Diss. ETH No. XXXXX

End-to-end Predictability and Efficiency in Low-power Wireless Networks

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
MARCO ZIMMERLING
Diploma in Computer Science,
Dresden University of Technology
born August 14, 1982
citizen of Germany

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Tarek Abdelzaher, co-examiner

2015

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. XYZ

Marco Zimmerling

End-to-end Predictability and Efficiency in Low-power Wireless Networks

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A dissertation submitted to
ETH Zurich
for the degree of Doctor of Sciences

Diss. ETH No. XXXXX

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Tarek Abdelzaher, co-examiner
Examination date: July 10, 2015

ISBN 978-1492251033
DOI 10.3929/ethz-a-010000257

Abstract

The confluence of networked embedded computing, low-power wireless communications, and sensor technology has spawned a whole spectrum of powerful applications that are commonly believed to radically change the way we perceive and interact with the physical world. Data collection applications, for example, enable the monitoring of physical phenomena with unprecedented spatial and temporal resolutions, and cyber-physical systems (CPS) applications can control physical processes by integrating sensing and computation with actuation into distributed feedback loops. Application domains include transportation, healthcare, and buildings.

Data collection and CPS applications alike demand predictability and efficiency from the wireless communication substrate to function correctly and effectively. In particular, these applications require a certain energy efficiency, reliability, and timeliness of end-to-end packet transmissions. Meeting perhaps multiple such non-functional requirements is, however, extremely challenging. This is due to, for example, the need for multi-hop communication over lossy low-power wireless channels, unpredictable and non-deterministic changes in the environment, and limited resources of the employed devices in terms of computation, memory, and energy.

Dedicated solutions have been proposed that attempt to tackle these challenges in order to satisfy the needs of either non-critical data collection or critical CPS applications. As for the former, adapting the operational parameters of the MAC protocol proved to be highly effective; however, current efforts focus only on a single performance metric or consider local metrics, whereas applications often exhibit requirements along multiple metrics that are most naturally expressed in global, network-wide terms. As for the latter, state-of-the-art solutions including industry standards do not provide hard end-to-end real-time guarantees because of a localized operation, or can hardly keep up with dynamic changes in the network.

To address these problems, this thesis presents new analytical results as well as real implementations of novel protocols and systems that make use of them. Specifically, we make three main contributions:

- We design pTunes, a framework that meets multiple soft application requirements on network lifetime, end-to-end reliability, and end-to-end latency by adapting the MAC protocol parameters at runtime in response to changes in the network and the traffic load. pTunes exploits a centralized approach that is similar in spirit to a model-predictive controller. Results from testbed experiments show that relative to carefully chosen fixed MAC parameters pTunes extends network lifetime by up to 3×, and reduces packet loss by 70–80 % during periods of wireless interference or when multiple nodes fail.

- A new breed of protocols that utilize synchronous transmissions has been shown to enhance the reliability and efficiency of protocols that use link-based transmissions. We find that these emerging protocols also enable simpler and more accurate models, which play a key role in system design, verification, and runtime adaptation to meet given requirements. We show through testbed experiments and statistical analyses that unlike link-based transmissions, packet receptions and losses using synchronous transmissions with Glossy can largely be considered statistically independent events. This property greatly simplifies the accurate modeling of protocols based on synchronous transmissions. We demonstrate this by obtaining an unprecedented error below 0.25 % in the energy model of the Glossy-based Low-power Wireless Bus (LWB), and providing sufficient conditions for probabilistic guarantees on LWB's end-to-end reliability.

- We present Blink, the first protocol that provides hard guarantees on end-to-end packet deadlines in large multi-hop low-power wireless networks. Built on top of LWB as communication support, we map the scheduling problem in Blink to uniprocessor scheduling. We devise earliest deadline first (EDF) based scheduling policies that Blink employs to compute online a schedule that provably meets all deadlines of packets released by admitted real-time packet streams while minimizing the network-wide energy consumption within the limits of LWB, tolerating changes in the network and the set of streams. An efficient priority queue data structure and algorithms we design prove instrumental for a viable implementation of these policies on resource-constrained nodes. Our experiments show that Blink meets nearly 100 % of packet deadlines on a large multi-hop testbed, and achieves speed-ups of up to 4.1× over a conventional scheduler implementation on state-of-the-art microcontrollers.

Contents

Abstract **i**

List of Figures **v**

List of Tables **vii**

1 Introduction **1**
- 1.1 Application Requirements 2
- 1.2 Challenges to Meeting Requirements 4
- 1.3 State of the Art 6
- 1.4 Thesis Contributions and Road Map 9

2 pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols **13**
- 2.1 Optimization Problem 16
- 2.2 Modeling Framework 17
- 2.3 System Support 29
- 2.4 Implementation Details 32
- 2.5 Experimental Results 33
- 2.6 Discussion 44
- 2.7 Related Work 45
- 2.8 Summary 47

3 Modeling Protocols Based on Synchronous Transmissions **49**
- 3.1 Background and Related Work 51
- 3.2 Bernoulli Assumption 53
- 3.3 Low-power Wireless Bus 59
- 3.4 End-to-end Reliability in LWB 60
- 3.5 Energy Consumption in LWB 63
- 3.6 Validation 68
- 3.7 Summary 73

4 Blink: Real-time Communication in Multi-hop Low-power Wireless **75**
- 4.1 Problem Statement 79
- 4.2 Foundation 81
- 4.3 Overview 83
- 4.4 Design and Implementation 85
- 4.5 Evaluation 103
- 4.6 Discussion and Limitations 114
- 4.7 Summary 115

4.A Synchronous Busy Period Computation 115
4.B Supporting Sub-second Deadlines 117

5 Conclusions and Outlook 121
5.1 Contributions 121
5.2 Possible Future Directions 122

Bibliography 125

List of Publications 139

Curriculum Vitæ 143

List of Figures

1.1 Tmote (also known as Tmote Sky) embedded device 5
1.2 Multi-hop low-power wireless network 6

2.1 Overview of pTunes framework 14
2.2 Layered modeling framework of pTunes 17
2.3 Successful unicast transmission in X-MAC 23
2.4 Successful unicast transmission in LPP 26
2.5 Layout of testbed with 44 nodes used to evaluate pTunes 34
2.6 Impact of pTunes on queue overflows and goodput 36
2.7 Performance of pTunes as traffic load changes 40
2.8 Reliability and X-MAC parameters with pTunes under interference 41
2.9 Reliability and parent switches with pTunes as nodes fail 43

3.1 Link-based transmissions versus synchronous transmissions . . 50
3.2 Example of a weakly stationary and a non-stationary trace . . . 55
3.3 Sample autocorrelation of two packet reception traces 57
3.4 Percentage of weakly stationary traces for which the Bernoulli assumption does not hold 58
3.5 Time-triggered operation of LWB 59
3.6 Slots and activities during a LWB round 60
3.7 FSM modeling the behavior of a LWB node 65
3.8 DTMC corresponding to FSM in Figure 3.7 67
3.9 Stationary distribution of the DTMC in Figure 3.8 68
3.10 Measured and guaranteed end-to-end reliability of LWB 71
3.11 Measured and estimated radio on-time per LWB round 72
3.12 Fraction of time in FSM states when discarding 50 % of schedules 73

4.1 Time-triggered operation and sequence of slots in a LWB round 81
4.2 Illustration of important problems Blink needs to address 84
4.3 Discrete-time model of LWB 85
4.4 Example motivating EDF traversal of the set of stream 88
4.5 Illustration of proof of Theorem 1 89
4.6 Bucket queue implemented as circular array of doubly-linked lists 90
4.7 Illustration of second termination criterion in Algorithm 1 . . . 91
4.8 Example execution comparing **CS**, **GS**, and **LS** policies 93

4.9 Illustration of how **LS** computes start time of next round 94
4.10 Illustration of how far **LS** needs to look into the future 95
4.11 Example of a stream set that is not schedulable 100
4.12 Main steps in Blink's real-time scheduler 103
4.13 Real trace of Blink dynamically scheduling streams 105
4.14 Average radio duty cycle of Blink with **LS**, **GS**, and **CS** 108
4.15 Maximum synchronous busy period against bandwidth demand 110
4.16 Execution time of **LS** scheduler against bandwidth demand . . . 113
4.17 Slots and processing in a complete LWB round in Blink 117
4.18 Illustration of a Glossy flood in a 3-hop network 119
4.19 Round length against network diameter and number of data slots 120

List of Tables

2.1 Terms denoting network state and protocol-dependent quantities 18
2.2 Fixed MAC parameters for X-MAC and LPP used for comparison 35
2.3 Average absolute errors of network-wide performance model . . 36
2.4 Lifetime gains of pTUNES relative to fixed MAC parameters . . . 38

3.1 Number of non-stationary and weakly stationary traces 56
3.2 Meaning and radio on-times of FSM states in Figure 3.7 64

4.1 Operations on set of streams needed for EDF scheduling in Blink 87
4.2 Stream profiles used in the experiment of Section 4.5.3 112
4.3 Constants of CC2420 radio and Glossy implementation on TelosB 119

1 Introduction

Low-power wireless communications has been a key enabling technology for innovative applications over the last 15 years. First and foremost, low-power wireless has been the primary choice for networking embedded low-power sensing devices in distributed *wireless sensor networks*. These networks have given rise to different classes of important applications.

A prominent example is the class of *data collection* applications, where tens to hundreds of sensing devices gather information to *monitor* physical processes. Real data collection applications range from habitat [MCP+02], soil ecology [METS+06], permafrost [BGH+09], microclimate [TPS+05], vital sign [CLBR10] as well as structural health monitoring [CFP+06] to traffic [SMR+12], fire [HHSH06], and wildlife tracking [DEM+10]. Many of these applications exploit the possibility of collecting sensor data with unprecedented spatio-temporal resolution using networked devices that are deeply embedded into the environment around us or even inside our bodies to gain a deeper understanding of certain physical phenomena.

Another representative example is the emerging class of *cyber-physical systems (CPS)* applications. These systems are facilitated by augmenting traditional wireless sensor networks with actuating devices such that sensing, computation, and actuation can jointly work in concert within distributed feedback loops to *control* physical processes. CPS applications include factory and building automation, infrastructure control, precision agriculture, distributed robotics, assisted living, traffic safety, industrial process control, and advanced automotive and avionic systems [Lee08, Sta08]. It is widely anticipated that CPS will be key to solving a number of significant societal challenges in the 21st century [NAE, RLSS10].

Irrespective of the application class, low-power wireless communication is what glues everything together by allowing devices to exchange application and protocol data over short distances at low energy cost.

