the design strategies in section 5.1.2. After describing single-channel scheduled and contention-based MAC protocols, we look at first implementations of multi-channel protocols.

5.2.1. Contention-based Protocols

B-MAC

B-MAC [60] and "Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks" [22] have been the first two protocols to employ the low-power listening technique described in the previous section to reduce idle listening. In addition, B-MAC exposes certain MAC configuration parameters, e.g., the channel polling period and its back-off settings to higher layers respectively to the the application layer. B-MAC only implements a minimal back-off strategy. Per default, the protocol waits for a random time before sending a packet, and also uses a small random time in case the medium is busy. This allows for protocols on higher layers to define their own back-off strategy. Similarly, exposing the channel polling period, which determines the latency and the energy consumption of idle listening, allows to adapt the MAC protocol to application needs. However, for two nodes to communicate, the same polling period has to be used. Hence, the polling period should be the same in the whole network, and it is difficult to adapt the polling period at run-time in response to spontaneous changes.

X-MAC

X-MAC [11] improves upon the basic low-power listening concept by replacing the extended preamble by a sequence of short announcement packets that indicate the destination address of the actual packet. After each announcement packet, a short pause allows for the target node to respond with an *early acknowledgment* (ACK). Upon receipt of the early ACK, the sender can skip the rest of the preamble and start transmitting the actual data. As a node will wake up after half the preamble on average, this scheme saves half of the energy required for the long preamble. X-MAC specifies a custom back-off strategy based on the idea that a node that has been woken-up and has responded with an early ACK may receive further packets without long strobed preambles by staying awake after the first packet for a certain time. Hence, if node A attempts to send a packet to node B but instead of a free medium it detects a preamble, node A will wait for the ACK to check if it was sent by node B. In this case, node A will wait for the time it takes to transmit a packet with maximal payload plus a random back-off time before trying again to transmit its data. The random back-off time is intended to deal with multiple nodes that want to send to the same node. By this, the collision probability of random

access is increased as multiple senders that want to transmit to the same receiver concentrate their send attempts during the time the receiver waits for further packets. A further benefit of X-MAC compared to B-MAC is that it also accommodates radio transceivers that do not allow to send arbitrarily long preambles, such as the 802.15.4-based radio transceivers used on the Telos-B [97], MIZAz [97], and the TMote Sky [100] sensor nodes.

WiseMAC

WiseMAC [23] reduces the length of the preamble used for the low-power listening technique by keeping track of neighbors' sleeping schedules. The required schedule information is piggy-backed onto message acknowledgements and allows to predict the next time the neighbor is polling the channel, taking into account possible clock drift. With the estimated information about the next wake-up time, WiseMAC can reduce the length of the preamble to a minimum for unicast communications. All nodes that want to send to the same receiver are implicitly synchronized to the receiver's wake-up time. To avoid collisions in this case, an additional medium reservation window is used. A sender node picks a random time during this reservation window to sample the medium, and starts sending its preamble if the medium is free. If the medium is occupied, the binary exponential back-off strategy is applied to reduce the chance of collisions for consecutive transmission attempts.

SCP-MAC

SCP-MAC [91] is named after its "Scheduled Channel Polling" approach. Similar to the low-power listening technique, the nodes periodically poll the channel for ongoing transmissions. Here, the time interval between each channel polling is termed *polling interval*. In contrast to the previous three protocols, SCP-MAC synchronizes the sleep schedules of all nodes such that all nodes poll at the same time. By this, a sender can transmit a packet just at the moment when all neighbors wake-up, without an extended preamble and also allows for the efficient transmission of broadcast packets. Even the contention between multiple potential senders is performed before neighboring nodes wake up. With this, the idle listening energy is minimized to the energy required for time synchronization. However, the synchronized medium access leads to a higher chance of collisions that have to be resolved by back-off mechanisms. In addition,

only a single packet is transmitted per polling intervall. To increase the available bandwidth, SCP-MAC proposes an “adaptive channel polling” mode, which allows to send multiple packets during a polling interval. This is achieved by repeating the channel polling process multiple times during each polling interval and transmitting one packet in each so-called “high-frequeny polling slot”. This mode also allows to efficiently forward packets along a path through the network as a packet can be received and forwarded during the same polling interval. However, this streaming scheme only works for a single path as transmissions by multiple senders in the same area cannot be handled this way. The protocol does not specify a particular back-off strategy for the case that a node looses the competition to sent against another node. Therefore, it will try again at the next possible time.

5.2.2. Schedule-based Protocols

LEACH

LEACH [32] partitions the network into clusters, such that all nodes in a cluster can communicate with each other. In each cluster, the cluster head assigns a fixed TDMA schedule to the nodes. Non-cluster-head nodes send their data to the cluster head, which forwards the data to the sink. The role of the cluster-head is rotated to distribute energy consumption equally among all nodes. However, since cluster heads communicate via long-range radio with the sink, all nodes must be capable of long-range communication, which is a strong assumption.

BMA

The “bit-map-assisted” (BMA) MAC protocol [50] is similar to LEACH, but provides dynamic slot allocations within each cluster. For the slot allocation, a "contention period" is used, where each cluster member is assigned a short send request slot. After the send requests are collected, the cluster head broadcasts a transmission schedule. Although the collection of send requests requires only a single bit of information per node, it is not discussed whether this can be implemented efficiently. Instead, a control packet with a total length of 18 bytes is used to communicate a single-bit send request in the analytical part of the paper.

TRAMA

With TRAMA [62], the TDMA scheduling is replicated over the nodes of the network. However, the schedule can only be adapted to long-term traffic flows through the network. The protocol additionally assumes an out-of-band mechanism for time synchronization and exhibits a high latency.

LMAC

In contrast to the previous protocols, LMAC [81] has been implemented and used in sensor network deployments. LMAC establishes a slot structure for the whole network and assigns each node to a slot in a TMDA schedule using 2-hop coloring. For this, each node transmits in each round which of the slots are used by itself and its neighboring nodes. By computing the union of the allocated slots of its neighbors, a new node can learn about the occupied slots in its two-hop neighborhood. From the list of free slots, the new node then picks a random slot that it uses for its transmission in the future. With this scheme, LMAC effectively avoids hidden-terminal problems at the expense of using more slots than the maximal degree of nodes in the network. The original paper suggests 32 slots. However, all nodes must listen during all slots to receive data. This implies a significant energy overhead, resulting in a radio duty cycle $\gg 1\%$ even in idle mode.

AI-LMAC

The "Adaptive, Information-centric and Lightweight MAC" [17] is based on the LMAC protocol and allows nodes to own more than a single slot for sending. With AI-MAC, in a data-gathering application, nodes closer to the sink can uses more slots, which improves throughput and latency. However, the decision on how many slots a node should use is left to the application. To this end, the paper proposes a data management framework that deals with predictable traffic, e.g., if the network has to respond to dynamic queries and the distribution of the queried sensors is known.

Dozer

While the previous protocols have been general purpose MAC protocols, Dozer [14] is an integrated data collection stack for ultra-low data rates, consisting of MAC layer, topology control, and routing. After an initial

spanning-tree construction, each inner node coordinates the data collection from its child nodes. The child nodes are enumerated and a basic fixed schedule allows each child node to forward its data to the parent node. In each slot, a child node can send multiple packets until the end of its slots. Each packet is then directly acknowledged by the parent node. Although the communication between a parent and its children nodes are scheduled, transmissions of different parent-child star topologies may interfere. For the low data rates, for which Dozer was designed, such collisions are infrequent. To avoid the situation that schedules of neighboring star topologies are synchronized, which would result in systematic collisions, the length of the TDMA schedule is extended each round by a small time interval by the parent of each star. However, in the case of correlated traffic bursts, most slots in each star schedule are used, which results in significant collisions between the transmissions of neighboring stars.

5.2.3. Multi-Frequency Protocols

"A Practical Multi-channel Media Access Control Protocol for WSN"

The protocol by H. Le et. al. [48] distributes nodes over the available frequency channels in a way to maximize the total network bandwidth. Each node listens on a single channel and this channel is called its "home channel". Initially, all nodes start with the same home channel. When this channel becomes overloaded, some nodes migrate to another non-interfering channel. As nodes keep track of other nodes' home channels, a node A with home channel X can still send a message to node B with home channel Y by temporarily switching to channel Y. The decisions when to migrate to a different channel and to which channel are performed by a local heuristic algorithm. The protocol is proposed as an additional component on top of an existing single-channel MAC protocol. For contention-based protocols, this poses no new problems as a message can be sent at any time. However, it is not clear how the channel switch required to transmit a message to a node on a different home channel can be integrated into the scheduling of TDMA protocols. Finally, the focus of the protocol is on network throughput. Energy efficiency has not been addressed and the additional periodic channel update packets that are used to maintain connectivity increase energy consumption significantly for the idle case.

Y-MAC

The Y-MAC [42] protocol also increases the network bandwidth. In contrast to the previous protocol, which builds on top of an existing single-channel MAC protocol, Y-MAC is based on a TDMA scheme where each node is assigned a time slot similar to LMAC. However, whereas in LMAC a node sends during its slot, in Y-MAC it has to listen. All nodes that want to send to a particular receiver compete for the medium during a contention window in the beginning of the receiver's time slot. In addition to these unicast slots, the protocol provides a dedicated broadcast period in which a node can reach all neighbors. By this, a node only has to listen during the broadcast period and during its own unicast slot which reduces the cost for idle listening. However, the TDMA approach increases the latency and reduces the throughput compared to, e.g., SCP-MAC, where a packet can be sent in each slot. To allow for traffic bursts, a node that received a packet during its assigned slot x will change to a different channel and be ready to receive another packet on the new channel in slot $x + 1$. The sender of the first packet as well as other contenders that lost during slot x will follow the receiver to the new channel. This process continues as long as there are transmissions for the receiver. Although orthogonal channels and a TDMA-approach are used, contention is still used for each transmitted packet. In particular, all nodes that try to send to the same receiver (which is the common case in tree-based collection protocols) will send at the same time on the same channel; hence, they suffer from the same contention overhead as in the SCP-MAC protocol.

5.2.4. Discussion

Out of the surveyed protocols, only SCP-MAC, WiseMAC, and Dozer achieve a low idle-duty cycle below 1%. However, correlated traffic bursts lead to collisions or cause a large number of nodes to repeatedly compete for the medium and have them back-off while the medium is constantly busy. Collisions and the high number of times a node has to compete for the medium increase the energy consumption per packet significantly. In contrast, most scheduled protocols, and especially LMAC, can handle traffic bursts without collisions, but their idle listening overhead is higher and the maximal throughput is severely limited by the fixed slot assignment.

5.3. Problem Statement

In the previous section, we have surveyed existing MAC protocols with an emphasis on energy efficiency and collision avoidance. None of them both achieves a low idle duty-cycle and can handle correlated traffic bursts efficiently at the same time.

Based our study of existing MAC protocols and our work on the inspection of sensor networks, we now outline the requirements for a MAC protocol specifically designed for event-triggered applications with correlated traffic bursts.

5.3.1. Determinism

In order to reduce uncertainty in WSN, the new MAC protocol should be able to handle correlated traffic bursts in a deterministic way. Since contention-based protocols are not deterministic, their behavior is not predictable. Therefore, the analysis of problems is aggravated. Furthermore, as the probability for collisions increases with the number of nodes trying to send simultaneously, traffic bursts might even incur chaotic behavior. Therefore, the new MAC protocol should be scheduled, and the behavior of all relevant components should be deterministic also during traffic bursts.

5.3.2. Energy Efficiency

The traffic of event-triggered applications is characterized by long periods where no data is generated. Therefore, the energy consumption during these periods has to be on the same level as current state-of-the-art WSN MAC protocols. This means, that the radio duty cycle for idle networks should be below 1%. Then, in the case of an event, many nodes will try to report their measurements simultaneously. For this, the scheduling overhead should be low to keep the energy for the actual data collection at reasonable levels. Finally, if large amounts of data are generated, this data has to be transferred efficiently by the network.

5.3.3. Latency

In general, there is a trade-off between energy consumption and the reactivity of a network. A shorter latency usually requires more energy. How-

ever, we still expect that the latency between event detection and event reporting to be in the order of seconds and not minutes.

5.4. Approach

In this section, we first explain our strategy to fullfill the stated requirements of event-triggered applications with correlated traffic bursts, then we outline the basic structure of our protocols, and give an overview over the remaining sections and the remainder of the second part of the thesis.

As pointed out in section 5.3.1, we focus on schedule-based MAC protocols as these achieve deterministic behavior in the presence of correlated traffic bursts. However, at first glance, this choice incurs higher energy consumption and latency. We address these issues by combining several existing and new techniques to balance the cost for the deterministic behavior.

A major improvement in terms of energy efficiency, especially in idle situations, is achieved by synchronizing the time when nodes exchange messages, similar to the SCP-MAC protocol. In addition, we developed a new single-bit transmission technique that allows to collect small amounts of the information without the associated high overhead of sending complete packets and thus allows for efficient on-demand scheduling. Finally, the coordination of neighboring scheduling domains is achieved, implicitly based on the global time and the node's ID, without explicit communication.

So far, schedule-based MAC protocols have dealt with the hidden terminal problem by scheduling slots to nodes in a way that would prevent nodes that share a common neighbor to use the same slot. However, this incurs a high latency compared to contention-based MAC protocols because a node can only send during the slot that it has been assigned. In addition to scheduling over time, we make use of the ability of current radio transceivers of sensor nodes to communicate on a range of orthogonal channels by distributing the nodes over the available channels. All nodes that want to communicate with a particular node switch to that node's channel. This effectively avoids the hidden terminal problem while increasing the network capacity at the same time.

We designed two protocols: BitMAC and BurstMAC. In both protocols, nodes are organized into 1-hop star networks, where a coordinator node communicates with its direct neighbors. The whole network then consists of all interconnected star networks. The nodes of each star net-

work communicate on a radio channel orthogonal to the one of other co-located stars. The radio channels are determined by a 2-hop-coloring of the network as depicted in figure 5.1. We assume that there are enough channels available for this and will evaluate this assumption in chapter 6. We further assume in both protocols that a node receives the "or" of the transmissions of all senders within its communication range. We will validate this assumption in the next section and demonstrate as a first example for such a *cooperative transmission scheme* how send requests of multiple nodes can be collected with minimal overhead.

In BitMAC, we focused on data-gathering applications with a static network and a central sink. We made use of new cooperative transmission schemes to achieve a parallel and time-bounded coloring of unstructured networks. However, some parts of the protocol do not cope well with changes in the network topology. In our second protocol, BurstMAC, we used a less efficient approach for the coloring of the network, but this led to a protocol with less stringent timing requirements and which can deal with topology changes.

After this overview of our approach, we will introduce in the next section our work on cooperative transmission schemes which allow a set of nodes to communicate at the same time in a useful way. Then, we present BitMAC and BurstMAC in chapter 6 and chapter 7 respectively.

5.5. Cooperative Transmission Schemes

In this section, we introduce the concept of cooperative transmission. We first state and valildate our assumption on the used radio technology for a concrete sensor node radio transceiver, the ChipCon CC1000, before we present related work on this topic in a broader context. Finally, we demonstrate how a single bit of information can be collected from all neighbors without the need for preambles. Further, more advanced techniques will be presented as part of the BitMAC protocol in section 6.3.

5.5.1. Assumptions

Our work is based on the assumption, that a node receives the "or" of the transmissions of all senders within communication range. In particular, if bit transmissions are synchronized (e.g., slotted access to the medium) among a set of senders, a receiver will see the bitwise "or" of these transmissions.

This behavior can actually be found in practice for radios that use On-Off-Keying (OOK), where ”1”/“0” bits are transmitted by turning the radio transmitter on/off. Note that transmitting a zero is then equal to sending nothing.

Although the use of OOK modulation has been superseded by more efficient modulation schemes over time, it is still possible to receive the “or” of the transmissions of all senders within communication range. The first generation of sensor nodes that achieved widespread use (e.g., MICA and RENE motes [97], Scatterweb ESB [102], TecO Particles [103], and the EYES project prototype [99]) employed the RTF TR1000 radio transceiver that directly provided OOK modulation. The next generation of sensor nodes (e.g., MICA2 motes, MANTIS Nymph [101], BTnode Rev. 3) was based on the Chipcon CC1000 series, which only supported Frequency-Shift-Keying (FSK) modulation. However, an application note [96] of the manufacturer describes how OOK modulation can be emulated for the CC1000. The newer Chipcon CC1020 and CC1021 provide support for OOK modulation in addition to FSK and GFSK (Gaussian Frequency Shift Keying).

Our MAC protocols will use this communication model (and hence OOK) only for a limited number of control operations. For the remainder (e.g., payload data transmission), other, perhaps more efficient modulation schemes can be used if supported by the radio. In this case it might be necessary to adjust the transmit power such that the communication ranges are similar for both modulation schemes.

5.5.2. Validation

In a first experiment, we validate our communication model by showing that a node can indeed receive the bitwise “or” of the transmissions of two other nodes.

For the hardware experiments, we used the BTnode Rev. 3, which provides a Chipcon CC1000 low-power radio. As this radio does not directly support OOK transmission it has to be emulated. For this, the transmitter sets the frequency separation to 0 Hz and switches the power amplifier in the transmitter section on and off to transmit single bits. On the receiver side, the received signal strength indicator (RSSI) is used to decide whether a “1” is transmitted or not. A “1” is assumed if the RSSI value is above a fixed threshold. The built-in analog-to-digital converter (ADC) of the ATmega128 was used to measure the analog RSSI output. The

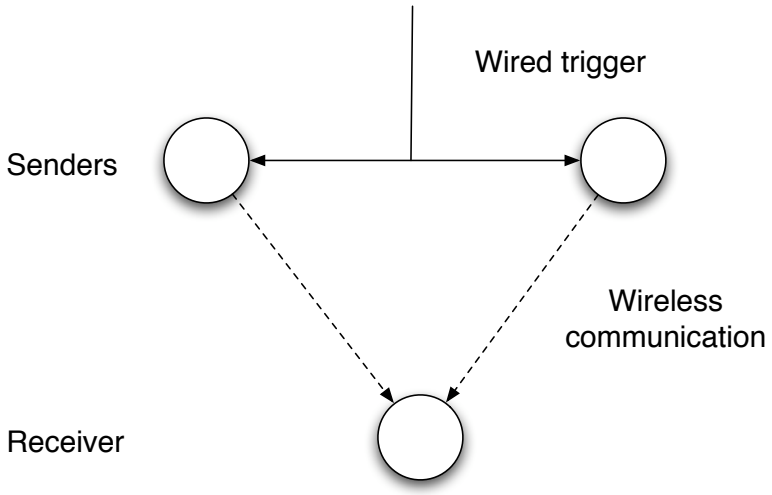


Figure 5.2.: Experiment setup for the OOK emulation on the CC1000

BTnode provides a 7.328 MHz system clock, which allows a sampling rate of 250 kilo samples per second (i.e., one sample every 4μs).