1.1 Application Requirements

The utility of any real-world low-power wireless application is judged on the grounds of *requirements* that are specified by the user, which could be an environmental scientist or a control engineer. This thesis deals with *non-functional requirements* put on the wireless communication substrate that express the desired energy efficiency, reliability, and timeliness of packet transmissions. Both data collection and CPS applications exhibit requirements along these key dimensions, although to varying degrees.

Metric #1: energy. Two tangible benefits of low-power wireless are higher flexibility and lower costs by avoiding any sort of wiring, and this notably includes power cables. At the same time, the vast majority of applications needs to operate without interruption and possibly unattended for several months or even multiple years [CMP+09, CCD+11]. As an example, for intelligent telemetry of freight railroad trains to be economically viable, railroad cars should not be hauled in just to service the networking infrastructure; rather, replacing or recharging batteries is only plausible during regular maintenance of a railroad car [RC10]. Thus, the network lifetime must be at least as long as the maintenance cycle of a car, which can easily exceed five years [ZDR08]. To achieve this, reducing the energy consumption due to communication is an important requirement, because the radio transceiver is one of the most power-hungry components on a typical low-power wireless platform [Lan08].

Metric #2: reliability. In general, an application would like to see as many packets as possible being delivered from the sources to the destination(s). However, successful delivery of all packets cannot be guaranteed over a channel that is lossy because of fading, interference, and environmental effects [SDTL10]. To account for this, a low-power wireless application should tolerate a reasonable amount of packet loss [RCCD09], yet some applications are inherently more resilient to packet loss than others. For instance, when monitoring relatively slow-changing environmental parameters such as temperature and humidity, meaningful long-term analyses may be possible even with 10–20 % of packet loss. However, in high data rate applications such as acoustic source localization [AYC+07] and structural health monitoring [CFP+06], much smaller packet loss rates can adversely affect the accuracy of the corresponding algorithms [PG07]. Similarly, CPS applications typically require a packet reliability well above 99 % to make well-informed control decisions [SSF+04].

Metric #3: latency. Besides the reliable delivery of packets, it is sometimes equally important that packets be delivered in a timely manner. This may be a *soft requirement* in the sense that ″most“ of the packets should arrive at the destination(s) a given interval after they were generated by the sources (e.g., to provide an up-to-date view of the observed phenomenon to the user), but packets arriving late are nevertheless useful for the application (e.g., for later analyses) and do not have any catastrophic consequences. Specific CPS applications, however, impose *hard requirements* on packet latency, typically in the form of *deadlines*; that is, any packet arriving after its deadline is useless to the application and thus counts as lost. Such *real-time applications* are often found in safety-critical scenarios, for example, when control law computations need to occur at pre-determined times to guarantee the stability of the controlled physical process [ÅGB11].

Data collection and CPS applications put different emphasis on these metrics. In the former, energy efficiency is typically paramount, reliability comes second, and latency plays only a minor role. In the latter, instead, meeting packet deadlines through timely and reliable delivery is typically the most important requirement, which leaves energy efficiency, if at all, a secondary objective. In fact, energy consumption may only be a concern for a subset of the devices in the network, for example, for mobile devices running on batteries, while other devices, such as a static base station or programmable logic controller (PLC), enjoy a steady power supply.

Irrespective of the concrete scenario, there are two general characteristics of application requirements one needs to take into account:

- *Application requirements are specified from a network-wide perspective.* Ultimately, the end user only cares about the performance she gets from the system as a whole. This includes the lifetime of the entire *network* as well as the *end-to-end* latency and the *end-to-end* reliability between packet sources and their destinations. It is therefore most natural and convenient for domain experts and other users alike to specify requirements in terms of these *global, network-wide* metrics. Instead, *local metrics* referring to the performance inside the network, for example, the *per-hop* packet delay between neighboring devices, are only of interest to the system or protocol designer.

- *Application requirements are at odds with each other.* Many applications do exhibit multiple requirements. As an example, in a building fire detection system based on wireless battery-powered sensors, real-time packet delivery is mandated by fire regulations, but a high level of energy efficiency is also required to keep the maintenance costs to a minimum. Meeting multiple requirements simultaneously often involves striking a balance between conflicting goals. To illustrate this, consider a protocol using *radio duty cycling*, whereby the radio transceiver is put into a low-power mode as often and for as long as possible. However, two devices can only talk with each other when they both have their radios on. Thus, while radio duty cycling saves energy, it can increase latency and negatively affect reliability, too.

1.2 Challenges to Meeting Requirements

Meeting the requirements of real-world low-power wireless applications is far from trivial. This is due to the need for multi-hop communication protocols that *constantly adapt* their functioning to *unpredictable* and *non-deterministic* changes in the environment, while operating very close to the resource limits of the employed devices. We next discuss these challenges before moving on to a high-level review of prior attempts to tackling them with the goal of meeting soft and hard application requirements.

Low-power wireless links are volatile. Low-power wireless communications are notoriously unreliable. *Fading* because of multipath propagation or shadowing can reduce the power of the received signal to a point where successful reception is impossible; *interference* from wireless transmitters operating on almost the same frequency at the same time can destroy the information encoded in signals; and *meteorological conditions* such as air temperature and humidity affect the link quality, too [BKM+12, WHR+13]. As a result, low-power wireless links are volatile with coherence times as small as a few hundred milliseconds [SKAL08], thus suffering from unpredictable packet loss that can vary significantly over time [SDTL10].

A common approach to combat packet loss is to possibly transmit a packet more than once. Such a packet *retransmission* is triggered by the sender when no acknowledgment from the receiver has arrived within a predetermined time interval [GFJ+09]. Another possibility is the use of an error-correction code, such as Reed-Solomon, whereby multiple check symbols are added to the packet by the sender to detect a certain number of erroneous symbols at the receiver [LPLT10]. While both retransmissions and coding help improve packet reliability, they increase average packet latency and energy consumption. These two examples already show that parameters of communication protocols, such as the maximum number of retransmissions per packet or the number of check symbols to be added, could be important for meeting the application requirements, yet it may be non-trivial to find the right trade-off between all performance metrics.

Resource-constrained devices. To benefit the most from low costs, high flexibility, and deep embedding in the environment, low-power wireless devices often come in small form factors with severely limited resources.

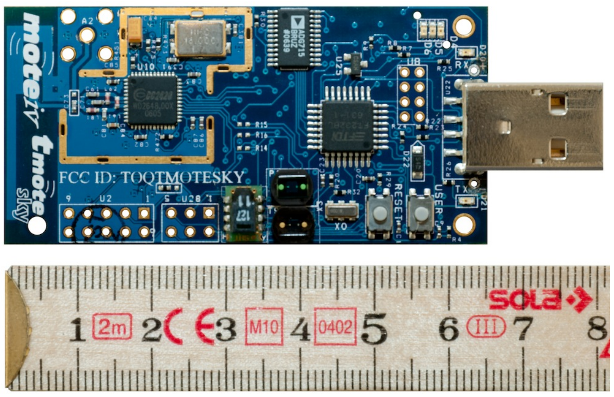


Figure 1.1: A TelosB (also known as Tmote Sky) fits an 8 MHz microcontroller and a 2.4 GHz low-power wireless radio within a few square centimeters.

Typical platforms feature a low-power microcontroller (MCU), a low data rate radio with a relatively short range, and a limited amount of code and data memory. For example, the TelosB [PSC05] shown in Figure 1.1—still one of the most widely used platforms in low-power wireless research today despite its release a decade ago—has a 16-bit MSP430 MCU running at a speed of up to 8 MHz, an IEEE 802.15.4 compliant CC2420 wireless radio operating in the 2.4 GHz ISM band at a fixed data rate of 250 kbps, 10 kB of RAM, 48 kB of program memory, and 1 MB of non-volatile flash storage. In addition, energy is often limited by the battery capacity or the maximum possible intake in an energy harvesting scenario [BSBT14].

These resource constraints put limits on what can be computed, stored, and communicated using low-power wireless platforms. Although there are increasingly more powerful yet very efficient MCUs appearing on the market, including the recent ARM Cortex-M0+, these represent only the middle to upper end of the spectrum. At the lower end, there will soon be true “Smart Dust” chips that integrate computation, communication, storage, and sensing in a cubic-millimeter [LBL+13]. It is clear that these devices will be even more resource-constrained than the smallest devices that challenge the designers of communication protocols today.

Multi-hop communication. One limit that deeply affects communication protocol design is the transmission range of low-power wireless radios of a few tens of meters indoors and a little over a hundred meters outdoors. The locations of nodes in a real deployment are, however, dictated by the application, which may require to cover significantly larger distances. For instance, intelligent telemetry of freight railroad trains requires networks that span the length of a train, which can be up to 2.7 kilometers [ZDR08]; a network for monitoring and controlling a modern paper mill needs to extend across about 150 meters; and process control in chemical plants or

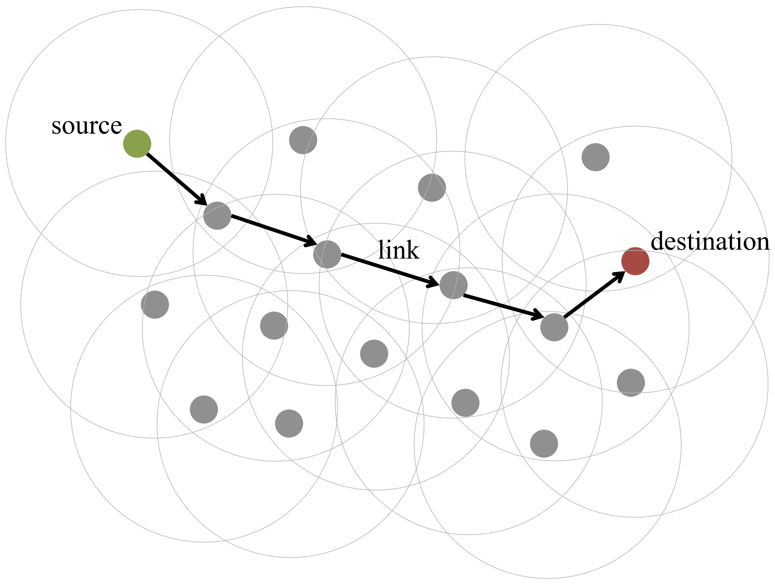


Figure 1.2: Example of a multi-hop low-power wireless network with 16 nodes. Arrows represent physical communication links between neighboring devices; circles represent the nodes' communication ranges. *Due to the limited transmission range of low-power wireless radios, the source relies on intermediate nodes that relay its packets along a routing path to the destination. The quality of each link on the path varies unpredictably over time because of fading, interference, and environmental factors.*

refineries and building automation scenarios require network diameters that are multiples of a node's transmission range. Therefore, as shown in Figure 1.2, end-to-end packet delivery in these networks relies on *multi-hop communication*, where intermediate nodes relay packets on behalf of sources that cannot directly communicate with the intended destinations.