To verify OOK transmissions, we used three nodes in the setup shown in figure 5.2. Two sender nodes are triggered concurrently via a wire to start sending their bits via radio. A receiver hears both transmissions. As illustrated by the oscilloscope images in figure 5.3, the senders transmit the bit sequences (a) “101000” and (b) “100010”. The third node receives (c) “101010” as expected. The images were obtained by connecting the analog RSSI output of the receiver to the oscilloscope. A low voltage represents a “0” bit.

Although figure 5.3 clearly shows the constructive superposition of the sent signals, superposition can also result in destructive interference, which would render our “or” assumption useless. However, Krohn et. al. could show in [43] that the modulation of the transmitter carrier frequency by band-limited noise reduces the effect of destructive interference to a minimum.

Note that in this experiment, we used a bit time of 300 us, which corresponds to a bit rate of 3.3 kbps, to assert that the transmitted OOK bits can be decoded correctly. Radios with direct support for OOK and with digital signal output (such as the RFM TR 1000 or newer Chipcon radios) allow to use shorter bits. Hence, our experiments with the Chipcon CC1000 can be considered as worst-case results. In the implementation of BurstMAC in chapter 7, we were able to reduce the bit duration to 150 us.

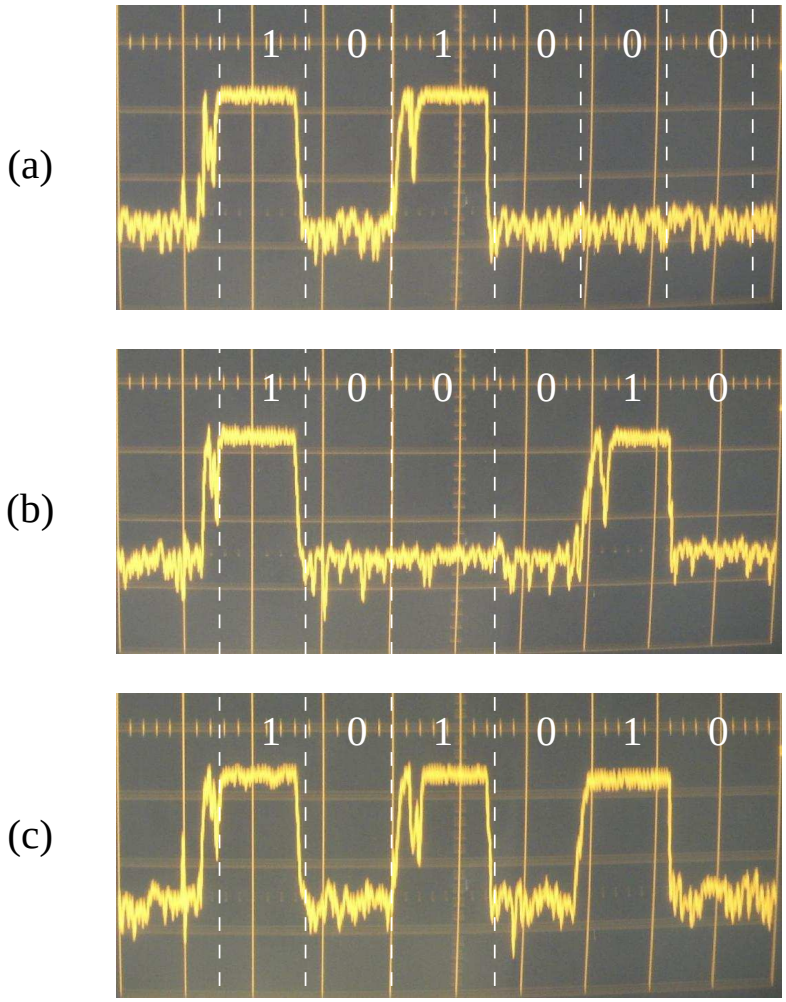


Figure 5.3.: RSSI measurements for reception of OOK data (time 0.2 ms/div, voltage 0.5/div), low voltage represents a “0” bit. (a) First node sending “101000” (b) Second node sending “100010” (c) Both nodes sending.

5.5.3. Related Work

The concept of cooperative transmission has been proposed and explored before in various settings. Here, we present its use in three different areas: time synchronization, data aggregation, and network density estimation.

“Asymptotical Optimal Time Synchronization in Dense Sensor Networks”

In [34], a time synchronization protocol based on cooperative transmission is described. In this approach, a designated node emits a sequence of pulses. Nodes that hear this pulse sequence predict when the next pulse

will be sent and transmit a synchronous pulse themselves at the predicted point in time. Eventually, all nodes in the network will hear and transmit synchronous pulse sequences.

"A Lightweight MAC and Data Aggregation Protocol'

In [56], the concept of multiple nodes sending a bit vector of length k in parallel is proposed. As a consequence, the logical OR of the sent bit vectors is received. In addition to the OR operation, it is suggested that the MAX operation can be implemented similarly. However, this suggestion is not explored any further. Also, no details on a possible implementation on real hardware are given in the article.

Synchronous Distributed Jam Signaling

An interesting approach to estimate the number of neighboring nodes is presented in [44]. The common approach would be to broadcast a "hello" message and have all neighboring nodes answer with an "hello-ack" message. However, this is inefficient, as all nodes would have to send a complete packet just to transmit their ID, which would result in a correlated traffic burst that would stress the MAC protocol and could cause collisions. To overcome this problems, cooperative transmission is used by the "Synchronous Distributed Jam Signaling" (SDJS) scheme. After an initial synchronization message, each node randomly chooses a slot out of the available N slots to send a jamming signal. The number of slots in which at least one or more nodes sent a jamming signal is an indirect measure for the number of nodes. Although two or more nodes might jam the same slot, statistical methods allow to calculate the most likely number of sending nodes given the fixed number of jamming slots and the observed number.

5.5.4. Single-Bit Transmissions and Preamble Elimination

After validating our assumption on concurrent transmissions and proving example use cases in other areas, in this section we show, how single-bit transmissions can be implement to eliminate the need for preambles of send requests in TDMA-based protocols.

Every transmission of payload data typically has to be preceded by a preamble (often a "101010..." bit sequence) and a start-of-packet (SOP) delimiter with a total size of about 100 bit. Preamble and SOP are needed

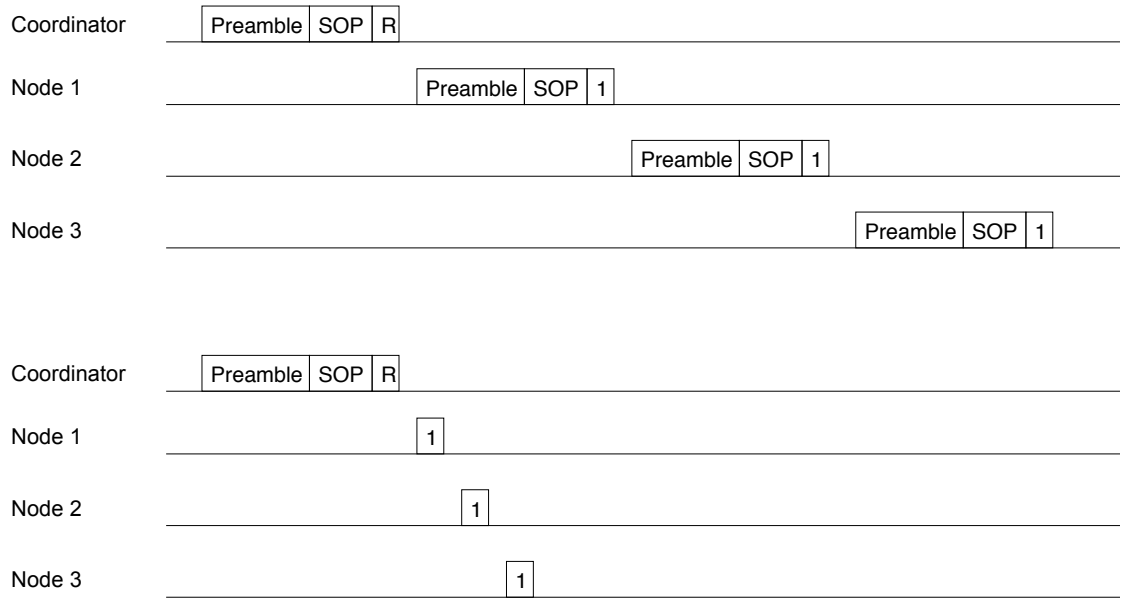


Figure 5.4.: Single-bit transmissions to a coordinator node: with (top) and without (bottom) preambles. Time increases to the right.

to synchronize the receiver to the sender and to adjust the bit-decision threshold.

Control traffic of a MAC protocol often consists of data packets with few or even single bits. For example, in a TDMA protocol without a fixed slot assignment, each node that wants to transmit has to signal a send request to a coordinator node such that a schedule can be constructed. Preceding every send request with a preamble represents significant overhead of MAC protocols as depicted in figure 5.4(top).

Fortunately, many preamble transmissions can be eliminated. To collect send requests, the coordinator may trigger (by means of a request message "R" in figure 5.4) the other nodes to send their requests within a short time. Typically, these single bit transmissions are preceded by preambles as in figure 5.4(top).

However, since the coordinator's preamble does already synchronize all other nodes, the latter can maintain synchronization for the duration of several hundred bits using their hardware clocks. Hence, single bits can be transmitted without preambles as depicted in figure 5.4(bottom).

5.6. Summary

Existing MAC protocols are not able to handle the concurrent traffic bursts that occur in event-triggered WSN applications without collisions and frequent retransmissions. In order to reduce the uncertainty in the behavior of these protocols, schedule-based protocols are needed. But while schedule-based protocols can solve the problem with collisions, they incur a high overhead for idle situations. As a first step towards efficient schedule-base protocols, we showed in this section how the collection of send requests which has been proposed before in the BMA protocol, can actually be implemented on common sensor node platforms. In the following two chapters, we will show how this single-bit transmission technique is used together with other novel techniques to create an efficient MAC protocol for event-triggered applications.

6. BitMAC

The previous chapter motivated the need for schedule-based MAC protocols with low idle duty cycle for event-triggered applications. This chapter will focus on the BitMAC protocol that achieves this in a fully deterministic manner without collisions. Even more, we show that cooperative transmissions can be embraced as building blocks for the design of a collision-free protocol.

After stating the assumptions for BitMAC, we present an overview of the protocol. Then, we describe the protocol in detail and evaluate its major components, before concluding the chapter with a discussion on remaining issues and a summary.

6.1. Assumptions

BitMAC was designed for a common use case of WSN with respect to the application characteristics and the network topology, which we state now.

6.1.1. Application Characteristics

BitMAC is designed for data-collection sensor networks, where many densely deployed sensor nodes report sensory data to a sink across multiple hops. In order to avoid the bottleneck of the sink, data from multiple sensor nodes may be aggregated by nodes in the network. Data communication is mostly uplink from the sensor nodes to the sink, although the sink may issue control messages to the sensor nodes. One prominent example of this application class are directed diffusion [36] and TinyDB [51]. Many concrete applications (e.g., [6, 40, 66, 75]) show this behavior as well.

6.1.2. Network Topology

Furthermore, it is assumed that the network topology is mostly static. That is, after initial deployment, node mobility and addition are rare

events. However, BitMAC is designed to support a large class of permanent or temporary node failures efficiently and without introducing contention or indeterminism. Hence, BitMAC can support applications with real-time and robustness requirements. Applications such as [6, 40, 75] fall into this class.

6.2. Protocol Overview

BitMAC is based on a spanning tree of the sensor network with the sink at the root. In this tree, every internal node and its direct children form a star network. That is, the tree consists of a number of interconnected stars. Within each star, time-division multiplexing is used to avoid interference between the children sending to the parent. Time slots are allocated on demand to nodes that actually need to send. Using a distributed graph-coloring algorithm, neighboring stars are assigned different channels as to avoid interference between them. Both the setup phase and actual data transmission are deterministic and free of collisions.

In the following sections we will describe the complete protocol with increasing level of complexity. We will begin with new cooperative transmission schemes that build upon those in section 5.5 and are used by the graph-coloring algorithm. We will then discuss the part of the MAC protocol used to control a single star. Finally, we will describe how these stars can be assembled to yield the complete multi-hop MAC protocol.

6.3. Advanced Cooperative Transmission Schemes

In this section, we discuss the potential of cooperative transmission schemes to implement several logical and arithmetic operations for single and multiple overlapping star networks as depicted in figure 6.1.

6.3.1. Integer Operations

Let us assume that all or a subset of children need to transmit k-bit unsigned integer values to the parent, where the latter is interested in various aggregation operations (OR, AND, MIN, MAX) on the set of values of the children. Below we discuss efficient ways to implement these operations, assuming synchronized nodes and the communication model described in section 5.5.1.

Obviously, a bitwise "or" can be implemented by having the children synchronously transmit their values bit by bit. Our communication model ensures that the parent will receive the bitwise "or" in time $O(k)$. Since $x \text{ AND } y = \overline{\bar{x} \text{ OR } \bar{y}}$ (where $\bar{x}$ is the bitwise inversion of x), the bitwise "and" can be obtained if the children invert their values before transmission and if the parent inverts the received value.

By interpreting an integer value as a set (where i is contained in the set if and only if the i-th bit of the value is 1), the operations OR and AND implement the UNION and INTERSECTION of integer sets, respectively. Integer values with k bits can then support sets of up to k elements. Often, long sequences of zero bits have to be transmitted where values represent integer sets with only a few elements. However, as mentioned in section 5.5.1, transmitting sequences of zeros equals doing nothing, allowing for an energy-efficient implementation by switching off the radio transmitter.

In order to compute MAX, k communication rounds are performed. In the i-th round, all children send the i-th bit of their value (where $i = 0$ refers to the most significant bit), such that the parent receives the bitwise "or". The parent maintains a variable $maxval$ which is initialized to zero. When the parent receives a one, it sets the i-th bit of $maxval$ to one. The parent then sends back the received bit to the children. Children stop participation in the algorithm if the received bit does not equal the i-th bit of their value as this implies that a higher value of another child exists. After k rounds or time $O(k)$, $maxval$ will hold the maximum among the values of the children. Note that children who sent the maximum will implicitly know, since they did not stop participation. Likewise, children who did not send the maximum can also detect this. Additionally, all children can find out the maximum by listening to *all* messages sent by the parent. The MIN operation can be implemented if the children invert their values before the procedure and if the parent inverts $maxval$ after the procedure.

6.3.2. Vectorial and Parallel Integer Operations

In the previous section we discussed, how a single instance of an integer operation can be performed. In this section we discuss the efficient execution of multiple instances of the same operation. We distinguish two different problems: *vectorial* and *parallel* integer operations.

For a vectorial operation, each child has a vector of n integer values, such that the parent would have to perform an operation n times, where

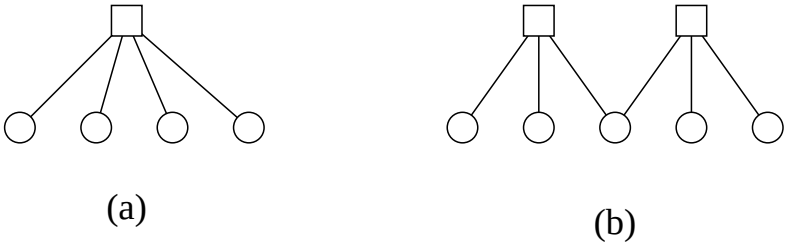


Figure 6.1.: (a) Star network with a single parent. (b) Multiple stars with shared children.

the i-th execution considers the values at position i in the vectors of the children (cf. single instruction multiple data). However, these n sequential operations can be combined into a single operation as described in the previous section, where the messages contain the respective information for all elements of the vector. For k-bit integers, this requires time $O(nk)$. A vectorial operation is more efficient than the independent execution of many operations, since many preambles and radio switches (transmit to receive and vice versa) can be omitted.

For a parallel operation, the parents of multiple stars that share one or more children have to perform the same integer operation as depicted in figure 6.1 (b). If all involved nodes are synchronized, all of the above integer operations can be performed (synchronously) in parallel. For the OR and AND operations, our communication model will ensure that all parents will obtain the correct result for their respective children. For MIN and MAX, the parent will in general not obtain the correct result. However, each child n will find out whether it presented the MIN/MAX value among the children who share a parent with n. As explained in section 6.3.1, a node presented the MIN/MAX if it did not stop participation in the algorithm. Note that any number of such parallel operations on k-bit values can be performed in time $O(k)$. Vectorial and parallel operations can also be combined, requiring time $O(nk)$.

6.3.3. Discussion

The techniques described in the previous sections are sensitive to errors, where one or more bits are not correctly delivered to the receiver. For the discussion of error handling we distinguish two cases. In the first case, one or more nodes send the same bit value concurrently. In this case, traditional error handling mechanisms such as checksums or coding techniques can be used.

In the second case, two or more transmitters send different bits concurrently. According to our communication model, the receiver will see the "or" of these bits. In this case, checksums and coding techniques cannot be applied, since the bitwise "or" of checksums or encoded values is generally not equal to the checksum or encoded value of the bitwise "or" of the original bits. For example, if $b_1 \neq b_2$ are the bits to be transmitted by two nodes and $e(b_i)$ are the encoded bits, then $e(b_1 \mathrm{OR} b_2) \neq e(b_1) \mathrm{OR} e(b_2)$ must be expected.

To detect errors in this case, the bits can be sent unencoded two or more times. If different values are received, then an error is assumed. If three or more transmissions are performed, then errors can be corrected if the majority of the transmissions are identical. Bluetooth uses such a forward error recovery to protect its protocol headers.

Sometimes it is possible to handle transmission errors more efficiently at the application level. Errors can be detected, for example, if constraints (e.g., minimum/maximum/exact number of "1" bits in a received bit vector) are known on the transmitted values. For example, if a node signals a send request, then it should be assigned a time slot for data transmission. Due to bit errors, a node may find that it hasn't been assigned a slot and can retry transmission in a later round. Likewise, if a node didn't signal a send request, then bit errors may lead to the false assignment of a time slot. Although this can result in reduced efficiency, it is not strictly necessary to correct this error.

6.4. Star Network

Using the techniques presented in the previous section, we present a MAC protocol for star networks, which will be used as a building block for the multi-hop protocol presented in the subsequent section.

The protocol supports uplink communication from the children to a parent, and downlink communication from the parent to one or more children. Time-division multiplexing is used to avoid collisions and to ensure a deterministic behavior of the protocol. In order to optimize bandwidth utilization, time slots are allocated on demand to nodes that actually need to send data.

Let us assume for the discussion that children have been assigned small, unique integer IDs in the range $1...N$. We will show in section 6.5 how these can be assigned.

The protocol proceeds in rounds with the parent acting as a coordinator.

A round starts with the parent broadcasting a beacon message to the children. The beacon contains an indicator whether this round is downlink or uplink communication. If the round is downlink, the beacon message will be followed by the payload data, which will usually contain the address (e.g., ID) of the target node(s). If this is an uplink round, children will transmit send requests to the parent. After receiving these requests, the parent constructs a schedule and broadcasts it to the children. From this schedule, children can deduce their time slot for transmission. During their time slots, children send their payload data to the parent. The parent will then acknowledge successful receipt. If transmission failed, the affected children will try a retransmission in the next round.

The beacon serves multiple purposes: it indicates the begin of a new round, carries control information, and synchronizes the children (by means of a preamble). For the send requests and the acknowledgments, an integer set of node IDs (i.e., a bit vector of length N) is used to reduce preamble transmissions as explained in section 5.5.4. A node with ID i is part of such a set if and only if the bit i is "1" in the bit vector.

For the send requests, each child with pending data sends such an integer set containing only its ID without a preamble. The parent will then receive the UNION of these sets as depicted in the lower part of figure 5.4. The parent then sends back the received integer set to the children. Having received it, the children also know the IDs of the nodes with pending send requests. Assuming that nodes with smaller IDs send first, both parent and children can assign time slots for the transmissions. After receiving all the data transmissions, the parent sends an integer set to the children that contains the IDs of the children with successful transmissions.

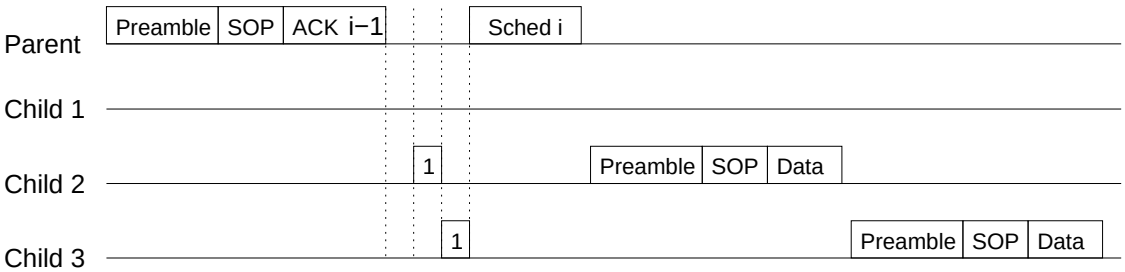


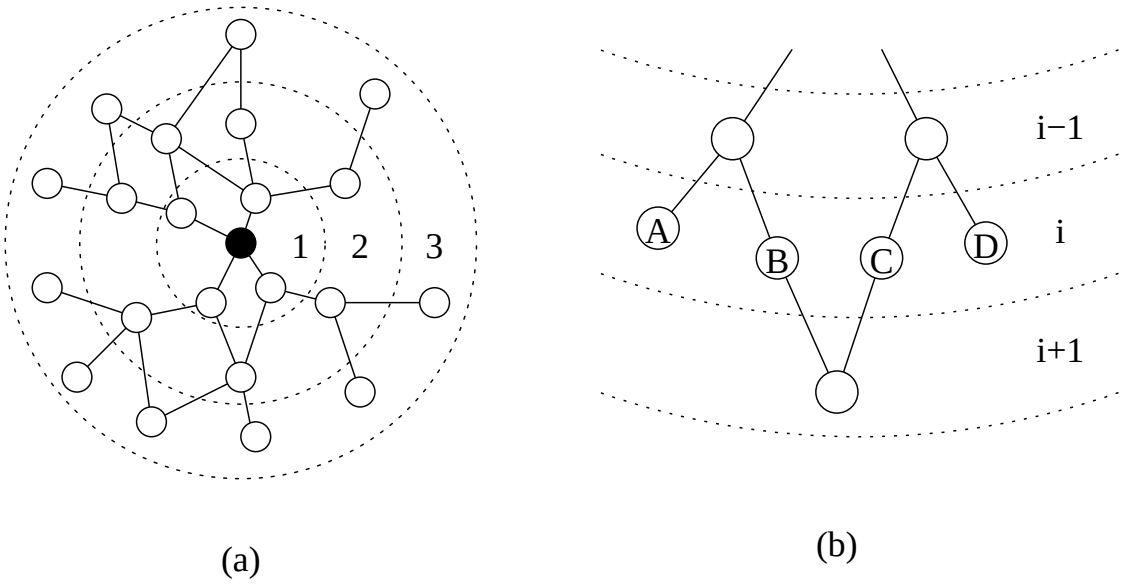
Figure 6.2.: Round i of the optimized MAC protocol for star networks.

As a further optimization, the acknowledgment set can be concatenated with the beacon of the following message as depicted in figure 6.2. Also, the schedule can be sent without an additional preamble, since the preamble that is part of the beacon is sufficient to synchronize the children for

the duration of several hundreds of bits (see section 6.6.1). Hence, preambles are only needed for the beacon and for the payload data packets.

6.5. Multi-Hop Network

We now show how to extend the protocol for star networks to a multi-hop network. This will be based on a spanning tree of the network with the sink at the root, where each internal node and its direct children form a star that uses the protocol presented in the previous section. As to avoid interference, stars are assigned different communication channels.



(a) (b)

Figure 6.3.: Distance from the sink (•) imposes a ring structure on the network. Network links between nodes in the same ring are not shown.

For our discussion let us assume that nodes possess unique MAC addresses (e.g., 16 bit identifiers) and that each node knows its hop-distance from the sink. We show in section 6.5.2 how this can be achieved. Nodes in the network can now be partitioned based on their distance from the sink, such that nodes with equal distance form a ring around the sink. We will denote the ring of nodes with distance i as the ring i. A node in ring i now has one or more parents in ring $i-1$, and zero or more children in ring $i+1$. Note that nodes in ring $i+1$ by definition are *not* within communication range of nodes in ring $i-1$. Nodes in the same ring will not interfere with each other, since they will either all send or all receive at the same time.

6.5.1. Assigning Channels and IDs

In order to turn this hierarchical ring structure into a tree, each node must be assigned to a single parent. A parent in ring i then must share a channel with its children in ring $i + 1$, such that no other parent in ring i of the children uses the same channel. More formally, this requires the assignment of small integers (i.e., channel identifiers) to nodes in ring i, such that nodes who share a child in ring $i + 1$ are assigned different numbers.

We assumed in section 6.4 that children of a single parent are assigned small unique integer numbers. With respect to the ring structure, this task can be formulated as assigning small integers to nodes in ring i, such that nodes who share a parent in ring $i - 1$ are assigned different numbers.

The above two problems can be combined into a two-hop graph coloring problem: assign small integers (i.e., *colors*) to nodes in ring i, such that nodes with the same number do not share a common neighbor in ring $i-1$ or $i+1$. Note that common neighbors in ring i are not considered. In figure 6.3 (b), a valid color assignment would be $A = D = 1, B = 2, C = 3$.

In order to solve this problem, let us assume for now that all nodes in rings $i - 1, i, i + 1$ are synchronized, such that the vectorial and parallel integer operations described in sections 6.3.1 and 6.3.2 can be applied.

Let us further assume that a small number C is known, such that the numbers $1...C$ (the *color space*) are sufficient to solve the coloring problem. We will discuss C in section 6.6.2. In practice, C will be equal to the number of available radio channels.

Under these assumptions, we can use a deterministic variant of the algorithm presented in [39] to solve the above described two-hop graph coloring problem for ring i. For this algorithm, each node in ring i maintains a set P (the *palette*) of available colors, which initially contains $1...C$. Initially, all nodes use the same communication channel.

The algorithm proceeds in rounds. In each round, every node in ring i selects an arbitrary (e.g., smallest, random) color c from its palette P. Some nodes may have selected conflicting colors. For each possible color, the algorithm will allow the nodes with the largest respective MAC address to keep its color. All other nodes reject the color. For this, each node sets up a vector $v[1...C]$ of MAC addresses with one entry for each possible color. $v[c]$ is set to the MAC address of the node, all other entries are zero.

Now, all nodes in rings $i-1$ and $i+1$ synchronously perform a vectorial parallel MAX operation (see section 6.3.2) over the vectors v of the nodes

in ring i. As a side effect of this operation, all nodes in ring i will then know whether or not they are the node with the maximum MAC address for the selected color within two hops. If a node is the maximum, then it keeps color c and is finished. The node will not participate in consecutive rounds. If a node detects that it is not the maximum, it removes c from P and proceeds with the next round. By listening to the values received from the parents as part of the MAX operation, the node can also find out which colors have been selected by other nodes. These are also removed from P.

Since the above algorithm assigns at least one color per round, it requires at most C rounds to finish.

A node in ring i with color c will use the channel associated with c for communication with its children in ring $i+1$. c will also be used by nodes in ring i as the small integer ID for communication with its parent in ring $i-1$ (see section 6.4).

In a final step, all nodes in ring i synchronously broadcast an integer set (i.e., bit vector) that contains the selected color c. Nodes in ring $i+1$ will receive the UNION (see section 6.3.1) of the colors of all their parents. By picking one of the colors and switching to the according communication channel, every node in ring $i+1$ can select a single parent. Note that this selection can be changed at any time, for example, if the parent fails.

Note that the above procedure can be performed for multiple rings in parallel, because rings $i-1, i, i+1$ do not interfere with rings $i+3, i+4, i+5$. Due to this, rings $i, i+4, i+8, i+12, ...$ can be colored in parallel. Hence, four parallel coloring steps are sufficient to color an arbitrary network with an arbitrary number of rings. Since each step requires at most C rounds of the coloring algorithm, an arbitrary network can be colored in $4C$ parallel rounds. Moreover, as each node knows its ring number, it can autonomously decide when to start the coloring algorithm, since it knows how many rounds are required to color the other rings (e.g., after some point in time t_0, a node in ring i waits $C(i \mod 4)$ rounds before starting with coloring).

6.5.2. Time Synchronization and Ring Discovery

In the previous section, we assumed that rings are synchronized and each node knows its ring number r (i.e., distance from the sink). In this section, we show how this can be accomplished.

In the protocol for stars described in section 6.4, the parent sent a beacon

message to synchronize its children. This approach can be adopted to the ring structure (and hence also to trees) as follows. For ring discovery, we also include a *level* field in the beacon message. The sink emits such a beacon with level 1. Nodes that receive this beacon set r to the level contained in the beacon and do themselves emit a beacon message with level $r + 1$. In addition, the node must ignore the beacon sent by its children (which will contain level $r + 2$).

During the setup phase, such a beacon broadcast is performed before each round of the coloring algorithm to keep the nodes synchronized.

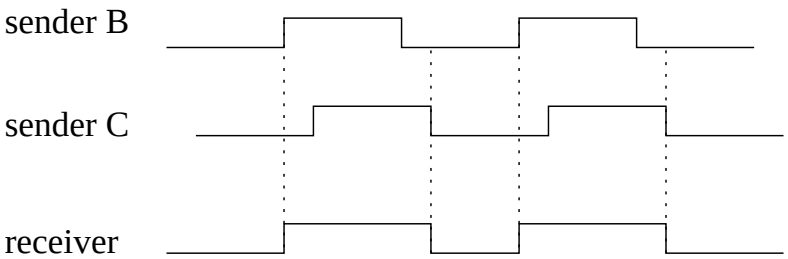


Figure 6.4.: Identical transmissions by two senders with small synchronization errors. The receiver will see slightly stretched "1" bits and slightly compressed "0" bits.

A beacon broadcast can be used to synchronize the receivers with respect to each other with high precision, since the broadcast is received almost concurrently by all receivers [24]. Since the MAC protocol knows the size of the beacon message and bit lengths and has total control over access to the medium, the receivers can also accurately synchronize to the sender of the beacon [26]. As in [24], multiple beacon transmissions can be used to compensate the clock drift of the hardware clocks of the nodes, which allows the nodes to stay synchronized even longer after the transmission of a beacon.

In the initial ring structure used during tree setup, each node may receive the beacon from two or more (already synchronized) parents concurrently. In figure 6.3 (b), for example, the node in ring $i+1$ will receive the beacon concurrently from nodes B and C in ring i. According to our communication model, these identical transmissions are merged, such the receiver will see the "or" of all transmissions. If the parents have a small mutual synchronization error, then the receiver will see slightly stretched "1" bits and compressed "0" bits as illustrated in figure 6.4. However, if the duration of a bit is long compared to synchronization errors, this can be tolerated by the receiver. Hence, the bit length has to be selected

appropriately. See section 6.6.1 for a discussion of this issue.

If the receiver synchronizes to the merged bit sequence of all its parents, it will effectively synchronize to the average of its parents. As a result, multiple parents improve the robustness of synchronization with the parents. Additionally, the synchronization precision errors among neighboring receivers that share multiple parents will benefit from this averaging process. We will analyze synchronization performance in section 6.6.1.

6.5.3. Maintenance

So far we have described the setup phase of BitMAC that organizes the nodes of a deployed network into an interference-free tree with the sink at the root. In this section we discuss what happens if nodes are added or removed after this initial setup. Note that node mobility can be interpreted as removing a node and adding it again at a different place. Please keep in mind that we assumed in section 6.1.1 that node mobility and node additions are rare events.

Let us first consider what happens if a node is removed temporarily or permanently, or if communication is temporarily disturbed. In dense networks, each node will typically possess communication links to multiple parents in the initial ring structure. As the last step of the setup procedure in section 6.5.1, each node was informed of the channels of all its parents, such that the node could choose one parent by selecting the respective channel. We also noted that this selection can be changed at any time without the need for repeating tree setup. Hence, if the selected parent fails, a node can switch over to another parent simply by selecting the appropriate communication channel. If a node "reappears" after a temporary communication failure, no further actions are required. Note that parent re-selection can also be used to balance the load of the parents or to select a parent with the best communication link etc.

If a node in ring i runs out of operational parents, the tree is either partitioned or there are connections to the remainder of the tree via neighbors in ring i or $i + 1$. If the node scans all communication channels and does not receive any beacon messages, the network is partitioned. If the node does receive a beacon, then the node will effectively move to another ring $> i$, since the sender of the beacon in ring $\geq i$ will become the new parent of the node. In this case, the setup phase must be repeated. For this, a special bit is used during signaling of send requests. If a parent receives a "1" during this bit slot, it will propagate the request to its parent and so

on, until the sink is informed, which eventually initiates a reconstruction of the tree.

If a node is added to the network, it will first scan the communication channels for beacon messages from other nodes. Upon receiving a beacon, the node will signal a rebuild request to its parent to enforce a reconstruction.

6.5.4. Operation Phase

Up to now we have discussed setup and maintenance of BitMAC. In this section we discuss the operation phase, where data are transmitted up and down the tree.

For synchronization, the procedure described in the previous sections is used as well. However, at this point every node has selected a single parent, and the stars operate largely independent of each other (see below for a detailed discussion of this issue). Each node will therefore receive the beacon from its single parent only, which is sufficient to synchronize a parent and its children.

As discussed in section 6.4, the MAC protocol for the stars proceeds in rounds, where each round starts with a beacon broadcast. Using the procedure described in section 6.5.2, the rounds are synchronized among the stars. However, in contrast to the setup phase, each node will receive the beacon only from a single parent and bit-level synchronization is only required among a parent and its children. Synchronization requirements across different stars are rather relaxed, since the stars operate independent of each other due to the interference-free channel assignment.

The rounds in all stars are of equal length. Hence, the number of time slots for data transmission in a round is also limited. If in a star more children signal a send request than available time slots, the parent will schedule the maximum possible number of transmission for the round. Children which have not been assigned a time slot will retry in the next round. However, the parent remembers the children with rejected send requests to give them preference in the next round(s).

Note that a node in ring i has to act both as a parent for communication with children in ring $i+1$ and as a child for communication with its parent in ring $i-1$. Therefore, a node will act as a parent in even rounds and as a child in odd rounds.

So far, BitMAC enables nodes to send data to its parent or to its children in the tree. Various approaches can be used to route data over multiple

hops. For the applications described in section 6.1.1, data is either sent from nodes to the sink, or from the sink to some or all nodes, which can be easily implemented. However, tree routing protocols can be used to exchange data between any two nodes in the network. Directed diffusion [36] or TinyDB [51] could also be easily adopted to BitMAC.

6.6. Evaluation

To evaluate BitMAC, we apply a mix of experiments involving actual hardware, analysis, and simulation. Experiments are used to verify the basic operation of the protocol elements on a few nodes. The obtained results are used analytically or in simulations to evaluate BitMAC in larger networks.

In particular, we will analyze the impact of the number of available communication channels, the precision of time synchronization, as well as delays and protocol overhead of BitMAC. For this evaluation, we assume a perfect channel (i.e., no bit errors).

6.6.1. Time Synchronization

In a second experiment, we evaluate the precision of the time synchronization approach described in section 6.5.2 using the setups shown in figure 6.5 (a-d). All nodes are connected by a wire to trigger the start of the experiment at t_0. In (a), the top node starts transmission of a single "1" bit at t_0, which will be received by the bottom node. In (b), the two top nodes start transmitting a "1" bit concurrently at t_0. The receiver (i.e., bottom node) measures the point in time t_c corresponding to the center of the received bit by averaging the points in time corresponding to the rising (t_l) and the falling edge (t_r) of the "1" bit. By observing the variation of t_c over multiple runs, we can estimate the precision of time synchronization. In (c) and (d), the top node starts transmitting a "1" bit at t_0. After receiving the bit and measuring t_c, the middle node(s) start(s) transmitting a "1" bit at $t_c + t_x$ with a small fixed delay t_x. The bottom node measures t_c using the (or-ed) bit received from the middle node(s).