In a multi-hop setting, the amount of *network state*—that is, information about the instantaneous conditions at the physical layer—that determine the success or failure of an end-to-end packet transmission is a function of the number of intermediate hops (or link) connecting the source with the destination. However, as explained before, the quality of low-power wireless links can fluctuate significantly over time even in a static network. Links can also vanish completely when devices suddenly fail because of battery depletion or damage, or when nodes move out of communication range. All these factors concur and make the network state a *continuously changing unknown*, which complicates multi-hop communication [AY05].

1.3 State of the Art

As discussed in the following, three important problems remain unsolved by prior work on low-power wireless communication protocols:

1. Support for non-critical data collection applications with multiple soft requirements on global, network-wide performance metrics.
2. Support for critical CPS applications having hard requirements on end-to-end packet deadlines and possibly also energy constraints.
3. Addressing problems 1. and 2. above in the face of unpredictable and non-deterministic changes in the environment.

Architecture of layered low-power wireless networking stacks. Numerous low-power wireless communication protocols have been developed to tackle the inherent network-level challenges discussed above. Traditional solutions that have been deployed with great success in the real world and also those that are freely available as part of the open source TinyOS and Contiki distributions typically include multiple protocols organized into *layers* [ASSC02]. In low-power wireless, such a *networking stack* often comprises only the three lower layers: physical, data link, and network. The *physical layer* includes the radio hardware and the software driver for transmitting individual bits, often grouped into symbols as defined by the modulation scheme, between two devices within communication range. The *link layer*, implemented by a *media access control (MAC)* protocol, uses techniques like carrier sense multiple access (CSMA) to arbitrate access to the shared wireless medium to let multiple neighboring devices exchange packets with one another. *Low-power MAC protocols* additionally use radio duty cycling and distinguish between unicast and broadcast to conserve energy, and use per-hop packet retransmissions to help improve reliability [Lan08]. The routing protocol at the *network layer* is then responsible for the end-to-end delivery of packets across multiple hops, for example, by establishing a routing tree that maintains a path from every source to the destination, which represents the root of the tree [AY05]. Only very few transport layer protocols exist, providing services like rate control and end-to-end acknowledgments to further help packet reliability [PG07].

Primary focus on energy. As for meeting the non-functional requirements of low-power wireless applications, the primary focus of existing low-power wireless communication stacks has been on reducing the nodes' energy consumption. Two techniques are commonly employed: radio duty cycling at the data link layer and finding routes that minimize the total number of transmissions per packet at the network layer [Lan08, AY05]. It has been shown that in particular the parameters of the low-power MAC protocol operating at the link layer largely determine not only the energy cost of communication, but also the per-hop latency and reliability of and the bandwidth available for communication [LM10].

The need to adapt. However, identifying a set of MAC parameters such that the resulting performance matches the application requirements is cumbersome and error-prone, but most importantly, a particular choice of MAC parameters may become unfit as the traffic load and/or the network state changes. A few works thus propose to adapt specific MAC protocol parameters at runtime in response to such changes, mostly with the sole goal of keeping the energy consumption to a minimum [JBL07, CWW10].

Focus on single or local metrics. There is even less work incorporating additional metrics, such as per-hop latency and per-hop reliability, into the adaptation decisions [PFJ10]. While these efforts are an important step in the right direction, they fall short of meeting the requirements of low-power applications in that they either focus only on a *single* metric or consider only *local* metrics. Nevertheless, as described in Section 1.1, low-power wireless applications often have requirements along *multiple* metrics, which are most naturally specified in *global, network-wide* terms.

No hard performance guarantees in multi-hop networks. Despite the usefulness of traditional networking stacks in enabling non-critical (data collection) applications, their complexity makes it almost impossible to provide hard guarantees on the end-to-end performance over multiple hops, which are definitely needed to support critical CPS scenarios. The root cause of this complexity is the *wireless link abstraction*: Many protocols on all layers of the stack adopt concepts from wired networks like *unicast transmission* and *routing path*, as shown in Figure 1.2, thereby treating the wireless channel between two devices as a point-to-point link [KRH+06]. The end-to-end behavior of these protocols, then, depends on the quality of multiple links, each of which is subject to several unpredictable factors.

The inability to keep up with the ever-changing network state is indeed the primary reason why previous solutions cannot support applications that require packets to be delivered within hard real-time deadlines. Both industry standards [har, isa] and research prototypes [OBB+13, SNSW10] exist that compute at runtime transmission schedules tailored for each node in the network at a central entity, based on information about the global network state. Assuming the latter would not change at all, these approaches could in principle guarantee end-to-end packet deadlines. In the real world, however, the network state changes, and because it takes considerable time from when such change occurs until when a change is reflected in new transmissions schedules—on the order of several minutes based on anecdotal evidence reported by our contacts at ABB Research—these routing-based solutions are fundamentally incapable of supporting hard real-time applications. This is also acknowledged by major industry players who contributed to the WirelessHART standard: "... none of the technologies provide any hard guarantees on deadlines, which is needed if you should dare to use the technology in critical applications" [Per].

1.4 Thesis Contributions and Road Map

To address these shortcomings, this thesis makes three key contributions.

Meeting soft network-wide performance requirements (Chapter 2). To serve the needs of real-world data collection applications, we introduce pTunes, a framework for runtime adaptation of low-power MAC protocol parameters. Compared with prior solutions, pTunes takes a more holistic approach by allowing the user to specify *multiple* performance goals from a *global, network-wide* perspective. These performance goals, specified in terms of network lifetime, end-to-end reliability, and end-to-end latency, represent soft requirements that are to be satisfied in the long run. Given a concrete requirements specification and a traditional low-power wireless stack running at each node, pTunes adapts the operational parameters of the low-power MAC protocol at runtime to meet the requirements against dynamic changes in network state, traffic load, and routing topology.

As detailed in Chapter 2, pTunes rests upon three building blocks:

- The design of pTunes revolves around a centralized approach that is similar in spirit to a model-predictive controller. To reason about network-wide performance, pTunes periodically collects at a central entity (e.g., the base station in a deployment) reports from each node that contain local routing and network state, among others. Based on the thus obtained *global network view* and an accurate *performance model*, pTunes first checks whether the application requirements are violated. If so, it automatically solves a *multi-objective optimization problem* in order to determine MAC parameters so that the predicted performance matches again the application requirements under the current global network view. The determined MAC parameters are then distributed in the network and installed on all devices.

- We structure the aforementioned performance model in a layered fashion, clearly separating application-level, protocol-independent, and protocol-dependent modeling quantities. This way, we simplify the integration of a different MAC protocol into pTunes by reusing common expressions and identifying the minimum set of quantities that needs to be altered. We show the effectiveness of our modeling approach by applying it to two state-of-the-art protocols, X-MAC and LPP, based on their implementations in Contiki.

- We design an efficient runtime support to "close the loop" in pTunes. Our approach uses fast and reliable Glossy floods [FZTS11] to collect network state and disseminate new MAC parameters. This enables pTunes to gather *consistent* snapshots of network state, taken with microsecond accuracy at all nodes simultaneously, with low energy costs and independent of other protocols running concurrently.

We demonstrate using testbed experiments that pTunes can achieve severalfold improvements in network lifetime over fixed MAC parameters, while satisfying soft end-to-end latency and reliability requirements in the long run despite unforeseen changes in the network caused by, for example, wireless interference and multiple node failures.

Modeling protocols that utilize synchronous transmissions (Chapter 3). The effectiveness of pTunes is fundamentally dependent on the accuracy of the performance model, which maps the global network view and the MAC protocol parameters to the three performance metrics we target. In essence, it is the performance model that closes the large conceptual gap between the high-level application requirements and the low-level MAC protocol parameters, and the solver exercises the model while computing the latter in order to satisfy the former. More generally, accurate models of a network's end-to-end performance can greatly aid in the design and verification of emerging systems, including CPS that "... must operate dependably, safely, securely, efficiently, and in real-time. [RLSS10]"

Unfortunately, traditional multi-hop low-power wireless protocols as considered in Chapter 2 are intricate and difficult to model. This is because their operation is conditional on the ever-changing network state, which leads to unpredictable and often uncoordinated changes in the protocol's behavior, for example, when some node in the routing tree locally decides to forward packets to a different parent [GFJ+09]. As a result, previous modeling efforts often stop at the link layer, where distributed interactions span only a single hop and hence reasoning is still manageable, achieving model errors in the range of 2–7 % (see Section 2.5.2). Only a few works model higher-layer functionality [YZDPHg11, GB12], but their validation is limited to simulations, which lack precisely the real-world dynamics of low-power wireless that complicate the modeling in the first place.

Fueled by our own work on the Glossy flooding architecture [FZTS11] and the Low-Power Wireless Bus (LWB) [FZMT12], a radically different breed of communication protocols has emerged that utilizes *synchronous transmissions*. Rather than one sender transmitting over a dedicated wireless link to a receiver, using synchronous transmissions *multiple* senders transmit *simultaneously* to the receiver. The sender diversity [RHK10] and two physical-layer phenomena, constructive baseband interference and capture effects [WLS14], let synchronous transmissions achieve a higher one-hop packet reliability than link-based transmissions [DDHC+10].

Chapter 3 of this thesis shows that certain protocols using synchronous transmissions are also *simpler to model* than link-based protocols with an *unparalleled accuracy*. Specifically, Chapter 3 contributes the following:

- Using statistical time series analyses of a large set of packet reception traces collected through extensive testbed experiments, we find that packet receptions and losses in Glossy largely adhere to a sequence of independent and identically distributed (i.i.d.) Bernoulli trials. This so-called *Bernoulli assumption* is typically made to simplify the modeling, yet we find that this assumption is significantly less valid when modeling protocols that operate on individual wireless links.
- Leveraging the validity of the Bernoulli assumption to synchronous transmissions, we devise a simple Markovian model that estimates LWB's long-term energy consumption with an unparalleled error of 0.25 % relative to real measurements, and sufficient conditions to give probabilistic guarantees on LWB's end-to-end packet reliability. In doing so, we demonstrate for the first time the accurate modeling of a *complete* multi-hop low-power wireless networking solution.

These results are particularly relevant to CPS applications employing feedback control. Many control algorithms can be designed to tolerate a small fraction of packet loss, say, less than 1 %, without sacrificing control performance and stability. Nevertheless, this assumes that the few losses do not happen as a longer burst of multiple consecutive losses [SSF+04]. The validity of the Bernoulli assumption for synchronous transmissions essentially says that such adverse bursts virtually never occur when using, for example, Glossy to communicate packets throughout the network.