During the measurement of t_c we observed rare (less than 1%) larger variations for t_l, while t_r is rather stable. However, since the duration of the bit t_{bit} is known, we can detect these errors and correct them by setting $t_l := t_r - t_{bit}$ in such cases. With this fix, we performed the above measurements 100000 times to determine the maximum variation t_{err} of

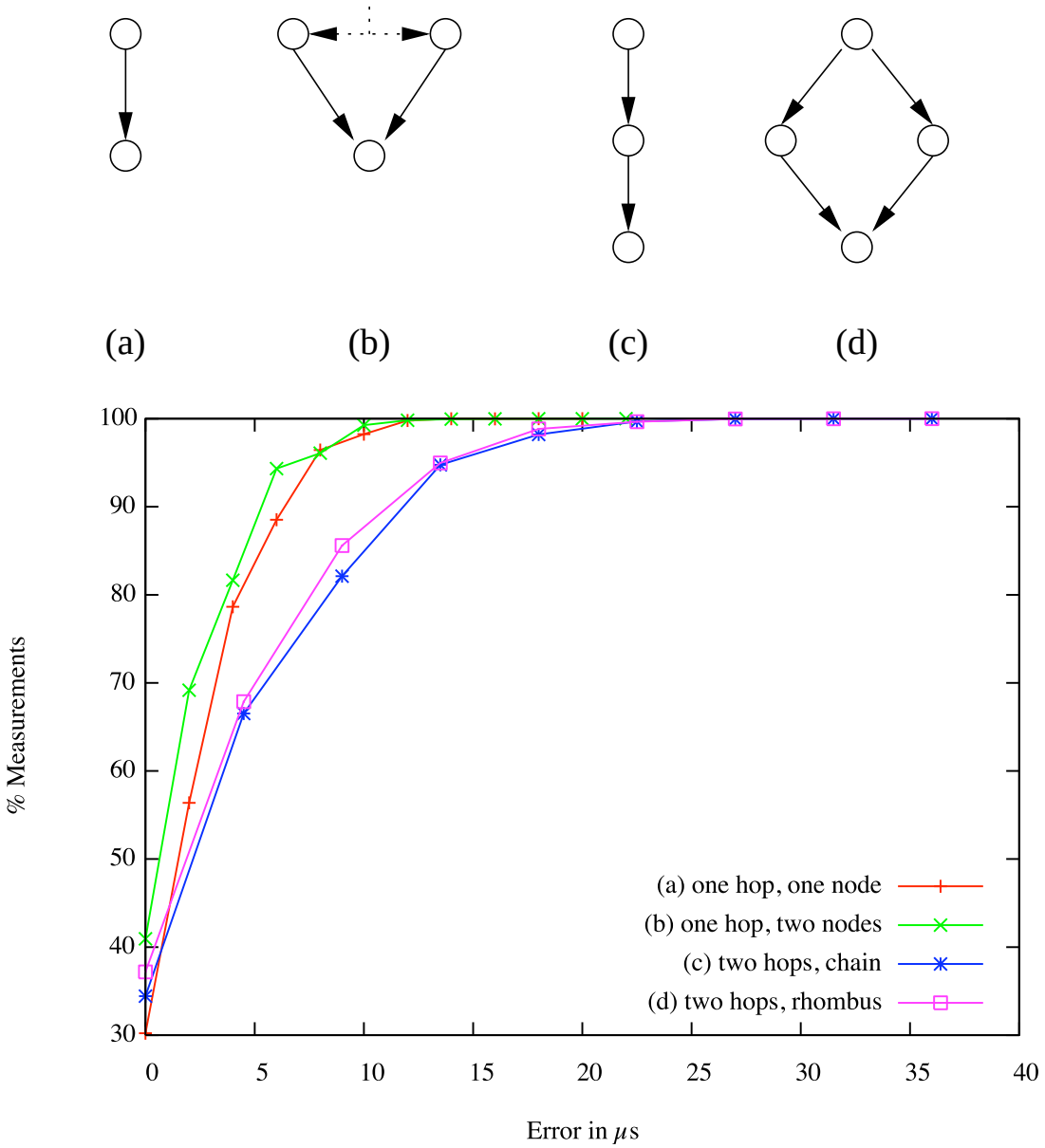


Figure 6.5.: Experiment setups and results for time synchronization. (a) one hop, one sender (b) one hop, two senders (c) two hops, chain (d) two hops, rhombus.

t_c. The results are shown in the diagram in figure 6.5. A point on the curve indicates the percentage (y axis) of the measurements for which the variation of t_c is bounded by an interval of a certain length (x axis).

The diagram shows than the maximum variation in the single-hop experiments (a) and (b) is about 20μs. For the two-hop experiments we would expect about twice the maximum variation of the single-hop measurements. In the experiments we observed a maximum variation of about 36μs, which is slightly smaller then the expected value. As mentioned in section 6.5.2, the results are indeed slightly better if multiple senders are

involved (cases (b) and (d)).

As mentioned in the previous section, we can expect that the above results can be significantly improved if the radio directly supports OOK modulation and provides a digital signal output. Indeed, the authors of [44] report a maximum error of 2μs for the setup (a) when using the RFM TR 1000, which is a 10-fold improvement over our measurement results.

Let us now consider the worst case synchronization error in larger networks. If t_{err} is the worst case error for a one-hop setup, then the worst case error after r hops is rt_{err}. Note, however, that we can expect better results if the network is dense, because then each node has many parents in the ring structure.

In section 6.5.2 we mentioned that synchronization errors lead to stretched or compressed bits. Hence, the duration of a bit t_{bit} has to be long enough to tolerate these effects. Let us assume that bits can still be correctly received if their length is reduced to $t_{bit}/2$ due to synchronization errors. If the radius of the ring is r, then t_{bit} must be at least $2rt_{err}$. For example, if $r = 7$ and $t_{err} = 20\mu$s, then $t_{bit} \geq 280\mu$s, which equals a bit rate of about 3.5 kilobit per second. With $t_{err} = 2\mu$s, we would obtain $t_{bit} \geq 28\mu$s or 35 kilobit per second. Note, however, that these "long" bits only have to be used during the setup phase for control traffic. Data transmissions can use any bit rate and modulation scheme that is supported by the radio.

Let us finally consider for how long two nodes can maintain synchronization using their hardware clocks, assuming they are perfectly synchronized initially. With a bounded clock drift ρ_{max}, at least $1/(4\rho_{max})$ bits can be transmitted until the synchronization error can result in a bit duration smaller than $t_{bit}/2$. For example, the oscillator used on the BTnode provides $\rho_{max} = 100$ ppm according to the data sheet, which allows to send up to 2500 bits without resynchronization. Note that multiple synchronization rounds could be used to compensate the clock drift to reduce the frequency of synchronization.

6.6.2. Network Density

In section 6.5.1 we assumed the knowledge of a constant C, such that the coloring problem can be solved with C colors. We also mentioned that in practice C is set to the number of available radio channels. The Chipcon CC1000, for example, can support up to 130 channels in the 915 MHz ISM band, or 35 channels in the 868 MHz ISM band (assuming a channel

width of 200 kHz). Hence, we have to examine which network density can be supported by BitMAC given a certain C.

It is a well-known fact that a graph with maximum node degree Δ can be colored with $\Delta + 1$ colors. Since the maximum two-hop degree (i.e., number of nodes within distance ≤ 2 hops) of a graph with maximum degree Δ is at most Δ^2, we would need $\Delta^2 + 1$ colors for a standard two-hop coloring, where any two nodes with distance ≤ 2 must be assigned different colors. However, recall from section 6.5.1 that in our coloring algorithm the effective two-hop neighborhood of a node n in ring i is the set of nodes in ring i that share a common neighbor with n in rings $i - 1$ or $i + 1$. We will call the respective two-hop degree the *two-hop ring degree*. We can expect that the two-hop ring degree of nodes in a plane is significantly smaller than Δ^2.

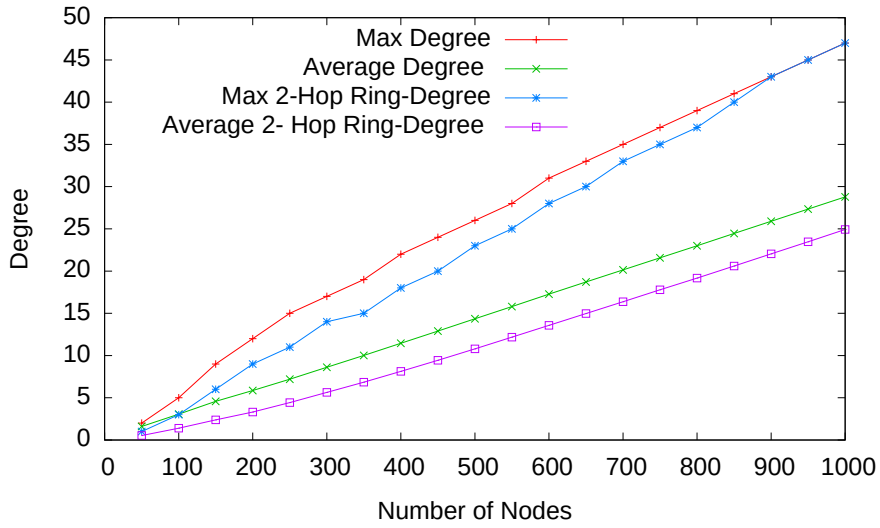


Figure 6.6.: Relationship between node density and node degrees. Nodes with communication range 1 are randomly placed in a 10-by-10 area.

To verify this expectation we performed a set of simulations, where a given number of nodes with communication range 1 is randomly placed in a 10-by-10 rectangular area. The sink is placed at the center of the area. For different numbers of nodes we determined the average and maximum degree, as well as the average and maximum two-hop ring degree. We performed 80 simulation runs and computed averages. The results are depicted in figure 6.6. As can be seen, the average and maximum two-hop ring degrees are even smaller than the average and maximum degree, respectively.

Recall that the Chipcon CC1000 supports up to 35 channels in the 868

MHz ISM band. With this setup we can expect to be able to color random graphs with an average degree of up to about 20.

If the number of channels is not sufficient to color the graph, then some nodes will end up without a color assignment after the coloring algorithm. Such nodes cannot participate in the sensor network. However, the remaining nodes will form an operational network. Due to the high degree, this network is most likely connected.

6.6.3. Setup Phase

In this section we examine the setup phase of our protocol, in particular we derive the amount of time t_{setup} that is needed to setup a network with given parameters. t_{setup} depends on the following parameters: the duration t_{bit} of a single bit as examined in section 6.6.1, the number C of channels, the length L_{mac} of a MAC address in bits, the length L_{beacon} of the beacon in bits, the time t_{rxtx} needed to switch the radio between transmit/receive modes, and the radius r of the ring.

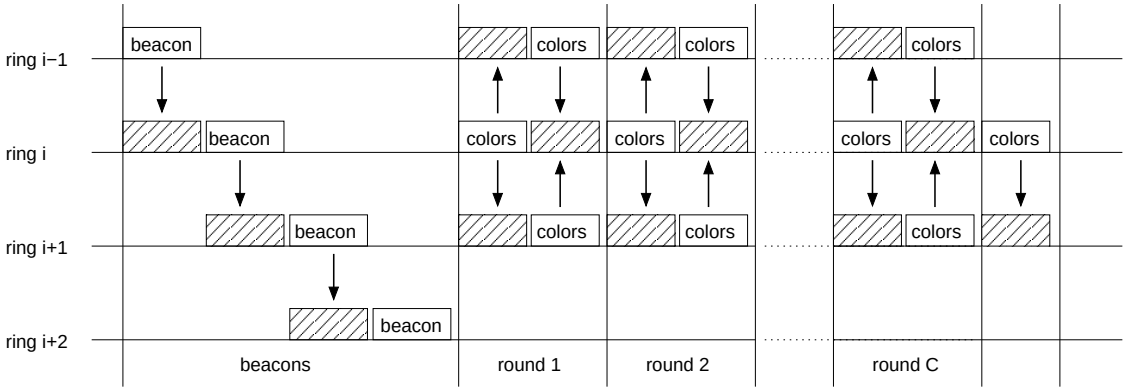


Figure 6.7.: One parallel step of the coloring algorithm on ring i. Shaded packets indicate data receptions.

As explained in section 6.5.1, rings with numbers $i, i+4, i+8, i+12, ...$ can be colored in parallel. Hence, 4 parallel coloring steps are required. One coloring step is illustrated in figure 6.7. Each step consists of a beacon broadcast for synchronization, followed by the coloring algorithm. Recall from section 6.5.1 that the coloring algorithm consists of C rounds. In each round, a parallel vectorial MAX operation is performed. In a final step, colored nodes announce assigned colors to the children. Let us consider the duration t_{step} of such a parallel coloring step, which can be expressed as follows:

$$\begin{aligned} t_{step} &\leq t_{beacon} + C\ t_{round} + t_{announce} \\ t_{beacon} &\leq 4\ (L_{beacon}\ t_{bit} + t_{rxtx}) \\ t_{round} &\leq 2\ L_{mac}\ (C\ t_{bit} + t_{rxtx}) \\ t_{announce} &\leq C\ t_{bit} + t_{rxtx} \end{aligned}$$

Due to the delay caused by the beacon broadcast, coloring of ring r is started at time $rt_{beacon}/4$ after the sink has initiated the setup. Hence, we obtain for the setup time:

$$t_{setup} \leq 4\ t_{step} + r\ t_{beacon}$$

Let us consider the sample network we used for the simulations in section 6.6.2 with 800 nodes. With $r = 7$, $C = 35$, $L_{mac} = 16$, $L_{beacon} = 110$, $t_{rxtx} = 250\mu$s, we obtain $t_{setup} \leq 48$s for $t_{bit} = 280\mu$s, or $t_{setup} \leq 6$s for $t_{bit} = 28\mu$s.

6.6.4. Operation Phase

In this section we evaluate the operation phase, where payload data is transmitted from nodes towards the sink. Let us assume that each round consists of S slots for data packets. Recall from section 6.5.4 that each node acts as a child in even rounds and as a parent in odd rounds. Hence, when a node wants to transmit, it may have to wait up to two rounds until it can transmit a send request to its parent. Then the packet is forwarded one hop towards the sink in every round. Since the overlay tree is a shortest-path tree from the sink to all nodes, packets are always forwarded on the shortest path to the sink. Hence, it takes at most $i + 2$ rounds to deliver a packet from a node in ring i to the sink, provided that a free slot can be allocated in each round. If a free slot cannot be assigned immediately, then a node may have to wait for $2\lfloor C/S \rfloor$ rounds until a slot is assigned, since a parent cannot have more than C children.

As with most other MAC protocols, there is a trade-off between latency and energy consumption in BitMAC. If the duration of a round (i.e., S) is increased, the latency will increase and energy consumption will decrease (due to fewer beacon transmissions). The energy consumption of an idle network is dominated by the duration of a round and the length of the beacon. In every idle round, each non-leaf node has to receive and

transmit a beacon (without acknowledgments and schedules), and has to listen to C send-request bits from its children. A child node only receives and forwards the beacon. Hence, we obtain for the average radio-on time:

$$t_{on} \leq 2\ (L_{beacon}\ t_{bit} + t_{rxtx}) + C\ t_{bit}/2$$

Let us consider an example, where $C = 35$, $t_{bit} = 52\mu$s (19200 baud), $t_{rxtx} = 250\mu$s, and $L_{beacon} = 110$. With these parameters we obtain $t_{on} \leq 13$ms. If the duration of a round is 200ms, then $S = 9$ data packets of 32 bytes each can be delivered in each round. The duty cycle of the nodes in an idle network is then about 6% on average.

If the network is not idle, then parents have to send schedules and acknowledgments, resulting in an additional $2C$ bits. Overall, the time per round that cannot be used for data transmissions is:

$$t_{overhead} \leq 2\ L_{beacon}\ t_{bit} + 4\ t_{rxtx} + 3\ C\ t_{bit}$$

Note that $t_{overhead}$ can be interpreted as the overhead of our protocol compared to an ideal protocol, where all nodes know a priori when to transmit data packets without collisions. For the above example we obtain $t_{overhead} \leq 18$ms, such that about 9% of the bandwidth cannot be used for payload data transmissions if a round lasts for 200ms.

6.7. Discussion

In the protocol description, we assumed that links between nodes are either error-free or nonexistent. A more realistic model to characterize communication links is the distinction of communication range, in which the communication is more or less error-free, and interference range, in which data transmitted from a node A to node B does harm node C's reception of data sent by node D. By design of BitMAC, there are no collisions between nodes in communication range, so we would like to discuss issues related to nodes in interference range here.

Nodes on the same ring either transmit or receive at the same time, hence neither collisions nor interference do occur on one ring. Also, by the way the rings are constructed, we conclude that transmissions from nodes on ring $i + 1$ to nodes on ring i do not interfere with parallel transmissions from ring $i - 1$ to $i - 2$ (3 hops). Even if they would, we could insert a "buffer ring" to avoid such interferences.

The vulnerable part of BitMAC is the channel and ID assignment. If there are two nodes A and B which do not have a common child or parent, they might choose the same channel independently. Not having a common child refers to not having a node in the communication range. Yet still node A might have a child node C which might be in interference range to node B and disturb transmissions from B's child nodes.

One way to alleviate this problem is to ensure that such a node C is seen as a common child node of A and B during coloring. This can be done by temporarily increasing the radio power in such a way that the communication range during this phase equals the interference range of the later communication. It is important that the radio power is increased only during coloring, otherwise the situation would not improve. By doing this, additional parent-child links are created which are not available at normal transmission power. Therefore, all links should be validated by an additional link probing phase.

6.8. Summary

We have presented and analyzed BitMAC, a deterministic, collision-free, and robust protocol for dense wireless sensor networks. BitMAC is based on an "or" channel, where synchronized senders can transmit concurrently, such that a receiver hears the bitwise "or" of the transmissions. Using the BTnode Rev. 3 platform, we have shown the practical feasibility of this communication model and analyzed the performance of time synchronization. We gave deterministic bounds on the execution time of all protocol elements and showed that the protocol overhead is small compared to an ideal protocol.

7. BurstMAC

Although the BitMAC protocol presented in the previous chapter fulfills the requirements for an efficient data-gathering protocol that can handle correlated traffic bursts, it still has some drawbacks, which make it unpractical for some realistic scenarios. The two main drawbacks of BitMAC are that changes in the network topology, especially the addition of nodes, are not handled well and that the communication pattern is limited to convergecast, which does not allow a node to communicate with all neighbors.

In this chapter, we present the BurstMAC protocol that builds upon successful components of BitMAC, but introduces some indeterminism to remedy the limitations above.

This chapter is structured as follows. We first give an overview of BurstMAC and its key techniques before we explain the protocol in detail. Then, we describe the implementation of BurstMAC on BTnodes. Finally, we report performance results, before concluding the chapter.

7.1. Protocol Overview

In this section, we present the key ideas behind BurstMAC and outline the basic protocol structure. A detailed discussion of the various BurstMAC components can be found in section 7.2. BurstMAC combines a number of techniques to provide high throughput and efficiency under load *and* low idle overhead. Most notably, scheduling and the use of multiple radio channels enable high throughput, while cooperative transmissions and techniques to eliminate preambles guarantee low idle overhead. Also, the different techniques are integrated in an innovative manner. For example, scheduling of time slots and assignment of channels is combined to reduce the overhead further.

One important underlying assumption on the radio is that a sufficient number of radio channels is supported. In particular, that number should be larger than the maximum number of two-hop neighbors of a node in the network. In our implementation on the ChipCon CC1000 radio, we

use 32 data and two additional control channels as detailed below.

7.1.1. General Approach

To avoid collisions, BurstMAC operates in synchronous rounds. Each round consists of N frames with N=32 in our implementation. Every frame contains a small CONTROL section and a large DATA section as depicted in Fig. 7.1. To maximize throughput and to allow for collision-free communication during the DATA section, BurstMAC uses N interference-free data channels and one control channel. The CONTROL section is used for time synchronization, to broadcast other information to all network neighbors, and to assign color ids to nodes. As a result of the latter, each node is assigned a color id $c \in 1..N$ that is unique within two hops. The color id c is used for two purposes. Firstly, a collision-free TDMA schedule of the control channel is implemented, such that a node with color id c sends a control message in the CONTROL section of frame c. Secondly, node c coordinates multiple senders on radio channel c during the DATA section, which allows it to receive data without collisions. The set of nodes which act as coordinators during a frame is determined by a *coordination-free transmission scheduling*, as described in the next paragraph. By using multiple channels, contrary to basic TDMA, nodes are able to send packets during multiple frames, which increases the total network bandwidth.

To achieve a low duty cycle in idle situations, the frame length is chosen rather large, 1s in our implementation. Therefore, a node is required to turn its radio on for the duration of the (short) control message in every frame. During the data section, coordinator nodes need to check for neighor transmissions. Due to the coordination-free transmission scheduling described below, a node is a coordinator in every other frame on average. This results in an idle duty cycle below %1.

7.1.2. Coordination-free Transmission Scheduling

As a node cannot send and receive at the same time due to the half-duplex nature of typical low-power radios, some form of coordination is required among the nodes to achieve agreement on when to send and when to receive. To realize this coordination without introducing additional control messages, BurstMAC uses the following communication-free communication approach. During each frame, a node is either in transmit or receive mode, that is, it can either only transmit or only receive data during the

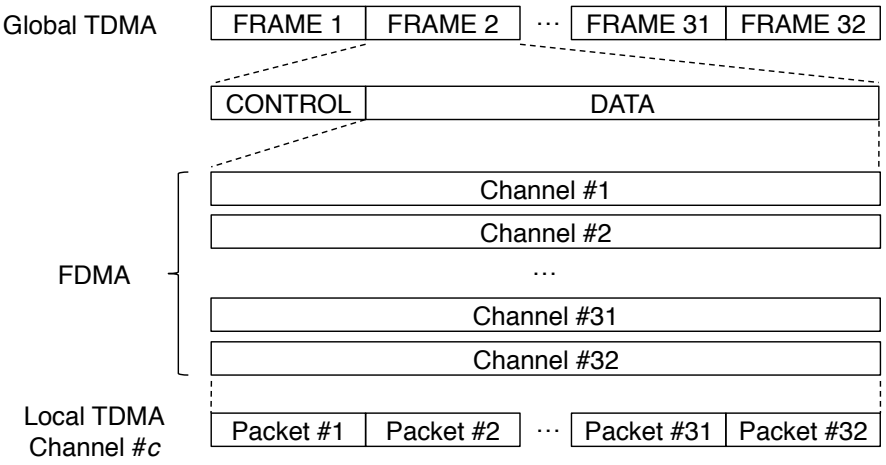


Figure 7.1.: BurstMAC protocol overview: Synchronous rounds consist of 32 frames (global TDMA). Each frame contains a CONTROL and a DATA section. Multiple radio channels are used for interference-free communication of collocated node clusters (FDMA). Communication within a node cluster is coordinated by local scheduling (local TDMA).

whole frame. The choice of mode is controlled by a pseudo-random number sequence that is seeded with the unique 16-bit node id. Knowing the node ids of its neighbors, a node can not only compute its own current mode, but also the current modes of its neighbors. If node A wants to send to neighbor B, then A has to wait for a frame when it is in send mode and B is in receive mode. A uses B's channel for the actual transmission. This approach avoids extra traffic for coordination among nodes completely.

This approach is loosely related to pseudo-hopping sequences used, e.g., in Bluetooth, or the uniform distribution of wake-up times in JAVeLEN [38], and avoids extra traffic for coordination among nodes.

7.1.3. Packet Bursts

To further increase throughput and reduce communication overhead, a sender can request the transmission of multiple packets in a row, eliminating lengthy preambles for all but the first packet. In contrast to the standard approach of sending packets back-to-back, BurstMAC reliably transmits the packet lengths of each individual packet and provides each packet with a checksum to detect bit errors. By this, the checksum of each packet can be checked separately and a single bit error does not cause the whole packet train to be dropped. The receiver replies a bit vector with a one bit for each packet that has been received correctly.

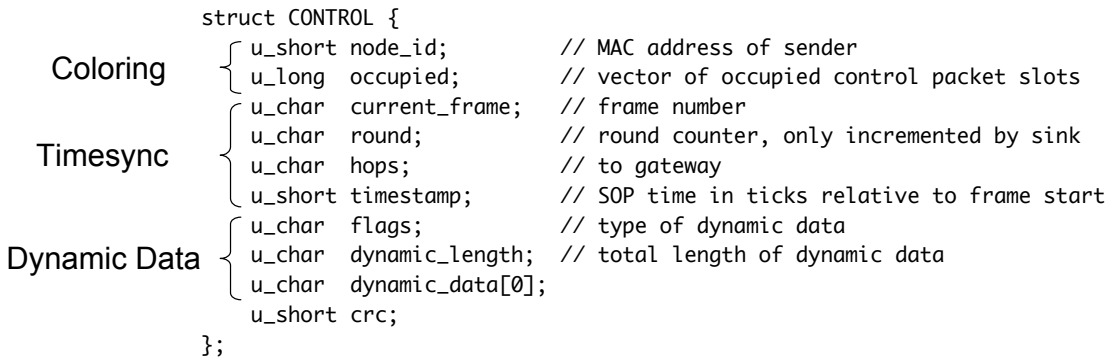


Figure 7.2.: Contents of the control message.

7.1.4. Cross-Layer Optimizations

Typical routing protocols such as MintRoute [89] need to perform neighbor discovery and link quality estimation, which requires each node to broadcast beacon packets at regular intervals. However, due to the existence of the control packets in BurstMAC, we can integrate neighbor discovery and link estimation into BurstMAC without additional overhead.

7.2. Protocol Details

In this section, we provide a detailed description of the BurstMAC protocol. Key functional components channel assignment, cross-layer support for routing (i.e., neighbor discovery and link estimation), actual data transmissions, as well as time synchronization (i.e., establishing a common time among the nodes in the network) and network startup (i.e., how nodes join a BurstMAC network).

The key protocol element to enable the above functions is the CONTROL section, where a node broadcasts a control message to all its neighbors on a common control channel in frame c of each round, where c is the color id that has been assigned to the node. As shown in figure 7.2, the control message consists of a fixed part which is always present with basic information for time synchronization, and coloring, as well as a dynamic section which contains one or more optional headers. A `flags` field indicates which of the optional headers are present in a message. We will reference the various field of the control message throughout this section.

7.2.1. 2-Hop Coloring

Each node has to be assigned a color id c that is unique within two hops. As discussed in section 7.1, c is used for two purposes: as a channel id for payload data transmissions and to schedule broadcasting of control messages on the control channel.

For coloring, all nodes keep track of the frames used by their neighbors for sending control messages similar to LMAC [81]. As a node with color id c transmits a control message in frame c, each node is aware of the colors assigned to its neighbors and periodically broadcasts a bit vector of these occupied color ids in its control message (field `occupied`). A newly joining and yet uncolored node with id i receives the list of used color ids in the control messages of all of its neighbors. The union of these sets equals the set of color ids used in its 2-hop neighborhood. The new node then randomly picks a color id c from the remaining free colors and transmits its control packet in frame c in the next round. A special flag in the control message requests other nodes to echo the node id contained in the control packet of frame c. If another node with node id j simultaneously picks the same color c, both node ids i and j will be reported for frame c by different neighbors. In this case, both newly colored nodes pick another free color at random.

Topology changes in the network may lead to a situation where two nodes with the same color id are two hops or less apart. This situation must be detected and at least one of the nodes must pick a new color id. For detecting such a situation, each node monitors the node id (field `node_id`) contained in the control messages. If a node observes that the node id of the control message in frame c is different from the node id of the control message in frame c during the previous round, the observing node reports in its next control message that nodes with color c should pick a new color according to the above procedure. Here we exploit the capture effect [88] – where a node will receive the stronger of two concurrent transmissions on the same channel rather than seeing a collision – paired with the spatial diversity of the nodes.

7.2.2. Transmission Scheduling

As discussed in Sect. 7.1, each node is either in receive or send mode during each frame, using a pseudo-random generator to control the choice of mode. Using these pseudo-random sequences, a node can compute whether it can send to a certain node during a certain frame. However,

more than one sender may want to transmit to a single receiver during the same frame on the same channel. Hence, these senders have to be coordinated in some way to avoid collisions. A key goal of BurstMAC is to minimize this coordination overhead in the idle case, where no sender wants to transmit.

Our solution is illustrated in Fig. 7.3, which shows the DATA section in more detail, time increases from left to right. One node is in receive mode, and two nodes in send mode (with colors $c = 1$ and $c = 3$) want to transmit a packet to the receiver. In segment A, both senders employ cooperative transmission and concurrently transmit a short jamming signal (i.e., unmodulated carrier signal) during 500 us to indicate a send request. If the receiver does not detect a jamming signal (i.e., if RSSI always smaller than a certain threshold during these 500 us), it can go back to sleep, optimizing the overhead in the idle case.

However, the simultaneous transmission of jam signals of multiple senders may result in destructive interference at the receiver, such that the receiver won't detect the presence of multiple jam signals. This problem can be addressed if senders transmit on slightly different frequencies, such that a beat with a certain period time will result at the receiver. If the receiver listens for at least that period time, it will encounter non-destructive interference at some point in time and detects the jam signal. Fortunately, there is a natural frequency diversity due to variations of crystal frequencies among nodes. For a 868 Mhz carrier signal, for example, a crystal frequency difference of 10 ppm between two transmitters results in a beat with a period length of 115 us, and a high probability for the receiver to detect the transmission if it listens for at least this amount of time. In our implementation, we used 500 us, which further increases the chance that a receiver will encounter non-destructive interference. In tests, we could not observe a significant drop in the number of detected jam signals when using multiple transmitters instead of a single one. If this natural diversity should not be enough to prevent errors from destructive interference on other hardware, additional frequency diversity could still be created explicitly as detailed in [43].

If a receiver detects a jam signal (i.e., if there is at least one sender), then the receiver exerts the single-bit transmission technique by transmitting a minimal sync packet consisting of a preamble and a start-of-packet symbol in segment B in Fig. 7.3. The sync packet allows a sender to accurately synchronize with other senders and the receiver. Each sender then transmits a single jamming "bit" in the bit slot which corresponds

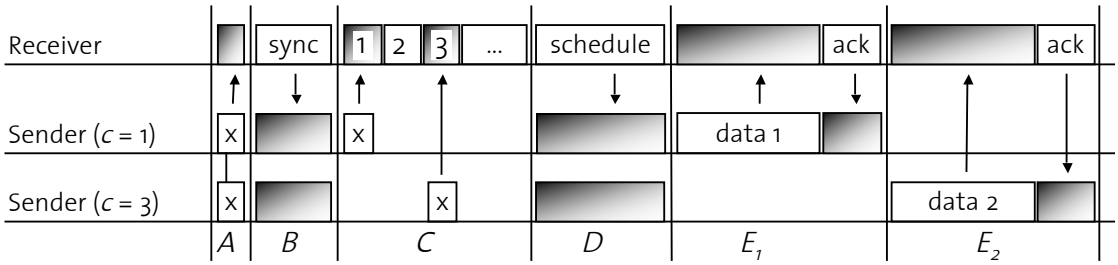


Figure 7.3.: Data SECTION with two senders transmitting a packet to the receiver. Cooperative transmission is used in segment A and single bit transmission in segments B and C to identify the senders.

to its color id c in segment C in Fig. 7.3. Based on the list of senders, the receiver then computes and broadcasts the transmission schedule in segment D. This schedule is essentially a copy of the bit vector received in section C, where bit i is set iff a sender with channel i has submitted a send request. However, if the receiver has insufficient buffer space to store all packets, it will randomly erase a bit until the number of remaining bits equals available buffer space. This way, we achieve a simple form of flow control to avoid packet loss due to buffer overruns.

A sender with channel i checks if bit i in the received schedule bit vector is set. If this is the case, the sender transmits a data packet in slot E_j, where j is the number of "1" bits in the schedule bit vector with an index smaller than i. The slots are of fixed length such that each sender can compute start and end of its slot from the schedule bit vector. To deal with packet loss caused by bit errors, each data packet is immediately acknowledged by the receiver. The sender retransmits the packet if it does not receive the acknowledgment in time. To eliminate duplicates caused by lost acknowledgments, each data packet contains a small header with a one byte sequence number, which also allows FIFO delivery of packets in the receiver.

The header of the data message also contains several flags. One of them, the `more` flag, can be set by the sender to indicate that it needs another slot to send a further packet. If available, the receiver will reply an unused slot k in the acknowledgment, such that the sender can send the next packet in slot E_k.

7.2.3. Packet Bursts

To reduce the probability of packet loss due to bit errors, packet sizes are rather small in BurstMAC as well as in all other MAC protocols for sen-

sor networks. However, this results in substantial overhead due to several reasons. Firstly, each data packet and acknowledgment needs to be preceded by a long preamble to synchronize sender and receiver at the bit level. In many cases, the preamble is longer than the actual payload data. Secondly, both sender and receiver need to switch the radio back and forth between transmit and receive mode, which consumes time and energy.

To reduce this overhead, BurstMAC offers a so-called burst mode, where a sender can use almost the whole DATA section of a frame to send a sequence of packets back-to-back preceded by only a single preamble and acknowledged with a single message. Each individual packet in the sequence has its own CRC, such that a bit error destroys only a single packet and not the whole burst[1]. The acknowledgment message contains a bit vector of the packets that have been successfully received.

To initiate a burst, a sender first sends a normal data packet with the `more` and `burst` flags set in the header of the data message, indicating that is has more data to send and wants to use burst mode. To ensure fairness, the receiver does not grant burst access to the first sender requesting it, but waits for a (small) random number of burst requests until granting access. Otherwise, the sender with the smallest color id would always be granted burst access because it is assigned the first slot, resulting in unfair behavior.

When granting burst access to a sender, the acknowledgment message will contain the first slot the sender can use for the burst. The burst will then extend until the end of the frame. The receiver will also stop assigning slots to other senders at this point. Next, the sender transmits a message containing the lengths of all packets (up to 51) in the burst. As each packet in the burst has a maximum length of 32 bytes, the length can be encoded into 5 bits per packet. The receiver needs this length information in order to split the burst into packets for checking their individual CRCs. As a packet burst without length information is useless to the receiver, length info and the actual data are sent in separately acknowledged packets. Upon reception of an acknowledgment from the receiver, the sender starts transmitting the packets back-to-back with individual CRCs, but without individual preambles. After all packets are transmitted, the receiver sends a single acknowledgment which contains a bit vector with a one bit for each packet that has been received correctly. Using this bit

[1]Note that modern radio transceivers are can receive arbitrarily sized packets without loosing the bit synchronization, even in the presence of bit errors. Therefore, the main caveat with sending packets back-to-back is the risk of a bit error in the length field of a header, as the start of all later packets would be missed and, hence, would have to be discarded.

vector, the sender can retransmit any lost packets in a future burst.

7.2.4. Cross-Layer Support for Routing

Most routing protocols need to perform neighbor discovery and link quality estimation as a prerequisite to computing routes. For this purpose, those protocols (e.g., MintRoute [89]) broadcast periodic beacon messages. In BurstMAC, we can use existing control messages to derive this information without introducing additional overhead. As these messages are free of collisions, we obtain better link qualities than in CSMA protocols, where beacon messages may collide in dense deployments, resulting in an incorrect link quality estimation.

BurstMAC offers an interface to retrieve the list of neighbors and their link qualities. As an exemplary routing protocol, we implemented a convergecast to the sink based on MintRoute [89] on top of BurstMAC. For this, each node periodically broadcasts its list of neighbors and link qualities in the control message. Receiving this information, each node computes a bidirectional link quality estimate for each neighbor and picks a set of "good" neighbors. Using only good neighbors, a distance vector approach is used to compute a spanning tree consisting of the best quality path from each node to the sink.

7.2.5. Time Synchronization

As BurstMAC makes use of synchronous rounds, we need some form of global time synchronization to make sure that rounds and frames begin at the nodes at approximately the same time. Note that the required synchronization accuracy is rather low as the time-critical events in BurstMAC are the transmission of control messages and the single-bit transmission at the beginning of the DATA section. We set the duration of the single-bit to 500 us as described in Sect. 7.2.2. A maximal synchronization error of half that duration between two neighboring nodes ensures an overlap of at least 250 us among neighbors for detection of the jam signal.

For time synchronization, we assume that a dedicated node provides a time reference for the whole network. Although this is not a requirement, the data sink will also act as the time reference in most practical settings. In contrast to typical tree-based synchronization protocols where each node synchronizes to a single parent node, BurstMAC uses a more robust and accurate approach where each node synchronizes to the average

time of all its parents (i.e., all nodes that are closer to the time reference than itself).

An important issue that needs to be dealt with is the dynamic nature of wireless links, that is, the set of parents of a node changes over time. In fact, a parent of a node may turn into a child (i.e., a node that is farther away from the reference as the node) due to a broken link. Here, we must avoid synchronizing to the former parent, which has become a child. To deal with this issue, the control message contains both a `hops` counter that holds the distance of the sending node from the reference, and a `round` number. The latter is incremented by the reference node before broadcasting a control message, all other nodes broadcast a copy of this value without incrementing. If a node receives a control message from a parent (i.e., a node which has a smaller hop counter than the node), whose round counter is the same as in the previous message from this parent, then the parent may have turned into a child and is not used during time synchronization of the node.

Time synchronization is realized by accurately timestamping the transmission and reception of the first bit of the control message in the radio interrupt handler. With this, we obtain a per-hop synchronization accuracy of few microseconds.

A node performs these measurements for two consecutive control messages from each parent and computes the duration of a round for each of the parents, which may differ from parent to parent due to different clock rates and drift. The node then computes the average round duration of its parents and adjusts its local round duration, increasing/decreasing the number of clock ticks of a frame if the average round duration of the parents is larger/smaller than its own round length. As the duration of a clock tick of a typical 32 kHz real-time clock is too large for this correction (resulting in a minimal increment per round of 30.5 $\mu s \times$ 32 frames), we use a variant of Bresenham's algorithm [9] to adjust the *average* frame length in smaller increments using only integer arithmetic. This approach proved to be very robust to outliers.