Meeting hard real-time requirements with low energy costs (Chapter 4). Since LWB employs only Glossy for communication and has been shown to keep end-to-end packet loss rates below 1 % [FZMT12], LWB could be a good candidate protocol for supporting CPS applications. Indeed, LWB's operation resembles that of *wired* fieldbusses, such as FlexRay [MT06] and Time-Triggered Protocol [KG93], which are used in classical embedded systems with high dependability and real-time requirements. Using LWB, nodes are synchronized and an appointed *host* node repeatedly computes a communication schedule that globally allocates non-overlapping time slots to nodes that have pending packets. That is, there is just one *global* schedule that applies to all nodes in the network, and every time slot in this schedule corresponds to a distinct network-wide Glossy flood. While working with Glossy and LWB over the past years, we began to nourish the hope that it could be possible to support CPS applications with hard real-time requirements by leveraging LWB's bus-like operation.

To show that our intuition was correct, we design Blink, the first low-power wireless protocol providing hard real-time guarantees on end-to-end packet deadlines in large multi-hop networks, while simultaneously incurring low energy costs. Blink uses LWB as underlying communication support, yet the original LWB scheduler is completely oblivious of packet deadlines. The key observation that makes Blink immune to the problem that prevents prior solutions from providing real-time guarantees across multiple hops (see Section 1.3) is that because Glossy's protocol logic is independent of the current network state, we do not need to consider the time-varying network state as an input to the scheduling problem either.

As detailed in Chapter 4, Blink's design rests upon three components:

- In LWB all nodes follow the same schedule, while Glossy provides very accurate network-wide time synchronization and allows us to ignore the network state. Due to these properties we can treat an entire multi-hop low-power wireless network as *a single resource that runs on a single clock*. This abstraction is powerful in that it allows us to map the real-time scheduling problem in Blink to *uniprocessor scheduling*, which is well known and easier to solve than the multiprocessor scheduling problem found in prior work [SXLC10].

- We conceive scheduling policies based on the earliest deadline first (EDF) principle [LL73]. Blink uses these policies to compute online a schedule that provably meets all deadlines of admitted packet streams, while minimizing the network-wide energy consumption within the limits of the underlying LWB communication support, tolerating changes in both the network state and the set of streams.

- We design and implement a highly efficient priority queue as well as algorithms that make use of it to enable EDF scheduling on resource-poor devices. Based on these, we can demonstrate the first working implementation of EDF on low-power embedded platforms.

We evaluate a Blink prototype on two testbeds, showing that it meets nearly 100 % of packet deadlines; the few deadline misses are entirely due to packet loss, which cannot be completely avoided in a wireless setting. Moreover, experiments on three state-of-the-art MCUs show that, thanks to our data structures and algorithms, Blink achieves speed-ups of up to 4.1× relative to a conventional scheduler implementation. These speed-ups prove instrumental to the viability of EDF-based real-time scheduling on specific low-power embedded platforms.

2

pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols

Media access control (MAC) protocols play a key role in determining the performance of low-power wireless networks, but very few of the many proposed solutions have been used in real deployments [KGN09, RC08].

Challenges. There exists a significant conceptual gap between the high-level application requirements on the one hand and the low-level MAC protocol operation on the other [KGN09]. In particular, it requires expert knowledge to find operating parameters of the low-power MAC protocol such that the performance satisfies given application requirements.

In most deployments today, the choice of MAC parameters is based on experience and rules of thumb involving a coarse-grained analysis of expected network load and topology dynamics. This can yield a performance far off the application requirements [LM10]. Alternatively, system designers conduct several field trials in order to identify suitable MAC parameters [CCD+11]. This time-consuming and deployment-specific practice, however, is hardly sustainable in the long term.

Even if the MAC parameters are appropriate at one time, they are likely to perform poorly when the network state changes. The quality of low-power wireless links varies greatly over time, leading to unpredictable packet loss [ZG03]; harsh environmental conditions cause nodes to be temporarily disconnected or to fail [BGH+09]; and changes in the routing topology or the sensing activity result in fluctuating traffic load. Statically

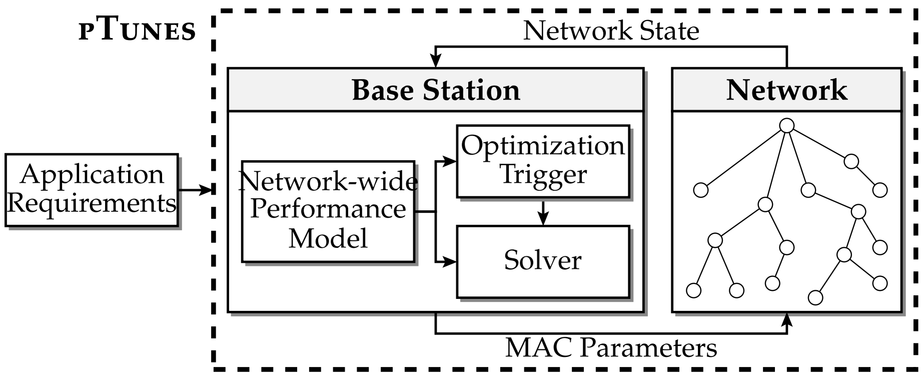


Figure 2.1: Overview of the pTunes framework. pTunes *takes advantage of a centralized approach that shares some similarities with a model-predictive controller.*

configured MAC protocols cannot cope with these dynamics.

To perform efficiently at all times, MAC protocols must adapt their operating parameters at runtime. One way to approach this problem is to embed adaptivity within the protocol operation [HB10]. This, however, hard-codes the adaptation policies and hence limits their applicability. Instead, separating adaptivity from the protocol operation enables higher-layer services to dynamically adjust the operating parameters [PHC04]. Although a few mechanisms utilize these "control knobs," they either focus on a single performance metric—typically energy [JBL07, MWZT10, CWW10]—or consider only local metrics, such as per-hop latency [PFJ10, BYAH06]. Real-world applications, however, often need to balance *multiple* conflicting performance metrics, such as reliability, energy, and latency, expressed on a *network-wide* scale [CMP+09, SMP+04, WALJ+06].

Contributions and road-map. To tackle the issues above, we present pTunes, a framework for runtime adaptation of low-power MAC protocol parameters. pTunes allows users to specify application requirements in terms of *network lifetime*, *end-to-end reliability*, and *end-to-end latency*, which are key performance metrics in real-world applications [CCD+11, CMP+09, SMP+04, WALJ+06, TPS+05]. Based on information about the current network state, pTunes automatically determines optimized MAC parameters whose performance meets the requirements specification.

This chapter makes the following contributions:

- We introduce the pTunes framework, targeting data collection systems employing tree routing atop low-power MAC protocols. As shown in Figure 2.1, using pTunes a base station collects reports on the *network state*, such as topology and link quality information, to evaluate the network-wide metrics we target. The *optimization trigger* decides when to carry out the parameter optimization, based on a periodic timer or some mechanism that uses the *network-wide performance model* to check if the *application requirements* are violated under the current network state. The *solver* determines *optimized MAC parameters*, which are disseminated in the network and installed on all nodes. Section 2.1 further characterizes the *multi-objective* parameter optimization problem in pTunes.

- We design a well-structured modeling framework to solve the parameter optimization problem. Our layered modeling approach, described in Section 2.2, separates application-level, protocol-independent, and protocol-dependent quantities. This increases generality and flexibility, as it cleanly determines what needs to be changed to account for a different MAC protocol. We apply this modeling approach to two state-of-the-art protocols, X-MAC [BYAH06] and LPP [MELT08], based on their implementations in Contiki. We use these models throughout this chapter, ultimately demonstrating that they are both practical and accurate.
- We present the design and implementation of an efficient system support to address the system-level challenges arising in pTunes. These include, for instance, the timely collection of accurate network state with little energy overhead and minimum disruption for the application operation. As described in Section 2.3, unlike most approaches in the literature, we meet these requirements with a novel solution for collecting network state and disseminating new MAC parameters *independent* of other protocols running concurrently. Our approach utilizes fast and reliable Glossy network floods [FZTS11], allowing pTunes to collect *consistent* network state snapshots, taken with microsecond accuracy at all nodes simultaneously, with very low energy cost.

After illustrating implementation details in Section 2.4, we evaluate pTunes in Section 2.5 using experiments with X-MAC and LPP on a 44-node testbed. For instance, we find that adapting their parameters using pTunes enables up to three-fold lifetime gains over static MAC parameters optimized for peak traffic, the latter being current practice in many real deployments [KGN09]. pTunes promptly reacts to changes in traffic load and link quality, meeting application-level requirements through an 80 % reduction in packet loss during periods of controlled wireless interference. Moreover, we find that pTunes helps the routing protocol recover from critical network changes, reducing the total number of parent switches and settling quickly on a stable, high-quality routing topology. This reduces packet loss by 70% in a scenario where multiple core routing nodes fail simultaneously.

We discuss design trade-offs of pTunes in Section 2.6, review related work in Section 2.7, and provide brief concluding remarks in Section 2.8.

2.1 Optimization Problem

In pTunes, we simultaneously consider three key performance metrics of real-world applications [CCD+11, CMP+09, SMP+04, WALJ+06, TPS+05]: network lifetime T, end-to-end reliability R, and end-to-end latency L. The MAC parameter optimization problem thus becomes a *multi-objective optimization problem (MOP)*. This involves optimizing the objective functions $T(\mathbf{c})$, $R(\mathbf{c})$, and $L(\mathbf{c})$, where $\mathbf{c}$ is a vector of MAC parameters, or *MAC configuration* for short. There may exist not one unique optimal solution to this MOP, but rather a set of solutions that are optimal in the sense that no other solution is superior in *all* objectives. These are known as *Pareto-optimal* solutions and represent different optimal trade-offs among T, R, and L.

Given the many Pareto-optimal solutions, a natural question is which solution best serves the application demands. pTunes needs to make this decision at runtime in an automated fashion, without involving the user (e.g., to manually select a solution from a set of candidates). With this requirement in mind, we adopt from among the many MOP solving techniques an approach inspired by the epsilon-constraint method [HLW71]. This method treats all but one objective as constraints, and thus provides a natural interface for specifying typical requirements of low-power wireless systems such as "batteries should last for at least 6 months." Using this approach, pTunes solves the MOP by optimizing one objective subject to constraints on the remaining objectives

$$\begin{array}{lll} \text{Maximize/Minimize} & M_1(\mathbf{c}) & \\ \text{Subject to} & M_2(\mathbf{c}) & \geq, \leq \quad C_1 \\ & M_3(\mathbf{c}) & \geq, \leq \quad C_2 \end{array} \tag{2.1}$$

where each M_i is one among $\{T, R, L\}$ and $\{C_1, C_2\}$ are *soft* requirements to be satisfied in the long run, corresponding to the best-effort operation of many data collection systems [GFJ+09]. By varying $\{C_1, C_2\}$, all Pareto-optimal solutions can be generated. Based on concrete values for $\{C_1, C_2\}$ set by the user on some objectives, pTunes translates the application requirements into a solution that optimizes the remaining objective. The resulting solution is Pareto-optimal while representing the trade-off provided by the user.

As an example, in long-term structural monitoring the major concern is typically network lifetime, but domain experts also require a certain

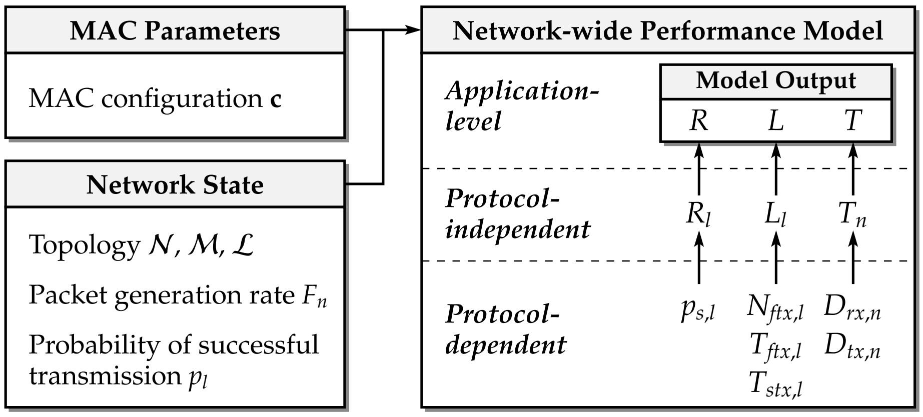

MAC Parameters
MAC configuration **c**
Network State
Topology $\mathcal{N}$, $\mathcal{M}$, $\mathcal{L}$
Packet generation rate F_n
Probability of successful transmission p_l
Network-wide Performance Model
Application-level
Model Output
R L T
Protocol-independent
R_l L_l T_n
Protocol-dependent
$p_{s,l}$ $N_{ftx,l}$ $T_{ftx,l}$ $T_{stx,l}$ $D_{rx,n}$ $D_{tx,n}$

Figure 2.2: Modeling framework of pTunes with inputs, output, and mapping between the different modeling layers. *The layered modeling approach simplifies the integration of new MAC protocols into* pTunes *by fostering reuse of common expressions and clearly identifying the minimum set of quantities that needs to be changed.*

reliability in delivering sensed data [CMP+09]. Instantiating (2.1), the user would specify the maximization of network lifetime subject to a minimum end-to-end reliability as follows

$$\begin{array}{lll} \text{Maximize} & T(\mathbf{c}) & \\ \text{Subject to} & R(\mathbf{c}) \;\geq\; R_{min} \end{array} \tag{2.2}$$

In addition, the user may impose an additional constraint on end-to-end latency, $L(\mathbf{c}) \leq L_{max}$, in case timely data delivery is also relevant.

2.2 Modeling Framework

To facilitate using pTunes with different low-power MAC protocols, we break up the modeling into three distinct layers, as shown in the model frame in Figure 2.2. The upper layer defines application-level metrics (R, L, T) as functions of link and node-specific metrics (R_l, L_l, T_n). The middle layer expresses these metrics in a protocol-independent manner, and provides the entry point for the modeling of a concrete MAC protocol by exposing six terms to the lower protocol-dependent layer. Binding these terms to concrete protocol-specific expressions is sufficient to adapt the network-wide performance model in pTunes to a given MAC protocol.

Model inputs are the MAC parameters and the network state, comprising information about routing topology, traffic volumes, and link qualities. As a measure of the latter, we take the probability of successful transmission p_l over the link to the parent in the routing tree. To keep

Table 2.1: Terms denoting network state and protocol-dependent quantities.

Term	Description
$\mathcal{N}$	Set of all nodes in the network excluding the sink
$\mathcal{M}$	Set of source nodes generating packets
$\mathcal{L}$	Set of all links forming the routing tree
F_n	Packet generation rate of node n
p_l	Probability of successful transmission over link l
$p_{s,l}$	Probability of successful unicast transm. over link l
$N_{ftx,l}$	Number of failed unicast transmissions before success over link l
$T_{ftx,l}$	Time for a failed unicast transmission over link l
$T_{stx,l}$	Time for a successful unicast transmission over link l
$D_{rx,n}$	Fraction of time radio is in receive mode at node n
$D_{tx,n}$	Fraction of time radio is in transmit mode at node n

our models simple and practical, we assume the delivery of individual packets to be independent of their size, of the delivery of any other packet, and of the link direction they travel along. As illustrated in Section 2.3, our runtime evaluation of p_l captures the impact of channel contention on link quality, allowing us not to consider it explicitly in our models. Testbed experiments in Section 2.5.2 show that this approach results in highly accurate models for both X-MAC and LPP.

2.2.1 Application-level Metrics

In a typical data collection scenario with static nodes, a tree-shaped routing topology provides a unique path from every sensor node to a sink node. These paths are generally time-varying, as the routing protocol adapts them according to link quality estimates among other things [GFJ+09, PH10]. In the following, we use $\mathcal{N}$ to denote the set of *all nodes* in the network excluding the sink, and $\mathcal{M} \subseteq \mathcal{N}$ to denote the set of *source nodes* generating packets. We also indicate with $\mathcal{L}$ the set of *communication links* that form the current routing tree. The *path* $\mathcal{P}_n \subseteq \mathcal{L}$ originating at node $n \in \mathcal{M}$ includes all intermediate links that connect node n to the sink. Table 2.1 lists these and other modeling terms we use to denote network state and protocol-dependent quantities.

End-to-end reliability and latency. The reliability $R_{\mathcal{P}_n}$ of path $\mathcal{P}_n$ is the expected fraction of packets delivered from node $n \in \mathcal{M}$ to the sink along $\mathcal{P}_n$. Thus, $R_{\mathcal{P}_n}$ is the product of per-hop reliabilities R_l, $l \in \mathcal{P}_n$. We define the *end-to-end reliability* R as the average reliability of all paths $\mathcal{P}_n$.

$$R = \frac{1}{|\mathcal{M}|} \sum_{n \in \mathcal{M}} R_{\mathcal{P}_n} = \frac{1}{|\mathcal{M}|} \sum_{n \in \mathcal{M}} \left(\prod_{l \in \mathcal{P}_n} R_l \right) \tag{2.3}$$

Likewise, the latency $L_{\mathcal{P}_n}$ of path $\mathcal{P}_n$ is the expected time between the first transmission of a packet at node $n \in \mathcal{M}$ and its reception at the sink. Thus, $L_{\mathcal{P}_n}$ is the sum of per-hop latencies L_l, $l \in \mathcal{P}_n$. Similar to (2.3), we define the *end-to-end latency* L for *successfully delivered* packets as the average latency of all paths $\mathcal{P}_n$, and omit the formula.

We define R and L as averages of all source-sink paths since the global, long-term performance is of ultimate interest for most data collection systems [WALJ+06, TPS+05, SMP+04]. Local, short-term deviations from the requirements are usually tolerated, provided they are compensated in the long run. In other scenarios (e.g., industrial settings), it might be more appropriate to define R and L as the minimum reliability and the maximum latency among all source-sink paths, which would only require modifying the two definitions above.

Network lifetime. Similar to prior work [ML06], we define the *network lifetime* T as the expected shortest node lifetime T_n, $n \in \mathcal{N}$. We assume the sink has infinite energy supply.

$$T = \min_{n \in \mathcal{N}} (T_n) \tag{2.4}$$

This choice is motivated by the fact that a single node failure can lead to network partition and service interruption. It is also possible to express other notions of network lifetime in pTunes, such as the time until some fraction of nodes fails, again requiring only to modify (2.4).

2.2.2 Protocol-independent Modeling

The section above expressed the application-level metrics R, L, and T as functions of per-hop reliability R_l, per-hop latency L_l, and node lifetime T_n (see Figure 2.2). We now define the latter three in a protocol-independent manner, which increases flexibility and generality by isolating protocol-dependent quantities.

Per-hop reliability and latency. Several factors influence these metrics: (*i*) the MAC operation when transmitting packets, (*ii*) packet queuing throughout the network stack due to insufficient bandwidth, and (*iii*) application-level buffering, for example, to perform in-network processing. The MAC parameters have an impact on (*i*) and may avoid the occurrence of (*ii*), provided a MAC configuration exists that provides sufficient bandwidth for the current traffic load. Application-specific in-network functionality akin to (*iii*) is out of the scope of this work.

We present next expressions for per-hop reliability and latency due to the MAC operation, corresponding to (*i*). Additionally, pTunes includes models to detect situations akin to (*ii*). In fact, as we show in Section 2.5.2, pTunes automatically adjusts the MAC parameters to provide higher bandwidth against increased traffic, thus avoiding the occurrence of local packet queuing until the network capacity attainable in our experimental setting is fully exhausted.

We define the *per-hop reliability* R_l of link $l \in \mathcal{L}$, which connects node $n \in \mathcal{N}$ to its parent m in the routing tree, as the probability that n successfully transmits a packet to m.

$$R_l = 1 - (1 - p_{s,l})^{N+1} \tag{2.5}$$

Here, $p_{s,l}$ represents the MAC-dependent probability that a single unicast transmission over link l succeeds, and N is the maximum number of retransmissions per packet, modeling automatic repeat request (ARQ) mechanisms used by many MAC protocols to improve reliability.

Furthermore, we define the *per-hop latency* L_l of link l as the time for node n to deliver a message to its parent m.

$$L_l = N_{ftx,l} \cdot T_{ftx,l} + T_{stx,l} \tag{2.6}$$

Here, $T_{ftx,l}$ and $T_{stx,l}$ are the MAC-dependent times needed for each failed and the final successful transmission, and $N_{ftx,l}$ is the expected number of failed transmissions before the final successful one.