7.2.6. Network Startup

Network startup is concerned with the problem of how nodes join a BurstMAC network. In fact, the main problem is getting in sync with the time reference. For example, during deployment nodes are switched on in random order, so each node needs to wait until it gets connected to a time

reference. As this waiting process may last quite long, we need to make sure that nodes do not spend much energy in this waiting state.

Our approach uses a dedicated wake-up channel where nodes transmit wake-up signals. Communication on this channel uses cooperative transmissions, i.e., concurrent transmissions by multiple nodes overlay in a non-destructive manner. On this channel, every node transmits a very short jamming signal (100 μs), called *Blip*, at the begin of *every* CONTROL section as depicted in figure 7.4. By this, a new node joining the network has to scan the wake-up channel at 100% duty cycle only for a single frame instead of a whole round. On detection of the Blip, the node already has approximate timing information on the start of the CONTROL section and will receive at least one control message in the next 32 frames. If a node starts up and the network is not active yet, it will not receive a Blip during one frame and start low-power listening with a period T slightly smaller than the duration of a frame. That is, the node wakes up every T time units, scans the wake-up channel for ongoing transmissions, and goes back to sleep. When the time reference starts up, it sends a pulsed jamming signal during the first frame to wake up nodes from low-power listening. Each jamming pulse starts exactly at the begin of the CONTROL section, providing woken-up nodes with a rough time information similar to the Blips. The idle-listing period T equals the duration of a wake-up pulse, which is slightly shorter than the frame length. Nodes that have been woken up this way will transmit a wake-up signal themselves to wake up nodes further away from the time reference, such that eventually the whole network will be activated and synchronized.

As depicted in figure 7.4, Blips and wake-up pulses have been designed such that if some nodes transmit a Blip and others transmit a wake-up pulse simultaneously, a receiver will see a wake-up pulse due to non-destructive interference.

Upon receiving the first Blip or wake-up pulse, a node obtains rough time information, which is sufficient to receive a control message in the next 32 frames. When it receives the first control message, a node sets its local clock to the time of the sender. After that, only the frame lengths are adjusted to stay synchronized as described in the previous section.

7.3. Implementation

We implemented BurstMAC on BTnode Rev. 3 nodes. The CC1000 on the BTnode is configured for the 868 MHz ISM-band, where the actual

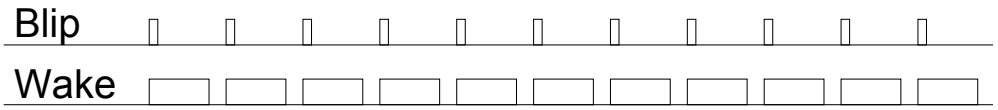


Figure 7.4.: (Top) Periodic start-of-frame signal: Blip, (Bottom) Wake-up signal with integrated start-of-frame information. The duration of Blip and wake-up in our prototypical implementation are 0.1 ms and 999.8 ms respectively/

baseband frequency within this band is configurable by software. We used 34 channels of which one is reserved for control broadcasts, one is used for the wake-up mechanism and the other 32 are used for data communication. The analog RSSI output of the CC1000 is used for clear-channel assessment, cooperative and single-bit transmission. We further make use of the CC1000's ability for precise MAC layer timestamping in the order of 10 us [54].

For the implementation, we used a bit rate of 19200 bps and a frame length of 1 s, which is split into 50 ms for the CONTROL section and 950 ms for the DATA section. By this, the DATA section provides 24 data slots for single bit transmission or up to 51 packets in burst mode. Each packet contains a type field and up to 32 bytes of payload. In this configuration, a node can send or receive a single packet and up to 51 burst packets of each 33 bytes in a single frame. Such a transfer of 1716 bytes results in a maximal usable bandwidth of 71.5% of the total bandwidth of 2400 bytes/s.

In fact, BurstMAC was also used to perform firmware updates during the experiments and to collect measurement results from the network, thus demonstrating BurstMAC's reliability.

7.4. Evaluation

We study the performance of the BurstMAC implementation on BTnodes. In particular, we investigate the accuracy of time synchronization, the idle overhead, as well as overhead and time to completion of correlated bursts. We compare BurstMAC against two established protocols that represent the two ends of the design space that are relevant for our work. Firstly, we choose SCP-MAC as a state-of-the-art contention-based protocol with very low idle overhead. Also, SCP-MAC does contain an adaptive channel polling mode which allows it to adapt to traffic bursts. Secondly, we chose LMAC as a scheduling-based and, hence, collision-free protocol that has been shown to outperform many other MAC protocols under high

data rates [46].

7.4.1. SCP-MAC and LMAC

In order to be able to compare the three protocols on a common hardware platform, we have implemented SCP-MAC and LMAC on the BTnode. In fact, we can switch between all protocols at runtime, performing firmware updates and collecting measurement results with BurstMAC, while the actual experiments make use of SCP-MAC or LMAC (or BurstMAC). The implementations are based on published papers and source code, with some additions and parameter choices to enable a fair comparison.

For LMAC, we used a frame length of 150 ms. By this, it is possible to send up to 300 bytes or up to 7 packets of maximal size in one frame similar to [81]. As LMAC provides only a single ACK per frame, increasing the frame length would result in a higher chance that all packets sent in a frame have to be dropped if a single bit error occurs. In LMAC, one or multiple payload packets are sent directly after the control packet. As described in a technical report [82], multiple packets are sent back-to-back with a single CRC at the end. The receiving node(s) acknowledge the reception of data packets in the control message of its own slot. If no acknowledgment is received, the packets will be retransmitted. If an acknowledgment gets lost, the destination will receive the packets a second time. Our only modification to LMAC is the use of sequence numbers as in BurstMAC to suppress duplicate packets.

We implemented SCP-MAC as described in [91], using the published source code for clarification. We used 8 contention slots before the wake-up tone and 16 for the second contention window as described in the paper. The duration of a single contention slot is 427 us, which is the minimal time for a node to assert a free channel, switch to transmit, and allow other contenders to detect this transmission at the start of the next contention slot. We also implemented overhearing avoidance by checking the destination address in a packet before it is completely received. Our implementation differs from the original paper in two aspects: time synchronization and acknowledgements. For the time synchronization, we embedded SCP-MAC into the DATA section of BurstMAC, using the CONTROL section of BurstMAC for time synchronization. By this, the channel polling period in SCP-MAC is identical to the BurstMAC frame length of one second. As BurstMAC's CONTROL section serves more purposes than just time synchronization and because SCP-MAC could tol-

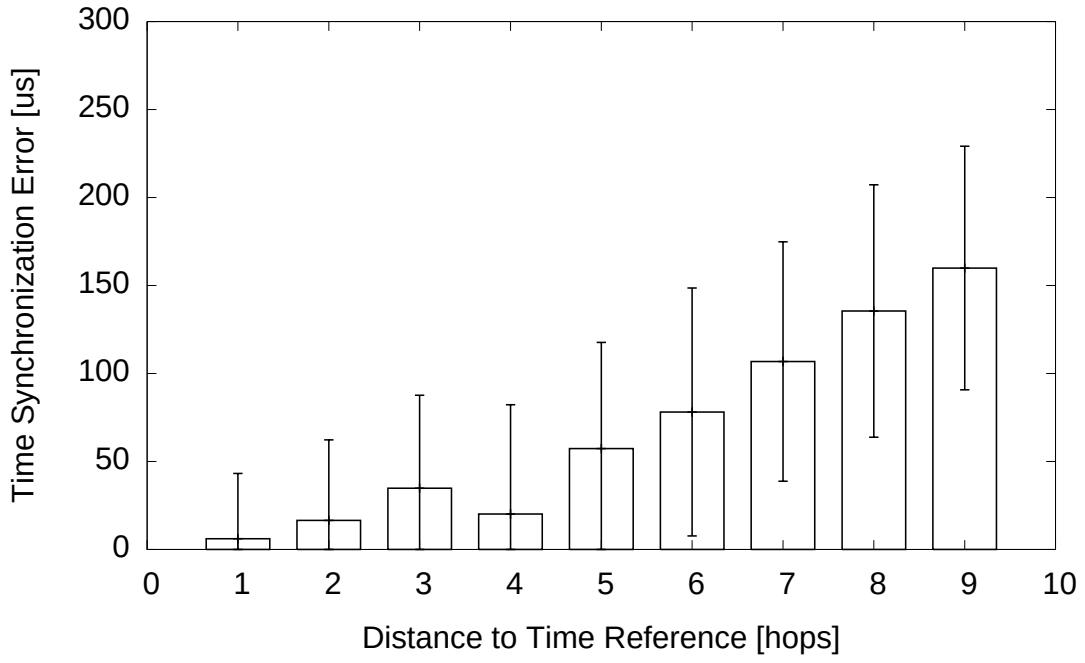


Figure 7.5.: Accuracy of time synchronization on a chain topology.

erate less frequent time sync updates, we exclude the radio time used for the CONTROL section from SCP-MAC's energy measurements. By this, SCP-MAC gains a slight advantage over both LMAC and BurstMAC, but this advantage is relevant only in an idle scenario. To allow for efficient and reliable link-layer packet transmission, we modified SCP-MAC to let nodes acknowledge the reception of a correct packet. We believe that this is favorable compared to other mechanisms, as for unicast packets, only the sender and receiver are active at this time allowing for a contention-free transmission of the acknowledgment.

7.4.2. Time Synchronization

We study the accuracy of time synchronization as a function of the network diameter to investigate the maximal network diameter that BurstMAC can support. For this, we arrange 10 nodes in a chain topology with the time reference at the end, forcing each node to use only its direct parent as a reference for synchronization. However, all nodes are within communication range of the time reference, such that each node can directly measure its synchronization error with respect to the time reference. We ran this setup for one hour, where each node measures its synchronization error once per round. After collecting the measurements from the network, we computed averages and standard deviations for all nodes in the chain as depicted in figure 7.5 (left). The results show that

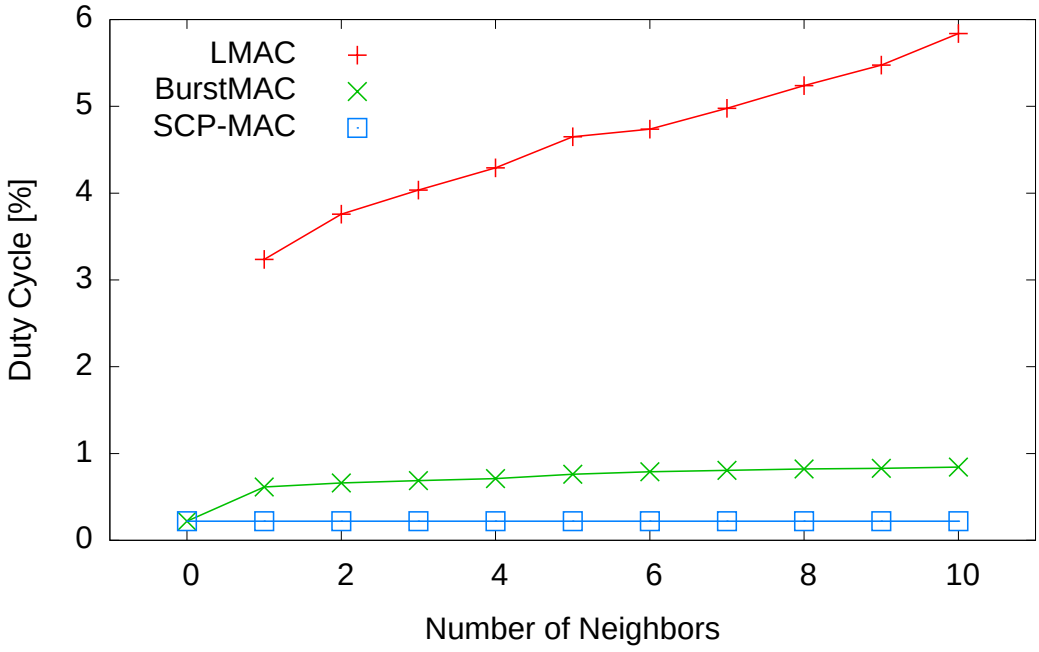


Figure 7.6.: Radio duty cycle in idle mode as a function of neighborhood size.

the average synchronization error increases by about 25 μs per hop. Note, however, that the accuracy of our clock is only 30.5 μs. As the required synchronization accuracy is in the order of few milliseconds (we only need to synchronize to rounds, not at the byte or bit level), we can easily support networks with a diameter in the order of several tens of hops.

7.4.3. Idle Case

We investigate the idle overhead of the protocols in terms of the radio duty cycle. For both BurstMAC and LMAC, the radio duty cycle is a function of the number of neighbors of a node as a node has to receive the control message from each of its neighbors. Therefore, we study the radio duty cycle of a node with a varying number of neighbors. For each neighborhood size, we ran the network for 5 minutes, measuring the time the radio was on and compute the average duty cycle as depicted in figure 7.6 (right). A neighborhood size of zero represents a special case. In this situation, BurstMAC behaves as described in section 7.2.6 and switches to low-power listening. As the behavior or LMAC is not specified in the technical report [82], we have to omit this data point. For more than one neighbor, we find that the duty cycle increases by about 0.02 % per added neighbor for BurstMAC and by about 0.5% for LMAC. The high duty cycle for LMAC is a consequence of its short frame rate which is required to deliver burst traffic. The duty cycle for SCP-MAC, without

time synchronization or neighborhood discovery packets, stays constantly at 0.22%, which is the cost of one clear channel assessment per second.

When comparing idle overhead to other MAC protocols, we need to take into account that the above numbers for BurstMAC already contain the overhead for maintaining the routing topology and allow for the collision-free sending of short broadcast messages without extra overhead.

7.4.4. Constant Traffic

In most sensor network applications, data collected in the network is extracted by a small number of sink nodes using convergecast. Therefore, each sink and its direct neighbors form a star topology in which all neighbors of the sink compete for the right to send to the sink. Similarly, the inner nodes in a data gathering tree form such a star topology. Hence, the performance of a star network, where N senders try to send to one receiver simultaneously, represents the performance bottleneck of tree collection protocols during correlated traffic bursts. Therefore, we first evaluate the performance of the protocols on such a star topology with $N = 7$ senders (a typical number to ensure connected networks) under different data rates.

In addition to the full version of BurstMAC, we also study BurstMAC's performance when the packet burst feature is disabled, termed BurstMAC NoBurst, to provide a better comparison to the other protocols. Each node creates packets with a constant data rate. Every 32 (resp. 38.4 for LMAC) seconds, the number of packets received without bit-errors and the total radio-on time are logged, and the packet rate is increased. We plot the average duty cycle per node and the radio time per packet in figure 7.7. Note that each protocol can handle a certain maximal packet rate depending on its design. To determine the maximal usable packet rate and the corresponding efficiency, we let the nodes send packets as fast as possible. Table 7.1 lists the results.

Although LMAC's idle duty cycle is higher than for other protocols, its collision and contention-free design allows to deliver packets more efficiently than SCP-MAC for packet rates higher than 6 packets/second. At its maximal packet rate of about 10 packets per second, its energy efficiency is even slightly better than BurstMAC without the packet burst mode. This results from the fact that LMAC only sends a single acknowledgement bit for up to 7 packets, whereas SCP-MAC and BurstMAC

	BurstMAC	BurstMAC (NoBurst)	SCP-MAC	LMAC
Packet Rate [1/s]	35.3	18.4	17.6	9.5
Time per Packet [ms]	43.07	71.1	94.9	79.8

Table 7.1.: Packet rate and radio time per packet for star topology with seven sender nodes.

NoBurst both send complete acknowledgement packets. Overall, BurstMAC with its efficient burst mode has the lowest energy consumption and also delivers packets faster than the other protocols.

Below we study the performance of the protocols during correlated traffic bursts in a more realistic multi-hop network. As LMAC outperforms SCP-MAC already in the simple star topology for high data rates, we limit further comparisons to LMAC. Also, increased contention and potential hidden terminal problems in multi-hop networks makes things even worse for SCP-MAC.

7.4.5. Correlated Burst Case

To investigate protocol performance during correlated bursts in a more realistic scenario, we setup a multi-hop network of 30 nodes in our lab. The diameter of the resulting network is four hops and each node has at most ten neighbors. We simulate a traffic burst in the volcano monitoring application [86], where an eruption triggers all nodes simultaneously to transmit a burst of 10 KB (i.e., 320 BurstMAC packets). We measure the following metrics: firstly, the time it takes until all data has been successfully delivered to the sink, and secondly, the total radio energy spent in the network for delivering the correlated burst.

During preliminary tests, we realized that LMAC does not provide a flow control mechanism. In a burst traffic scenario with a high packet load, this would result in significant packet loss due to buffer overruns. Hence, we added a minimal flow control mechanism to LMAC by having a node announce whether or not it is ready to receive data in the control packet. A node is ready to receive when less than the maximum number of packets per LMAC frame is in the outgoing message queue. This simple stop-and-go flow control mechanism effectively avoids dropped packets due to buffer overruns. To evaluate the effect of our flow-control implementation for LMAC, we performed the experiment with BurstMAC, plain LMAC, and LMAC with flow control.