To derive an expression for $N_{ftx,l}$, let $p_{f,l} = 1 - p_{s,l}$ be the probability that a single transmission over link l fails, and $p_{f,l}(k) = p_{f,l}(0) \cdot p_{f,l}^k$ the probability of k, $0 \leq k \leq N$, consecutive failed transmissions, where $p_{f,l}(0)$ denotes the probability that already the first transmission succeeds (i.e., no transmission fails). There can be between 0 and N failed packet transmissions. To compute the expectation $N_{ftx,l}$, we sum over all possible values $0 \leq k \leq N$, weighted by their probabilities of occurrence $p_{f,l}(k)$.

$$\begin{aligned} N_{ftx,l} &= \sum_{k=0}^{N} k \cdot p_{f,l}(k) \\ &= p_{f,l}(0) \cdot p_{f,l} \cdot \sum_{k=0}^{N} k \cdot p_{f,l}^{k-1} \end{aligned} \tag{2.7}$$

Since we consider the per-hop latency L_l in (2.6) only for *delivered* packets, that is, for packets that are eventually successfully transmitted, the sum of the different probabilities $p_{f,l}(k)$ for all possible k amounts to 1.

$$\begin{aligned} 1 &= \sum_{k=0}^{N} p_{f,l}(k) \\ &= p_{f,l}(0) \cdot \frac{1 - p_{f,l}^{N+1}}{1 - p_{f,l}} \end{aligned} \tag{2.8}$$

From this, we immediately obtain an expression for the probability $p_{f,l}(0)$ that the first packet transmissions succeeds.

$$p_{f,l}(0) = \frac{1 - p_{f,l}}{1 - p_{f,l}^{N+1}} \tag{2.9}$$

Replacing $p_{f,l}(0)$ in (2.7) with the expression in (2.9) we get

$$\begin{aligned} N_{ftx,l} &= \frac{p_{f,l} \cdot (1 - p_{f,l})}{1 - p_{f,l}^{N+1}} \cdot \sum_{k=0}^{N} k \cdot p_{f,l}^{k-1} \\ &= \frac{p_{f,l} \cdot (1 - p_{f,l})}{1 - p_{f,l}^{N+1}} \cdot \left(\frac{1 - p_{f,l}^{N+1}}{(1 - p_{f,l})^2} - \frac{(N+1) \cdot p_{f,l}^{N}}{1 - p_{f,l}} \right) \\ &= \frac{p_{f,l}}{1 - p_{f,l}} - (N+1) \cdot \frac{p_{f,l}^{N+1}}{1 - p_{f,l}^{N+1}} \end{aligned} \tag{2.10}$$

Node lifetime. Sensor nodes consume energy by communicating, sensing, processing, and storing data. Adapting the MAC parameters has no significant impact on the latter three, but affects energy expenditures on communication to a large extent, as the radio is typically a major energy consumer. Given a battery capacity Q, we define the *node lifetime* T_n of node $n \in \mathcal{N}$ as

$$T_n = Q/(D_{tx,n} \cdot I_{tx} + D_{rx,n} \cdot I_{rx} + D_{idle,n} \cdot I_{idle}) \tag{2.11}$$

where I_{tx}, I_{rx}, and I_i are the current draws of the radio in transmit, receive, and idle mode. T_n is thus the expected node lifetime based on the fractions of time in each mode $D_{tx,n}$, $D_{rx,n}$, and $D_{idle,n} = 1 - D_{tx,n} - D_{rx,n}$, which depend on the MAC protocol and the traffic volume at node n.

The *traffic volume* is the rate at which nodes send and receive packets. A node $n \in \mathcal{N}$ generates packets at rate F_n and receives packets from its children $C_n \subseteq \mathcal{N}$ in the routing tree, if any. The rate of packet reception depends on each child's packet transmission rate $F_{tx,c}$ and the individual per-hop reliabilities R_{l_c} of links l_c, $c \in C_n$, connecting each child c with n. Thus, node n transmits packets at rate

$$F_{tx,n} = (N_{rtx,l} + 1) \cdot \left(F_n + \sum_{c \in C_n} F_{tx,c} \cdot R_{l_c} \right) \tag{2.12}$$

Here, $N_{rtx,l}$ is the expected number of retransmissions per packet over link l. To compute it, we have to sum over all possible values, weighted by their probabilities of occurrence. The probability of k, $0 \leq k < N$, retransmissions is $p_{rt,l}^{k} \cdot (1 - p_{rt,l})$; that is, the first k packet transmissions fail (each with probability $p_{rt,l}$) and the last one succeeds (with probability $1 - p_{rt,l}$). With $p_{rt,l}^{N}$ denoting the probability that the maximum number of packet retransmissions is exhausted without delivering the packet, we derive the expected number of retransmissions per packet as follows.

$$\begin{aligned}
N_{rtx,l} &= N \cdot p_{rt,l}^{N} + \sum_{k=0}^{N-1} k \cdot p_{rt,l}^{k} \cdot (1 - p_{rt,l}) \\
&= N \cdot p_{rt,l}^{N} + p_{rt,l} \cdot (1 - p_{rt,l}) \cdot \sum_{k=0}^{N-1} k \cdot p_{rt,l}^{k-1} \\
&= N \cdot p_{rt,l}^{N} + p_{rt,l} \cdot (1 - p_{rt,l}) \cdot \left(\frac{1 - p_{rt,l}^{N}}{(1 - p_{rt,l})^2} - \frac{N \cdot p_{rt,l}^{N-1}}{1 - p_{rt,l}} \right) \\
&= \frac{p_{rt,l} \cdot (1 - p_{rt,l}^{N})}{1 - p_{rt,l}} \qquad (2.13)
\end{aligned}$$

Packet queuing. Whenever node n enqueues packets at a higher rate than it forwards (i.e., dequeues) packets, packets start to queue up at node n. The former rate is given by

$$F_0 = F_n + \sum_{c \in C_n} F_{tx,c} \cdot R_{l_c} \qquad (2.14)$$

adding up the local packet generation rate F_n and the rate at which packets are received from n's child nodes C_n. The latter rate, that is, the upper bound on the rate at which node n can forward (dequeue) packets, is the inverse of the expected MAC-dependent time needed for a packet transmission T_{tx}, including retransmissions and packets that are eventually dropped when the maximum number of retransmissions N has been exhausted. Thus, by imposing the constraint

$$\frac{1}{(N_{rtx,l} + 1) \cdot T_{tx}} \geq F_0 \qquad (2.15)$$

where $N_{rtx,l}$ is given by (2.13), we enforce that pTunes selects MAC parameters such that MAC-dependent queuing does not occur. Furthermore, if (2.15) is not satisfiable when estimating the network-wide performance based on the collected network state, pTunes essentially knows and can thus detect that there is MAC-dependent queuing within the network. High-layer functionality, such as data aggregation and other

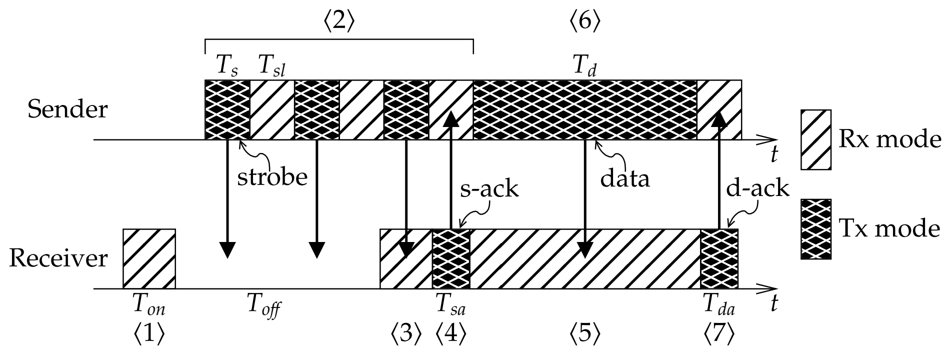


Figure 2.3: Sequence of radio modes and packet exchanges during a successful unicast transmission in X-MAC.

in-network processing, may introduce additional packet *buffering*, which is however independent of the MAC operation.

We demonstrate next the modeling of a concrete MAC protocol. This requires to find expressions for six protocol-specific terms, as shown in Figure 2.2 and described in Table 2.1.

2.2.3 Protocol-specific Modeling

We use two state-of-the-art MAC protocols to exemplify the protocol-specific modeling. X-MAC [BYAH06] is representative of many sender-initiated MAC protocols based on low-power listening (LPL) [PHC04] that proved viable in real-world deployments [KGN09]. More recent work focuses on receiver-initiated MAC protocols such as low-power probing (LPP) [MELT08]. In the following, we refer to implementations of X-MAC and LPP in Contiki 2.3, which we also use in our experiments in Section 2.5.

2.2.3.1 Sender-initiated: X-MAC

Figure 2.3 shows a successful unicast transmission in X-MAC. Nodes wake up periodically for T_{on} to poll the channel ⟨1⟩, where T_{off} is the time between two channel polls. To send a packet, a node transmits a sequence of *strobes* ⟨2⟩, short packets containing the identifier of the receiver. Strobing continues for a period sufficient to make at least one strobe overlap with a receiver wake-up ⟨3⟩. The receiver replies with a *strobe acknowledgment (s-ack)* ⟨4⟩ and keeps the radio on awaiting the transmission of the *data packet* ⟨5⟩. The sender transmits the data packet upon receiving the s-ack ⟨6⟩ and waits for the *data acknowledgment (d-ack)* ⟨7⟩ from the receiver. Afterward, both nodes turn off their radios.

Failed s-ack, d-ack, and data packet transmissions are handled by timeouts. When a timeout occurs, the sender backs off for a random period and retries beginning with the strobing phase, for at most N times. Broadcasts proceed similarly to unicast transmissions, but the strobing phase lasts for $T_m = 2 \cdot T_{on} + T_{off}$ to make a strobe overlap with the wake-up of all neighboring nodes. Nodes receiving a broadcast strobe keep their radio on until they receive the data packet at the end of the sender's strobing phase.

Several variables are adjustable in the X-MAC implementation we consider. However, three specific parameters affect its performance to a major extent.

$$\mathbf{c} = [T_{on}, T_{off}, N] \tag{2.16}$$

We let pTunes adapt these parameters at runtime, leveraging the X-MAC specific models presented next.

Per-hop reliability. We determine $p_{s,l}$ in (2.5), the probability that a single unicast from node n to its parent m succeeds. This is the case if m hears a strobe (with probability $p_{s,l}$), the s-ack reaches n, and m receives the data packet. Each of the latter two succeeds with probability p_l, collected at runtime as part of the network state (see Section 2.3).

$$p_{s,l} = p_{s,l} \cdot p_l^2 \tag{2.17}$$

The probability of receiving at least one strobe is

$$p_{s,l} = 1 - (1 - p_l)^{(T_{on} - T_s)/T_{it}} \tag{2.18}$$

where $T_{it} = T_s + T_{sl}$ is the duration of a strobe iteration at the sender, which includes the length of a strobe transmission T_s and listening T_{sl} for an s-ack.

Per-hop latency. We determine $T_{ftx,l}$ and $T_{stx,l}$ in (2.6), the times spent for failed and successful transmissions. $T_{ftx,l}$ depends on whether node n receives an s-ack. If so, n stops strobing, sends the data packet, and times out after T_{out}. Otherwise, n sends strobes for T_m. In either case, node n backs off for T_b before retransmitting.

$$T_{ftx,l} = (N_{it} T_{it} + T_d + T_{out}) p_{s,l} + T_m (1 - p_{s,l}) + T_b \tag{2.19}$$

Here, $N_{it} = (T_{on} + T_{off})/(2 \cdot T_{it})$ is the average number of strobe iterations before m possibly replies with an s-ack.

The time for a successful transmission $T_{stx,l}$ includes the time to wait for the s-ack and to send the data packet.

$$T_{stx,l} = N_{it} \cdot T_{it} + T_d \tag{2.20}$$

Node lifetime. We determine $D_{tx,n}$ and $D_{rx,n}$ in (2.11), the fractions of time spent by the radio in transmit and receive mode. Both quantities depend on the rate F_{arx,l_c} at which node n attempts to receive a packet from child c over link l_c

$$F_{arx,l_c} = (N_{rtx,l_c} + 1) \cdot F_{tx,c} \cdot p_{s,l_c} \tag{2.21}$$

where $F_{tx,c}$ and p_{s,l_c} are given by (2.12) and (2.18).

We start with $D_{tx,n}$. Node n transmits during packet receptions from child c and during packet transmissions to its parent m. Letting T_{rxt,l_c} and $T_{txt,l}$ denote the expected times the radio is in transmission mode during receptions over link l_c and transmissions over link l, we thus have

$$D_{tx,n} = F_{tx,n} \cdot T_{txt,l} + \sum_{c \in C_n} F_{arx,l_c} \cdot T_{rxt,l_c} \tag{2.22}$$

T_{rxt,l_c} includes the times to transmit the s-ack and, provided s-ack and data packet are successfully transmitted, to send the d-ack, which happens with probability ${p_{l_c}}^2$.

$$T_{rxt,l_c} = T_{sa} + T_{da} \cdot {p_{l_c}}^2 \tag{2.23}$$

To compute $T_{txt,l}$, we distinguish whether node n's parent m receives one of its strobes and successfully replies with a s-ack, which happens with probability $p_{s\text{-}ack,l} = p_{s,l} \cdot p_l$. If so, n is in transmit mode for sending N_{it} strobes and the data packet. Otherwise, n is in transmit mode for sending as many strobes as fit into the maximum length of the strobing period T_m.

$$T_{txt,l} = (N_{it} \cdot T_s + T_d) \cdot p_{s\text{-}ack,l} + \left(\frac{T_m}{T_{it}} \cdot T_s\right) \cdot (1 - p_{s\text{-}ack,l}) \tag{2.24}$$

Next we consider $D_{rx,n}$. Node n is in receive mode during packet transmissions to its parent m and packet receptions from child c. Let $T_{txr,l}$ and T_{rxr,l_c} be the expected times spent by the radio in reception mode during transmissions over link l and receptions over link l_c. The fraction of time in receive mode for *actual communication* is

$$D_{rxc,n} = F_{tx,n} \cdot T_{txr,l} + \sum_{c \in C_n} F_{arx,l_c} \cdot T_{rxr,l_c} \tag{2.25}$$

To compute T_{rxr,l_c}, we note that n is in receive mode during receptions from child c along link l_c to receive a strobe and the data packet. We account for the time to receive a strobe in the time for channel checks, as per (2.28). Thus, if the s-ack sent by node n and the data packet sent by c are successfully transmitted, n is in receive mode to receive the data packet; otherwise, n is in receive mode for T_{out} until a timeout expires. Additionally, we consider the turnaround time T_{turn} for switching between transmit and received modes time in receive mode.

$$T_{rxr,l_c} = 2 \cdot T_{turn} + (T_{turn} + T_d) \cdot {p_{l_c}}^2 + (T_{out}) \cdot \left(1 - {p_{l_c}}^2\right) \tag{2.26}$$

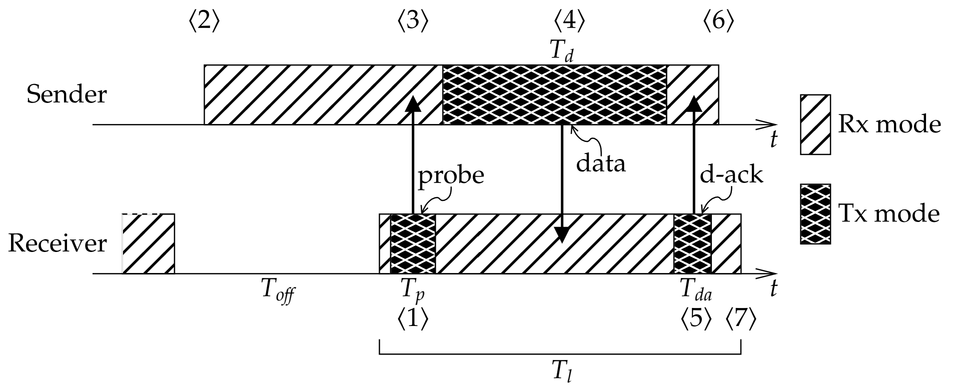


Figure 2.4: Sequence of radio modes and packet exchanges during a successful unicast transmission in LPP.

Next, we look at the expected time $T_{txr,l}$ node n is in receive mode during transmissions to its parent m along link l. If m eventually receives a strobe and successfully replies with a s-ack, n spends time in receive mode while waiting T_{sl} for m to reply with a s-ack between two strobes, to receive the d-ack from m, or to wait for the d-ack but timeout after T_{sl}. Otherwise, if either strobe or s-ack transmission fails, n spends time in receive mode during back-to-back strobe transmissions.

$$\begin{aligned} T_{txr,l} = {} & (N_{it} \cdot (2 \cdot T_{turn} + T_{sl})) \cdot p_{s\text{-}ack,l} \\ & + \left(2 \cdot T_{turn} + T_{da} \cdot p_l^2 + T_{sl} \cdot \left(1 - p_l^2\right)\right) \cdot p_{s\text{-}ack,l} \\ & + \left(\frac{T_m}{T_{it}} \cdot (2 \cdot T_{turn} + T_{sl})\right) \cdot (1 - p_{s\text{-}ack,l}) \end{aligned} \tag{2.27}$$

Finally, node n is in receive mode for $F_{cc} = T_{on}/(T_{on} + T_{off})$ during channel checks, which leads to

$$D_{rx,n} = D_{rxc,n} + (1 - D_{rxc,n}) \cdot F_{cc} \tag{2.28}$$

Expected time needed for transmission. To instantiate the constraint in (2.15) to prevent MAC-dependent packet queuing, we give an expression for the expected time needed for a single transmission attempt in X-MAC.

$$\begin{aligned} T_{tx} = {} & \left[N_{it} \cdot T_{it} + 2 \cdot T_{turn} + T_d + T_{da} \cdot p_l^2 + (T_{out} + T_b) \cdot (1 - p_l^2)\right] \cdot p_{s\text{-}ack,l} \\ & + (T_m + T_b) \cdot (1 - p_{s\text{-}ack,l}) \end{aligned} \tag{2.29}$$

2.2.3.2 Receiver-initiated: LPP

Figure 2.4 shows a successful unicast transmission in LPP. Nodes periodically turn on their radio for T_l and transmit a short *probe* ⟨1⟩

containing their own identifier. To send a packet, a node turns on its radio ⟨2⟩ and listens for a probe from the intended receiver ⟨3⟩, for at most T_{on}. Then the sender transmits the data packet ⟨4⟩, waits for the d-ack from the receiver ⟨5⟩, and goes back to sleep ⟨6⟩. After sending the d-ack, the receiver keeps the radio on until a timeout signals the end of the active phase ⟨7⟩. Between two active phases nodes sleep for T_{off}. To send a broadcast, the sender keeps its radio on for $T_m = 2 \cdot T_l + T_{off}$ to receive a probe from every neighbor, immediately replying to each received probe with the data packet. We let pTunes adapt the same set of LPP parameters **c** in (2.16) as for X-MAC (note that T_{on} has now a different meaning as explained above).

Per-hop reliability. A single LPP unicast from node n to its parent m succeeds if n receives a probe from m (with probability $p_{p,l}$) and then successfully transmits the data packet (with probability p_l).

$$p_{s,l} = p_{p,l} \cdot p_l \tag{2.30}$$

The probability that n receives a probe is given by

$$p_{p,l} = 1 - (1 - p_l)^k \tag{2.31}$$

where $k = (T_{on} - T_p)/T$ is the number of possible probe receptions while node n listens for at most T_{on}. The term $T = T_l + T_{off} + T_{rm}/2$ denotes the LPP duty cycle period, which is the sum of radio on-time, radio off-time, and a small random quantity with uniform distribution $\{0, \ldots, T_{rm}\}$ to scatter probe transmissions.

Per-hop latency. We determine the time for a failed transmission. If node n receives a probe after waiting for $T_{pw,l}$, it sends the data packet and times out after T_{out}. Otherwise, n listens for T_{on}. Node n retransmits after backing off for T_b.

$$T_{ftx,l} = (T_{pw,l} + T_d + T_{out})p_{p,l} + T_{on}\left(1 - p_{p,l}\right) + T_b \tag{2.32}$$

On average, node n receives a probe from its parent m after

$$T_{pw,l} = T_p + \sum_{i=1}^{\lfloor k \rfloor + 1} p_i \cdot T_i \tag{2.33}$$

where p_i is the probability that n receives the i-th probe, and T_i is the expected time to await the i-th probe. To compute p_i, we first write the possible terms as a function of the probability p_1 to receive the first probe.

$$p_i = \begin{cases} p_1 \cdot (1 - p_l)^{i-1} & \text{if } 1 \le i \le \lfloor k \rfloor \\ p_1 \cdot (1 - p_l)^{\lfloor k \rfloor} \cdot (k - \lfloor k \rfloor) & \text{if } i = \lfloor k \rfloor + 1 \end{cases} \tag{2.34}$$

Assuming a probe is eventually received, $\sum_{i=1}^{\lfloor k\rfloor+1} p_i = 1$ holds, and by expanding the sum we find an expression for p_1.

$$p_1 = p_l \cdot \left[1 - (1 - p_l)^{\lfloor k\rfloor} \cdot (1 - p_l \cdot (k - \lfloor k\rfloor))\right]^{-1} \tag{2.35}$$

By substituting (2.35) in (2.34), we obtain a general expression for p_i. The expected time needed to receive the i-th probe is given by

$$T_i = \begin{cases} (i - \frac{1}{2}) \cdot T & \text{if } 1 \leq i \leq \lfloor k\rfloor \\ \frac{k+\lfloor k\rfloor}{2} \cdot T & \text{if } i = \lfloor k\rfloor + 1 \end{cases} \tag{2.36}$$

The time for a successful transmission includes the time to wait for a probe and to send the data packet.

$$T_{stx,l} = T_{pw,l} + T_d \tag{2.37}$$

Node lifetime. We determine the fractions of time in transmit and receive mode. Both depend on the rate F_{arx,l_c} at which node n receives packets from child c over link l_c

$$F_{arx,l_c} = (N_{rtx,l_c} + 1) \cdot F_{tx,c} \cdot p_{s,l_c} \tag{2.38}$$

where $F_{tx,c}$ and p_{s,l_c} are given by (2.12) and (2.30).