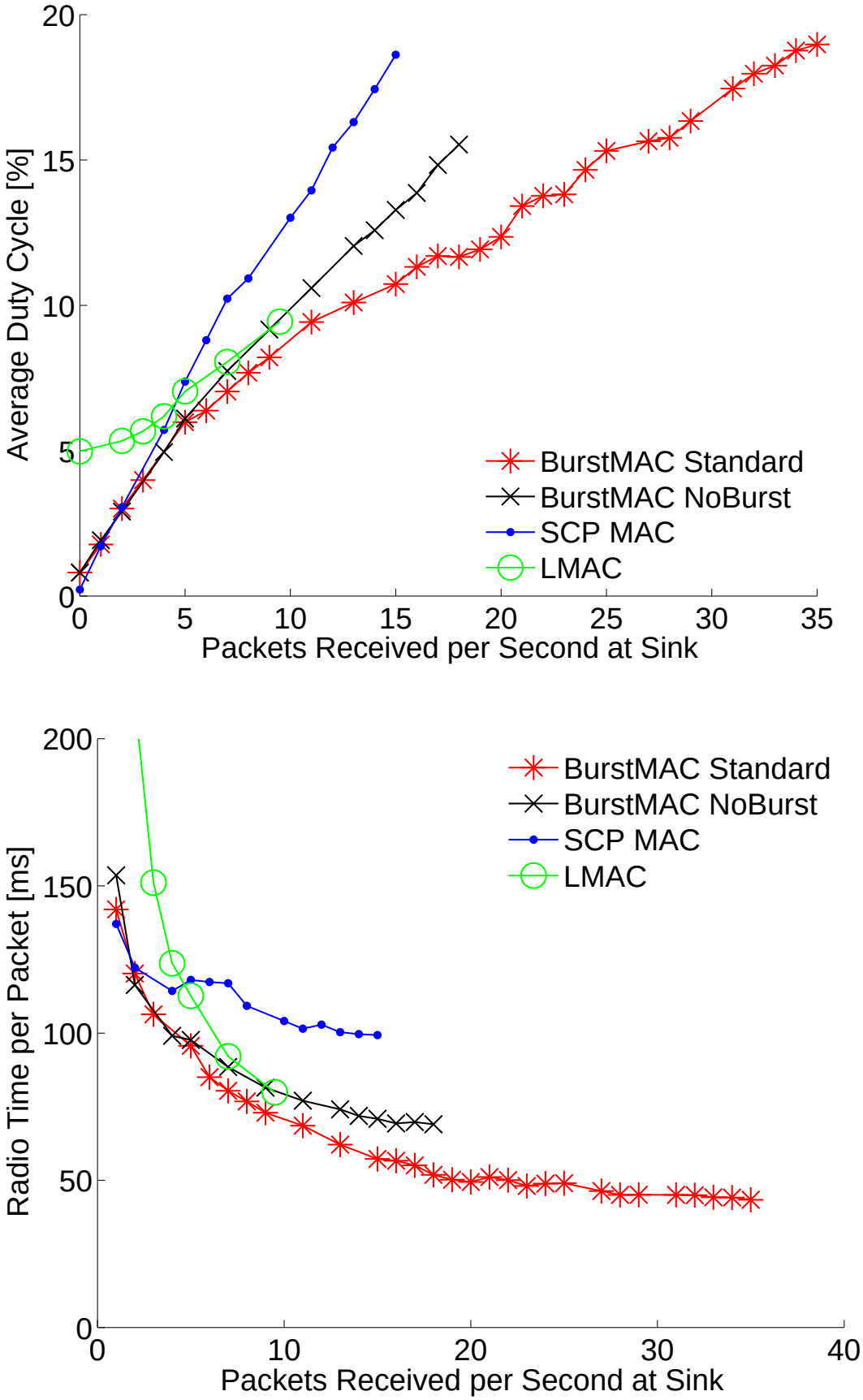


Figure 7.7.: Star topology with one sink and seven neighbor nodes: average duty cycle (top), radio time per error-free received packet (bottom). Note that each protocol can only support constant traffic up to a certain packet rate.

To compare the energy overhead of correlated bursts, we measure the radio-on time of each node from the start of the experiment until the node has finished sending all data and goes back to idle mode. By this, LMAC's comparatively high idle duty cycle does not affect the burst energy measurement. The results of the time measurement are shown in figure 7.8.

For LMAC without flow control, we measured a packet retransmission rate of about 120% which means that each packet had to be sent more than twice. With our flow-control implementation, the packet retransmission rate drops to about 6%. Although BurstMAC also provides efficient flow control, we measured a packet retransmissions rate of about 12%. We attribute these packet losses to the fact that a lost burst acknowledgment (due to bit errors) currently requires to resend the whole burst although a significant number of packets might have been received correctly.

BurstMAC delivers the 9280 packets in 448 seconds which is about 5 times faster than both LMAC variants. The effectiveness of BurstMAC's multiple radio channels is limited as all traffic has to flow to the sink through a single channel. Hence, convergecast with a single sink can be considered as the worst case for BurstMAC with respect to LMAC. In scenarios with multiple (also collocated) sinks or point-to-point multi-hop traffic flows, BurstMAC's gain over LMAC will be even higher. In the convergecast scenario, the main benefit of BurstMAC's multi-channel approach is the reduction of collisions and energy consumption, as we show next.

To estimate the protocol overhead introduced by the MAC protocols, we measured for all protocols the total radio-on time based on the collected metrics. In addition, we calculated the minimal radio-on time based on a packet length of 35 bytes (1 byte type, 32 bytes payload and 2 byte CRC) to be 14.5 ms at 19200 kbps. As the transmission of a packet involves both the sender and the receiver, we define the minimal radio on-time as 29 ms. Figure 7.9 depicts the average radio time per packet per protocol (bars) together with the minimal radio-on time (horizontal line). BurstMAC required 43.24 ms which is at the same level as during the constant traffic experiment and shows that BurstMAC can achieve its energy efficiency also in multi-hop data gathering applications without extra configuration. Note that the complete overhead is only about 50% higher than the minimum and that this already includes the control messages and the overhead caused by sending preambles and acknowledgments. LMAC with flow control requires more than three times the minimal radio-on time. LMAC without flow control requires even more energy but not as

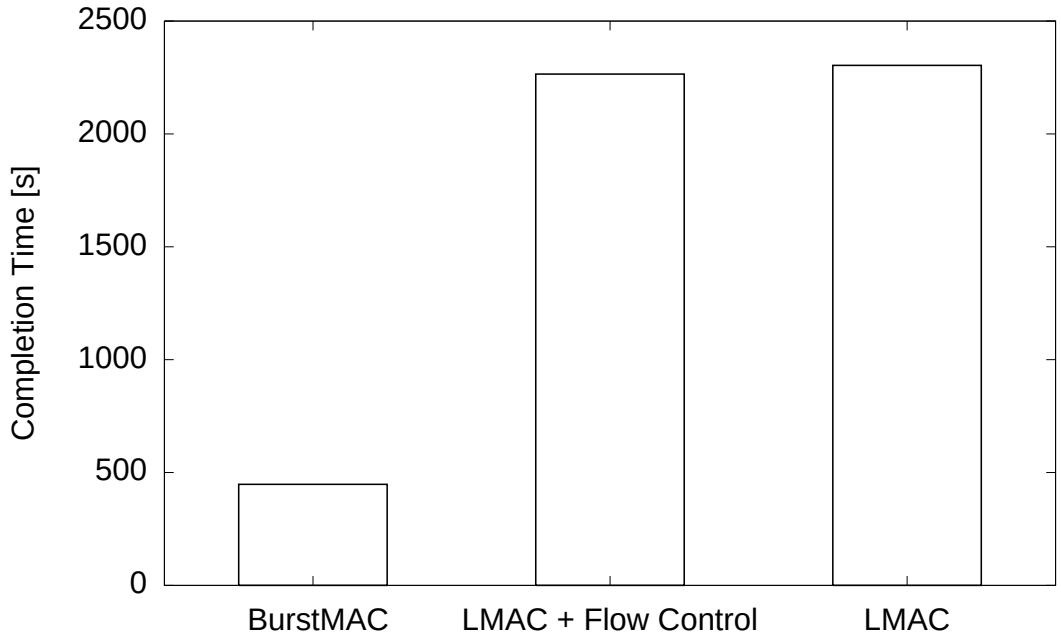


Figure 7.8.: Burst experiment: time to burst completion.

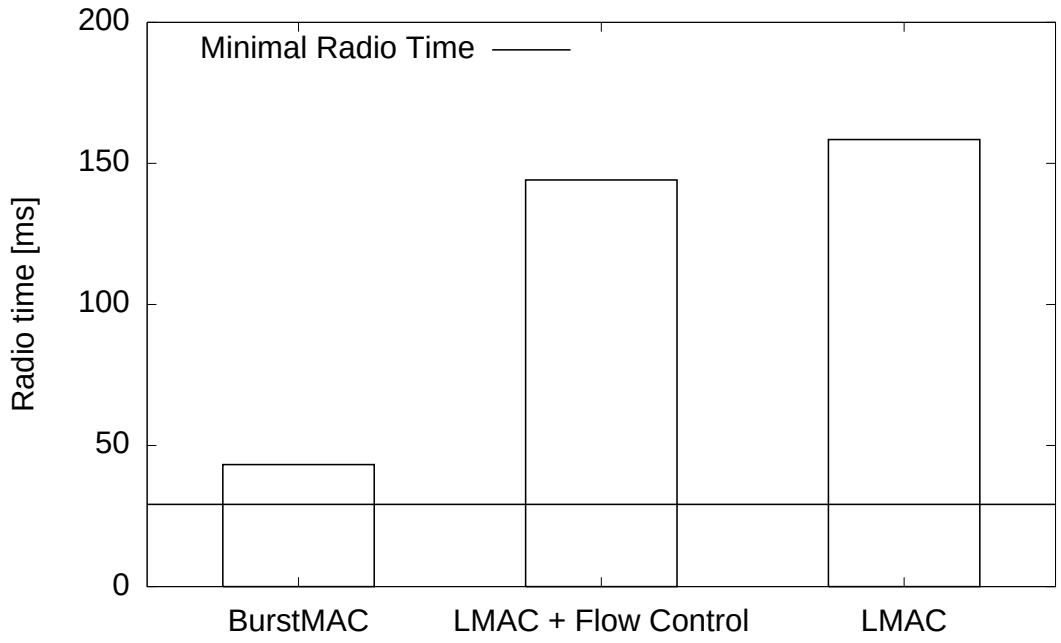


Figure 7.9.: Burst experiment: average and minimal radio time per packet.

much as we expected due to dropped packets caused by overruns, hence, the flow control did not significantly improve the delivery time, but it does reduce LMAC's energy consumption and should be added if LMAC is used in the future.

7.5. Summary

We have presented and analyzed BurstMAC, a collision-free and efficient protocol for event-triggered applications with correlated traffic bursts. BurstMAC shares some design concepts with its predecessor BitMAC, namely the use of cooperative and single-bit transmission for efficient communication in star topologies, and the use of multiple radio channels to avoid interference between neighboring star topologies. In contrast to BitMAC, BurstMAC can handle topology changes and is not restricted to the spanning-tree convergecast model, which allows for unicast communication between neighboring nodes. The new packet based topology maintenance significantly reduces the required time synchronization, and, together with the efficient network startup, yields a protocol ready for real-world deployments.

For the BTnode Rev. 3 platform, we could show that BurstMAC has a similar idle overhead as state-of-the-art low-power data collection MAC protocols under comparable conditions and that BurstMAC significantly outperforms existing scheduled protocols with respect to efficient handling of correlated traffic bursts.

Part III.

Conclusion

8. Conclusion and Future Work

In this final chapter, we summarize the contributions of our work and discuss some limitations of our approaches. We also sketch potential future research directions based on the proposed concepts and give recommendations for a sensor network protocol design that facilitates passive inspection.

8.1. Contributions

We surveyed the literature on the deployment of wireless sensor networks and could show that deployments are still often prone to failure. Apart from hardware-related problems, software failures exist even after the deployed sensor applications have been carefully tested in the lab. We proposed a classification for the reported deployment failures based on the number of involved nodes: single node problems, link problems, path problems and global problems. Most of the reported problems fall into the categories of link and path problems, but are hard to debug as sensor networks do not provide insight into the state of sensor nodes from remote. An important problem is the occurrence of bursty traffic patterns which cannot be handled by existing medium access control protocols.

With respect to the inspection of sensor networks, we showed that existing tools and approaches to provide insight into the network interfere significantly with the inspected application and require too much of the resources of sensor nodes. We proposed the concept of passive inspection that is based on overhearing and analyzing the network traffic of a deployed sensor network without instrumentation of the sensor nodes. We provided the Sensor Network Inspection Framework (SNIF) to implement this concept in a generic way. SNIF makes use of a robust and reliable deployment-support network and allows to inspect sensor networks that use a variety of operating systems and networking protocols. A data stream processing engine allows to adapt SNIF to different applications and to provide input for a graphical user interface. The configurable user interface both allows to inspect the live state of the network and to replay

and analyze recorded network traces.

To better support event-triggered applications with sporadic traffic bursts, we developed two collision-free medium access control protocols. Both protocols make use of multiple radio channels to dynamically cluster nodes into star topologies that can communicate without interference with other clusters. Within such a cluster, cooperative transmissions are used for the efficient scheduling of TDMA slots. In the first protocol, BitMAC, the network start-up and the frequency assignment is deterministic and we gave upper bounds on the execution time of all protocol elements. However, some parts of the protocol do not cope well with changes in the network topology. In the second protocol, BurstMAC, we used a less efficient approach for the frequency assignment, but this led to a protocol with less stringent timing constraints and which can deal with topology changes. We finally showed that BurstMAC can handle traffic bursts well while still providing an idle duty cycle below 1% with a frame length of 1 second.

8.2. Limitations and Future Work

There are a number of limitations and potential improvements with respect to our passive inspection approach and the proposed MAC protocols, which we will discuss in this section. We also discuss future work in the broader context of our work.

8.2.1. Passive Inspection

Limited Insight

The passive inspection approach we proposed solves the problem of inspecting an already deployed sensor network without active instrumentation. Naturally, this passive approach also has limitations. By design, the passive inspection approach only makes use of data broadcasted by sensor nodes. Consequently, if no data about the internal state of a sensor node is included, its state cannot be inferred. A special case in the analysis of network protocols is the question whether a packet was received by a particular node. Even if a message is received by the deployment-support network, the passive approach cannot decide whether an observed node has correctly received the message.

Multi-channel MAC Protocols

In addition to the limited insight, the fact that common radio transceivers can only receive on a single channel at the same time limits the use of SNIF for multi-channel protocols. In the case of BurstMAC, e.g., observing the common channel provides valuable information, especially as most link layer and routing information is broadcasted there. However, the actual data exchange cannot be captured. Optionally, at least the data traffic from and to a single node could be captured if the sniffer would follow the frequency changes according to the BurstMAC protocol. Providing a way to specify the channel changes based on the information encapsulated in overheard control messages would therefore be a natural extension to the current SNIF implementation.

Semi-passive Inspection

As we have shown in sections 3.3.2 and 4.9, for many common problems which occur during deployment of sensor networks, passive indicators exist that allow to infer the existence of a problem from overheard network traffic. However, in some situations we had to make assumptions about the underlying sensor network protocols that may not hold for all applications (e.g., indicators for message loss). For other problems, passive indicators provide only an approximate view of the ground truth (e.g., indicators for network congestion). In such situations, one could resort to *semi-passive observation*, where sensor nodes are instrumented with code to emit messages containing additional information about the state of the sensor node (e.g., level of congestion, battery voltage, etc.). These messages are overheard by the DSN and ignored by other sensor nodes. While this approach requires instrumentation of sensor nodes and transmission of additional messages, these messages do not have to be routed through the sensor network. Moreover, if the additional data is added already during the development of the sensor network application, the probe effect described in section 3.1.2 is avoided.

Graphical Interface for Operator Graph Configuration

While not actually a scientific problem, the availability of a graphical user interface for the configuration of the operator graph in SNIF would ease SNIF's use and lower the barrier for new users. In the current implementation, the operator graph is constructed by Java code. In a future version,

a configuration file could be used to denote the graph and the parameters of the operators. Finally, a graphical tool could be developed to support the construction and checking of the operator graph visually.

8.2.2. Collision-Free Medium Access

Packet-based Radio Transceiver

Currently, we have only implemented BurstMAC on the BTnode Rev. 3 platform with a Chipcon CC1000 radio transceiver that provides fine-grained control over the radio operation and supports OOK to implement efficient single-bit transmission. Together with the steady progress in hardware technology, the radio transceivers used on typical sensor nodes change over time. In the year 2000, single channel radio transceivers such as the RFM TR1000 were common, which required the microcontroller to process raw bit streams. Later, the Chipcon CC1000 with hardware support for Manchester coding allowed the microcontroller to send and receive at the byte level. Newer receivers, such as the 802.15.4-compatible Chipcon CC2420 implement basic MAC functionality in hardware and only allow to send and receive on the packet level. For these radios, BurstMAC cannot be implemented directly. However, BurstMAC uses its low-level techniques only in two cases: the cooperative signaling that at least one sender is ready and single-bit transmission to communicate the actual send requests. Similar to the the emulation of OOK on the CC1000, sending and receiving of OOK data may be implemented indirectly. The CC2420 for example provides a test mode to transmit an unmodulated carrier signal that can be used to send short jamming signals controlled by the microcontroller. Furthermore, it also provides hardware support for clear-channel assessment that can be used as an indicator for other nodes jamming signals. With these two methods, it should be possible to implement BurstMAC on this hardware. Although this example motivates the possibility of adapting BurstMAC to newer hardware, its concrete implementation has to be evaluated for each radio transceiver separately.

Extension for Dense Networks

Finally, the number of available radio channels currently limits the maximal density of sensor nodes. For random network topologies, such as the ones used in the BitMAC coloring evaluation in section 6.6.2, the 32+2 channels used in BurstMAC allow for an average degree of about 8 nodes,

which is enough to provide a well-connected network. To support more dense networks, nodes can act as cluster heads for nodes that could not be assigned a free channel. Even without an assigned channel, such a *leaf node* can still receive broadcast messages, like all other nodes, e.g., to receive configuration data or a sensor query. As it does not control a channel, however, it cannot receive unicast messages from other nodes. But, it can still send unicast messages to other nodes on their respective channel. For this to work, the number of single request bits in the DATA section might be increased by $e + 1$ and each a leaf nodes has to be assigned one of these e extra IDs by its new cluster head. While the first additional bit is used to signal the cluster head that a leaf node requests to be accepted, the other e bits are used to enumerate additional leaf nodes. If the first bit is set, the cluster head schedules a data slot in which all unassigned leaf nodes content to send a packet with their node ID. If the cluster head correctly receives the node ID of a leaf node, it will announce in the next control message that node with ID X can use the additional request bit Y. In the case of collisions, the leaf nodes follow an exponential back-off strategy until all leaf nodes are associated with a cluster-head. With this scheme, the number of nodes in a given area can be increased by a factor of $e + 1$.

8.3. Recommendations for Protocol Design

As we have shown in the paragraph on the PacketTracer in section 4.6.5, the omission of the two-byte sender address in Tiny OS messages transforms the tracking of multi-hop messages from a basic task into a hard problem. This example shows that protocol design decisions may have a strong impact on the ease of deployment. Therefore, we would like to end this thesis with a short list of guidelines for the design of new protocols that facilitates passive inspection *by design*.

- Include the per-hop source address in all messages to identify communication partners.
- Include a link layer sequence number in all messages so that the observing network can detect when it failed to overhear a message.
- Include information in multi-hop messages that allows to decide whether or not two packets belong to the same multi-hop message

exchange. As the source address of a multi-hop message is commonly included to specify which node did create a message, it is sufficient, e.g., to include an additional network layer sequence number that is forwarded as part of the message.

- Include information in routing messages that allows to reconstruct the routing graph. For example, a common routing beacon contains its address and the cost for forwarding a packet to the sink, but it does not announce which node is its parent. Adding this mere two bytes of information to the beacon message is enough to reconstruct the complete spanning-tree-based routing graph.

We hope that our work on the inspection of sensor nodes and on collision-free medium protocols helps to reduce the uncertainty in wireless sensor networks and will benefit future deployments.

References

[1] Marcos K. Aguilera, Jeffrey C. Mogul, Jaenet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, New York, USA, October 2003.

[2] Brian Babcock, Shivnath Babu, Mayur Data, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the 21st Symposium on Principles of Database Systems (PODS)*, Madison, Wisconsin, USA, June 2002.

[3] Paul T. Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (ODSI)*, pages 259–272, San Francisco, California, USA, October 2004.

[4] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, pages 43–56, New York, NY, USA, 2008. ACM.

[5] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 332–343, Washington, DC, USA, 2008. IEEE Computer Society.

[6] Richard Beckwith, Dan Teibel, and Pat Bowen. Pervasive computing and proactive agriculture. In *Adjunct Proceedings of PERVASIVE 2004*, Vienna, Austria, April 2004.

[7] Lucia Lo Bello and Orazio Mirabella. Clock synchronization issues in bluetooth-based industrial measurements. In *Proceedings of the*

Workshop on Factory Communication Systems, Torino, Italy, June 2006.

[8] Jan Beutel, Matthias Dyer, Lennard Meier, and Lothar Thiele. Scalable topology control for deployment-sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, Los Angeles, California, USA, April 2005.

[9] Jack E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[10] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000. Expanded version available as USC TR 99-702b at `http://www.isi.edu/~johnh/PAPERS/Bajaj99a.html`.

[11] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 307–320, New York, NY, USA, 2006. ACM.

[12] Randal Burns, Andreas Terzis, and Michael Franklin. Design tools for sensor-based science. In *Proceedings of the 3rd Workshop on Embedded Networked Sensors (EmNets)*, Cambridge, Massachusetts, USA, May 2006.

[13] Jenna Burrell, Tim Brooke, and Richard Beckwith. Vineyard computing: sensor networks in agricultural production. *Pervasive Computing, IEEE*, 3(1):38–45, Jan.-March 2004.

[14] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 450–459, Cambridge, Massachusetts, USA, April 2007.

[15] Michael Cammert, Christoph Heinz, Jürgen Krämer, Alexander Markowetz, and Bernhard Seeger. PIPES: A multi-threaded

publish-subscribe architecture for continuous queries over streaming data sources. Technical report, University of Marburg, Germany, 2003.

[16] Robert Casas, Héctor J. Garcia, Álvaro Marco, and Jorge L. Falco. Synchronization in wireless sensor networks using bluetooth. In *Proceedings of 3rd International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, Hamburg, Germany, May 2005.

[17] Supriyo Chatterjea, Lodewijk F.W. van Hoesel, and Paul J.M. Havinga. AI-LMAC: an adaptive, information-centric and lightweight mac protocol for wireless sensor networks. In *Proceedings of the 2nd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 381–388, Melbourne, Australia, Dec. 2004.

[18] Yu-Chung Cheng, John Bellardo, Péter Benkö, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, September 2006.

[19] Brent N. Chun, Philip Buonadonna, Alvin Auyoung, Chaki Ng, David C. Parkes, Jeffrey Shneidman, Alex C. Snoeren, and Amin Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNets)*, pages 19–28, Sydney, Australia, May 2005.

[20] Chuck Cranor, Theodore Johnson, Oliver Spatcheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 647–651, San Diego, California, USA, 2003.

[21] Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. Deployment support network - a toolkit for the development of wsns. In *Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN)*, pages 195–211, Delft, The Netherlands, 2007.

[22] Amre El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 3418–3423, New York, New York, USA, April 2002.

[23] Amre El-Hoiydi, Jean-Dominique Decotignie, Christian Enz, and Erwan Le Roux. WiseMAC, an ultra low power MAC protocol for the WiseNET wireless sensor network. In *Adjunct Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 302–303, Los Angeles, California, USA, November 2003.

[24] Jeremy Elson and Kay Römer. Wireless sensor networks: A new regime for time synchronization. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(1):149–154, January 2003.

[25] Emre Ertin, Anish Arora, Rajiv Ramnath, Vinayak Naik, Sandip Bapat, Vinod Kulathumani, Mukundan Sridharan, Hongwei Zhang, Hui Cao, and Mikhail Nesterenko. Kansei: a testbed for sensing at scale. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 399–406, Nashville, Tennessee, USA, 2006.

[26] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 138–149, Los Angeles, California, USA, November 2003.

[27] Deepak Ganesan, Bhashkar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report CSD-TR 02-0013, UCLA, Los Angeles, California, USA, 2002.

[28] Lewis Girod, Jeremy Elson., Alberto Cerpa, Thanos Stathapopoulos, Nithiya Ramananthan, and Deborah Estrin. EmStar: A software environment for developing and deploying wireless sensor networks. In *Proceedings of the USENIX Annual Technical Conference*, pages 283–296, Boston, Massachusetts, USA, July 2004.

[29] Lewis Girod, Thanos Stathopoulos, Nithya Ramanathan, Jeremy Elson, Deborah Estrin, Eric Osterweil, and Tom Schoellhammer.

A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 201–213, New York, NY, USA, 2004. ACM.

[30] Richard Guy, Ben Greenstein, John Hicks, Rahul Kapur, Nithya Ramanathan, Tom Schoellhammer, Thanos Stathopoulos, Karen Weeks, Kevin Chang, Lew Girod, and Deborah Estrin. Experiences with the extensible sensing system ESS. Technical Report 61, CENS, UCLA, Los Angeles, California, USA, January 2006.

[31] Vlado Handziski, Andreas Köpke, Andreas Willig, and Adam Wolisz. Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN)*, pages 63–70, New York, NY, USA, 2006. ACM.

[32] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless sensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, pages 8020–8029, Hawaii, USA, January 2000.

[33] Tristan Henderson and David Kotz. Measuring wireless LANs. In Rajeev Shorey, A. Ananda, Mun Choon Chan, and Wei Tsang Ooi, editors, *Mobile, Wireless, and Sensor Networks*, pages 5–29. Wiley, 2006.

[34] An-swol Hu and Sergio D. Servetto. Asymptotically optimal time synchronization in dense sensor networks. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Application (WSNA)*, pages 1–10, San Diego, USA, September 2003.

[35] Lifei Huang and Ten-Hwang Lai. On the scalability of IEEE 802.11 ad hoc networks. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 173 – 182, Lausanne, Switzerland, June 2002.

[36] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication

paradigm for sensor networks. In *Proceedings of the 6th International Conference on Mobile Computing and Networking (MobiCom)*, pages 56–67, Boston, Massachusetts, USA, August 2000.

[37] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[38] Jason Redi and Steve Kolek and Keith Manning and Craig Partridge and Regina Rosales-Hain and Ram Ramanathan and Isidro Castineyra. Javelen – an ultra-low energy ad hoc wireless network. *Ad Hoc Networks*, 6(1):108–126, 1 2008.

[39] Öjvind Johannson. Simple distributed $\delta + 1$ coloring of graphs. *Information Processing Letters*, 70:229–232, 1999.

[40] Cornelia Kappler and Georg Riegel. A real-world, simple wireless sensor network for monitoring electrical energy consumption. In *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN)*, pages 339–352, Berlin, Germany, January 2004.

[41] Mohammad Maifi Hasan Khan, Liqian Luo, Chengdu Huang, and Tarek Abdelzaher. SNTS: Sensor network troubleshooting suite. In *Proceedings of the 3rd International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 142–157, Santa Fe, New Mexico, USA, June 2007.

[42] 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 (IPSN)*, pages 53–63, St. Louis, Missouri, USA, 2008.

[43] Albert Krohn, Michael Beigl, Christian Decker, and Till Riedel. Syncob: Collaborative time synchronization in wireless sensor networks. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS)*, pages 283–290, Braunschweig, Germany, June 2007.

[44] Albert Krohn, Michael Beigl, and Sabin Wendhack. SDJS: Efficient statistics in wireless networks. In *Proceedings of the 12th*

IEEE Internatinonal Conference on Network Protocols (ICNP), pages 262– 270, Berlin, Germany, October 2004.

[45] Koen Langendoen. The MAC Alphabet Soup served in Wireless Sensor Networks. `http://http://www.st.ewi.tudelft.nl/~koen/MACsoup/`.

[46] Koen Langendoen. Medium access control in wireless sensor networks. In Hongyi Wu and Yi Pan, editors, *Medium access control in wireless networks, volume II: practive and standards*, pages 535–560. Nova Science Publishers, 2008.

[47] Koen Langendoen, Aline Baggio, and Otto Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proceeding of the 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Rhodes, Greece, April 2006.

[48] Hieu Khac Le, Dan Henriksson, and Tarek Abdelzaher. A practical multi-channel media access control protocol for wireless sensor networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 70–81, St. Louis, Missouri, USA, 2008.

[49] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 126–137, Los Angeles, California, USA, November 2003.

[50] Jing Li and Georgios Y. Lazarou. A bit-map-assisted energy-efficient MAC scheme for wireless sensor networks. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 55–60, Berkeley, California, USA, April 2004.

[51] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 131 – 146, Boston, USA, December 2002.

[52] Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan. Analyzing the mac-level behavior of wireless networks. In *Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 75–86, Pisa, Italy, September 2006.

[53] Alan Mainwaring, David Culler, Joe Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88 – 97, Atlanta, Georgia, USA, September 2002.

[54] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39 – 49, Baltimore, Maryland, USA, November 2004.

[55] Felix Michel and Philippe Wüger. Angewandte Uhrensynchronisation auf BTnodes. Term Project, D-ITET, ETH Zurich, Switzerland, 2005.

[56] Stehpan Olariu, Ashraf Wadaa, Larry Wilson, and Mohamed Eltoweissy. Wireless sensor networks - leveraging the virtual infrastructure. *IEEE Network*, pages 51–56, July 2004.

[57] Paritosh Padhy, Kirk Martinez, Alistair Riddoch, H. L. Royan Ong, and Jane K. Hart. Glacial environment monitoring using sensor networks. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN)*, Stockholm, Sweden, 2005.

[58] Jeongyeup Paek, Krishna Chintalapudi, Ramesh Govindan, John Caffrey, and Sami Masri. A wireless sensor network for structural health monitoring: Performance and experience. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNets)*, pages 1–10, Sydney, Australia, May 2005.

[59] Mathias Payer. Implementation of a Bluetooth stack for BTnodes and Nut/OS. `http://www.btnode.ethz.ch/static_docs/btnut/bt-stack.pdf`, 2004.

[60] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the*

2nd International Conference on Embedded Networked Sensor Systems (SenSys), pages 95–107, Baltimore, Maryland, USA, November 2004.

[61] Joseph Polastre Polastre, Rob Szewczyk, Alan Mainwaring, David Culler, and John Anderson. Analysis of wireless sensor networks for habitat monitoring. In Cauligi S. Raghavendra, Krishna M. Sivalingam, and Taieb Znati, editors, *Wireless Sensor Networks*, chapter 18. Kluwer Academic Publishers, 2004.

[62] Venkatesh Rajendran, Katia Obraczka, and J.J. Garcia-Luna-Aceves. Energy-efficient, collison-free medium access control for wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 181–192, Los Angeles, USA, November 2003.

[63] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 255 – 267, Los Angeles, California, USA, November 2005.

[64] Nithya Ramanathan, Eddie Kohler, and Deborah Estrin. Towards a debugging systems for sensor networks. *International Journal of Network Management*, 15(4):223–234, July 2005.

[65] Olof Rensfelt, Richard Gold, and Lars-Ake Larzon. LUNAR over Bluetooth. In *Proceedings of the 4th Scandinavian Workshop on Wireless Ad-hoc Networks (ADHOC)*, Stockholm, Sweden, May 2004.

[66] Ruud Riem-Vis. Cold chain management using an ultra low power wireless sensor network. In *Proceedings of the Workshop on Applications of Mobile Embedded Systems (WAMES)*, Boston, Massachusetts, USA, June 2004.

[67] Matthias Ringwald, Marc Cortesi, Kay Römer, and Andrea Vitaletti. Vialetti. Demo abstract: Passive inspection of deployed sensor networks with snif. In *Adjunct Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN)*, pages 45–46, Delft, The Netherlands, January 2007.

[68] Kay Römer, Philipp Blum, and Lennart Meier. Time synchronization and calibration in wireless sensor networks. In Ivan Stojmenovic, editor, *Handbook of Sensor Networks: Algorithms and Architectures*. John Wiley & Sons, September 2005.

[69] Bor rong Chen, Geoffrey Peterson, Geoff Mainland, and Matt Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. In *Proceedings of the 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2008)*, pages 79–98, Santorini Island, Greece, June 2008.

[70] Stanislav Rost and Hari Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON)*, pages 577–584, Reston, Virginia, USA, September 2006.

[71] Ahmed Sobeih, Mahesh Viswanathan, Darko Marinov, and Jennifer C. Hou. Finding bugs in network protocols using simulation code and protocol-specific heuristics. In *Proceedings of the 7th International Conference on Formal Engineering Methods (ICFEM)*, pages 235–250, Manchester, United Kingdom, 2005. Springer.

[72] Mark Sullivan and Andrew Heybey. Tribeca: A system for managing large databases of network traffic. In *Proceedings of the USENIX Annual Technical Conference*, pages 13–24, New Orleans, Louisiana, USA, June 1998.

[73] Sameer Sundresh, Wim Kim, and Gul Agha. SENS: A sensor, environment and network simulator. In *Proceedings of the 37th Annual Simulation Symposium (ANSS)*, pages 221–228, Arlington, Virginia, USA, April 2004.

[74] Robert Szewcyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 214 – 226, Baltimore, Maryland, USA, November 2004.

[75] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a sensor network expedition. In *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN)*, pages 307–322, Berlin, Germany, January 2004.

[76] Jane Tateson, Chrisopher Roadknight, Antonio Gonzalez, Steve Fitz, Nathan Boyd, Chris Vincent, and Ian Marshall. Real world issues in deploying a wireless sensor network for oceanography. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN)*, Stockholm, Sweden, June 2005.

[77] Sameer Tilak, Nael B. Abu-Ghazaleh, and Wendi Rabiner Heinzelman. A taxonomy of wireless micro-sensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, 6(2):28–36, April 2002.

[78] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 477– 482, Los Angeles, California, USA, April 2005.

[79] Gilman Tolle and David Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN)*, pages 121– 132, Istanbul, Turkey, January 2005.

[80] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 51–63, San Diego, California, USA, November 2005.

[81] Lodewijk van Hoesel and Paul Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *Proceedings of the 1st International Workshop on Networked Sensing Systems (INSS)*, pages 205–208, Tokyo, Japan, June 2004.

[82] Lodewijk van Hoesel and Paul Havinga. Design aspects of an energy-efficient, lightweight medium access control protocol for wireless sensor networks. Technical Report TR-CTIT-06-47, University of Twente, Enschede, July 2006.

[83] András Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM)*, pages 319–324, Prague, Czech Republic, June 2001.

[84] Roger Wattenhofer and Aaron Zollinger. XTC: a practical topology control algorithm for ad-hoc networks. *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 216–223, April 2004.

[85] Ye Wen, Wei Zhang, Rich Wolski, and Navraj Chohan. Simulation-based augmented reality for sensor network development. In *Proceedings of the 5th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, pages 275–288, New York, New York, USA, 2007.

[86] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 381 – 396, Seattle, Washington , USA, 2006.

[87] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Motelab: a wireless sensor network testbed. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 483–488, Los Angeles, California, USA, 2005.

[88] Kamin Whitehouse, Alec Woo, Fred Jiang, Joseph Polastre, and David Culler. Exploiting the capture effect for collision detection and recovery. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNets)*, pages 45– 52, Sydney, Australia, 2005.

[89] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 14–27, Los Angeles, California, USA, 2003.

[90] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the*

21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), volume 3, pages 1567–1576, New York, New York, USA, June 2002.

[91] Wei Ye, Fabio Silva, and John Heidemann. Ultra-low duty cycle MAC with scheduled channel polling. In *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys)*, pages 321–334, Boulder, Colorado, USA, 2006.

[92] Atmel Corporation. Atmel AVR 8-Bit RISC. `http://atmel.com/products/avr/default.asp`, 2008.

[93] Bluetooth SIG. `http://www.bluetooth.org`.

[94] BlueZ official linux bluetooth stack. `http://www.bluez.org`.

[95] BTnodes. a distributed environment for prototyping ad hoc networks. `http://www.btnode.ethz.ch`.

[96] Chipcon Application Note AN016: CC1000/CC1050 used with On-Off Keying. `http://focus.ti.com/lit/an/swra075/swra075.pdf`.

[97] Crossbow Technology. MICA2 Motes. `http://www.xbow.com/Products/productdetails.aspx?sid=174`.

[98] egnite GmbH. embedded ethernet. `http://www.ethernut.de`.

[99] EYES: Energy Efficient Sensor Networks. `http://eyes.eu.org`.

[100] Moteiv corporation. `http://www.motiev.com`.

[101] Multimodal Networks of In-situ Sensors (MANTIS). `http://mantis.cs.colorado.edu`.

[102] Scatterweb Embedded Sensor Board. `http://www.scatterweb.net`.

[103] Smart-Its Particle Computer. `http://particle.teco.edu`.

[104] SOWNet Techologies. T-Nodes. `http://www.sownet.nl`.

[105] TinyDB. http://telegraph.cs.berkeley.edu/tinydb.

[106] TinyOS. http://www.tinyos.net.

[107] LAN MAN standards committee of the IEEE Computer Society 802.11 - Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999.

[108] Texas Instruments Incorporated. Chipcon CC1000 single chip ultra low power RF tranceiver for 315/433/868/915 MHz SRD band. http://focus.ti.com/docs/prod/folders/print/cc1000.html, 2008.

Curriculum Vitae

Matthias Ringwald

Personal Data

Data of Birth	November 24, 1973
Birthplace	Böblingen, Germany
Citizenship	Germany

Education

2002-2009	Ph.D. Student, Departement of Computer Science, ETH Zürich
November 2001	Diplom Informatiker
2001	Internship, Georgia Institut of Technoloy, GA, USA
1998	Internship, Institute for Visual Media at the Center for Art and Media (ZKM), Karlsruhe
1996-2001	Hauptstudium, University of Karlsruhe
1994-1996	Grundstudium, University of Karlsruhe
May 1993	Abitur
1983-1993	Max-Plack-Gymnasium Böblingen
1980-1983	Johann-Brücker-Grundschule Schönaich

Civil Service

1993-1994	Kindergarten für schwerst-mehrfach behinderte Kinder, Sindelfingen

Employment

2002-2008	Research assistant, ETH Zurich
2001-2002	Freelance Software Development for 4as grafix, Magstadt
2000-2001	Graduade Assistent, Telecooperation Office (TeCO) Karlsruhe
1998-2001	Spare time-job, thetagroup, Karlsruhe