Node n transmits a probe every duty cycle period T and sends d-acks to child c with frequency F_{arx,l_c}. Further, n is in transmit mode for $T_{txt,l}$ to send packets to m.

$$D_{tx,n} = T_p/T + T_{da} \sum_{c\in C_n} F_{arx,l_c} + F_{tx,n} \cdot T_{txt,l} \tag{2.39}$$

Node n is in transmit mode for $T_{txt,l}$ during packet transmissions for the time needed to transmit the data packet T_d if it receives a probe from its parent m, which happens with probability $p_{p,l}$.

$$T_{txt,l} = p_{p,l} \cdot T_d \tag{2.40}$$

Node n is in receive mode when the radio is turned on but does not transmit probes or d-acks. Additionally, node n is in receive mode for $T_{txr,l}$ during packet transmissions.

$$D_{rx,n} = (T_l - T_p)/T - T_{da} \sum_{c\in C_n} F_{arx,l_c} + F_{tx,n} \cdot T_{txr,l} \tag{2.41}$$

We define $D_{on} = T_l \cdot F_{dc}$ as the average fraction of time a node has its radio turned on, where $F_{dc} = 1/T$ is the duty cycle frequency. With this, we can express the time in receive mode during packet transmissions as follows

$$\begin{aligned} T_{txr,l} = p_{p,l} \cdot \left[T_{pw,l} \cdot (1 - D_{on}) + 4 \cdot T_{turn} + p_l^2 \cdot T_{da} + (1 - p_l^2) \cdot T_{out}\right] \\ + (1 - p_{p,l}) \cdot [(T_{on} - 2 \cdot T_{turn}) \cdot (1 - D_{on}) + 2 \cdot T_{turn}] \end{aligned} \tag{2.42}$$

Expected time needed for transmission. To instantiate the constraint in (2.15) to prevent MAC-dependent packet queuing, we give an expression for the expected time needed for a single transmission attempt with LPP.

$$
\begin{aligned}
T_{tx} = &\left[T_{pw,l} + 4 \cdot T_{turn} + T_d + T_{da} \cdot p_l^2 + (T_{out} + T_b) \cdot (1 - p_l^2)\right] \cdot p_{p,l} \\
&+ (T_{on} + T_b) \cdot (1 - p_{p,l}) \qquad (2.43)
\end{aligned}
$$

2.3 System Support

pTunes must tackle several system-level challenges to obtain an efficient runtime operation. This section highlights these challenges and presents the system support we design to meet them. This includes a novel approach for collecting network state information and disseminating new MAC parameters, and the techniques and tools we use to solve the parameter optimization problem efficiently.

2.3.1 Challenges

Minimum disruption. pTunes must reduce the amount of disruption perceived by the application, particularly with respect to application data traffic, to avoid influencing its behavior beyond the adaptation of MAC parameters. This is in itself a major challenge in low-power wireless networks [CKJL09].

Timeliness. Timely collection of accurate network state, computation of optimized MAC parameters, and their reliable and rapid dissemination are fundamental to pTunes. Only this way pTunes can provide MAC operating parameters that do match the current network state. However, it is difficult to perform the above operations in a timely manner, especially when involving resource-constrained devices.

Consistency. pTunes requires consistent snapshots of network state, possibly captured by all nodes at the same time. Otherwise, optimizing MAC parameters based on information different from the actual network conditions may even negatively affect the system performance. Coordinating distributed sensor nodes to achieve consistency is challenging, given their bandwidth and energy constraints.

Energy efficiency. pTunes must meet all the previous challenges while introducing only a limited, possibly predictable, energy overhead at the sensor nodes. To be viable, the overhead of pTunes must not outweigh the gains obtained from adapting the MAC parameters.

2.3.2 Collection and Dissemination

pTunes uses Glossy network floods [FZTS11] to collect network state information and disseminate MAC parameters. In particular, pTunes exploits Glossy's time synchronization service to schedule and execute both operations within short time frames, repeated every *collection period* T_c. Every frame starts with a Glossy flood initiated by the sink, which serves to time-synchronize the nodes and disseminate new MAC parameters. Following the initial flood by the sink, each of the other nodes initiates a flood in turn within exclusive slots, reporting network state for the subsequent trigger decision and parameter optimization.

The collection period T_c can range from a few tens of seconds to several minutes depending on network dynamics and application needs, and represents a trade-off between the energy overhead of pTunes and its responsiveness to changes in the network: a shorter T_c permits more frequent parameter updates but increases the energy consumption of the nodes. The efficiency of Glossy allows us to limit the length of the periodic collection and dissemination frames, thus keeping the energy overhead to a minimum. For instance, we measure on a 44-node testbed an average duration of 5.2 ms for a single flood, and an average radio duty cycle of 0.35 % due to pTunes collection and dissemination for T_c = 1 minute, which reduces to about 0.07 % for T_c = 5 minutes. Given that state-of-the-art low-power MAC protocols exhibit duty cycles of 3–7 % in testbed settings comparable to ours [GFJ+09, DDHC+10], the energy overhead of pTunes is marginal.

An alternative to our approach may be to piggyback network state on application packets and to use a variant of Trickle [LPCS04] to disseminate MAC parameters. We employed this approach at an early stage of this work, but found it inadequate for our purposes. For instance, running Trickle concurrently with data collection increases contention, especially during parameter updates, which degrades application data yield [CKJL09]. Moreover, piggybacking on data packets induces a dependency on the rate and reliability of application traffic. In low-rate applications, it may take a very long time until network state from all nodes becomes available for optimization. Packets may also be generated at different times and experience varying end-to-end delays (e.g., due to contention or routing loops), so the collected network state is likely to be out-of-date and inconsistent. Our approach avoids these problems by temporally decoupling collection and dissemination from application tasks, and by leveraging consistent network state snapshots taken with microsecond accuracy at all nodes independently of application traffic.

In particular, pTunes collects three pieces of network state from each node: (*i*) the node id and the id of the routing parent, to allow pTunes to learn about the current routing tree ($\mathcal{N}$, $\mathcal{M}$, $\mathcal{L}$); (*ii*) the number of packets generated per second F_n, allowing pTunes to determine the traffic volumes; and (*iii*) the ratio $H_{s,l}/H_{t,l}$ of successful and total number of link-layer handshakes over link l to the routing parent. There are two handshakes in X-MAC, strobe/s-ack and data/d-ack; LPP features only the latter (see Figs. 2.3 and 2.4). To account for parent switches and link dynamics, a node maintains counters $H_{s,l}$ and $H_{t,l}$ in a way similar to an exponentially weighted moving average (EWMA). Based on their ratio received from each node and by taking the square root, pTunes obtains estimates of the probability of successful transmission p_l of all links in the current routing tree. The collected information totals 6 bytes per node.

2.3.3 Optimization Tools

Applying the optimization problem in (2.1) to our X-MAC and LPP models in Section 2.2 leads to a mixed-integer nonlinear program (MINLP) with non-convex objective and constraint functions. To solve it efficiently, we use the ECL^{i}PS^e constraint programming system [AW07]. Its high-level programming paradigm allows for a succinct modeling of our optimization problem. We use modules to separate protocol-independent from protocol-dependent code; the latter amounts to about 100 lines for each X-MAC and LPP.

We use the branch-and-bound algorithm coupled with a complete search routine, both provided by the interval constraint (IC) solver of ECL^{i}PS^e. The running time of the optimization depends to a large extent on the size of the search space. To reduce it, we exploit the fact that low-power MAC protocols are commonly implemented using hardware timers. The resolution of these timers determines the maximum granularity required for the timing parameters. We therefore discretize the domains of T_{on} and T_{off} considered for adaptation, letting ECL^{i}PS^e determine values with millisecond granularity. Based on the literature and our own experience, we set the upper bounds of N and T_{off} to 10 retransmissions and 1 s; T_{on} is chosen such that a node listens long enough to overlap with exactly one receiver wake-up in LPP, and with at least one but not more than three strobe transmissions in X-MAC. For these settings and in the scenarios we tested, representative of a large fraction of deployed sensor networks, ECL^{i}PS^e finds optimized MAC parameters within a few tens of seconds on a standard laptop computer. Compared with our current approach, which leverages general-purpose algorithms and off-the-shelf implementations, dedicated solution techniques and implementations are likely to improve significantly on this figure.

2.4 Implementation Details

On the sensor nodes, we deploy Contiki v2.3. We extended the existing X-MAC implementation in Contiki with link-layer retransmissions and an interface that allows to adjust the parameters in (2.16) at runtime. Since the existing LPP implementation suffered from performance problems that could bias our results, we re-implemented LPP within the Contiki stack and extended it in the same way as X-MAC. For data collection we use Contiki Collect, which maintains a tree-based routing topology using the expected number of transmissions (ETX) [DCABM03] as cost metric.

The pTunes control application that runs on the base station is implemented in Java. It retrieves collected network state from the sink, starts the optimization process depending on the trigger decision, and transfers new MAC parameters back to the sink for dissemination.

An important decision for pTunes is when to trigger the parameter optimization. In general, we would like to optimize as often as possible to make the MAC parameters closely match the changing network state. At the same time, we would like to minimize the energy overhead of network state collection and parameter dissemination, while taking into accout that running the solver takes time. Therefore, pTunes includes three basic optimization triggers to decide when to start the optimization process. Nevertheless, pTunes users can implement their own application-specific triggers against a set of basic interfaces we provide.

Among the triggers pTunes includes, *TimedTrigger* optimizes periodically, where the period is typically a multiple of the collection period T_c. In this way, a TimedTrigger can launch the solver immediately after the collection of network state, and pTunes can flood the new MAC parameters in the next dissemination phase. Depending on application-specific requirements and performance goals, users may also want to combine a TimedTrigger with one of the following two triggers.

A *ConstraintTrigger* uses themodel to estimate the current network performance based on the collected network state, and launches the solver only if any of the constraints in (2.1) is violated. A ConstraintTrigger can be implemented to tolerate short-term violations of a constraint, or a violation within some threshold around the constraint. Alternatively, a *NetworkStateTrigger* can infer directly from the network state if the MAC parameters should be updated. For example, a NetworkStateTrigger may fire if it detects a significant increase in traffic volume, thus starting the solver to find MAC parameters that provide higher bandwidth.

2.5 Experimental Results

This section uses measurements from a 44-node testbed to study both the effectiveness of pTunes and the interactions of MAC parameter adaptation with the routing protocol. Our key findings are the following:

- Validation against real measurements shows that our performance models of X-MAC and LPP are highly accurate.
- pTunes automatically determines MAC parameters that provide higher bandwidth in response to an increase in the traffic load. This avoids the occurrence of packet queuing until the network capacity attainable in our specific setting is fully exhausted.
- In the scenarios we tested, pTunes achieves up to three-fold lifetime gains over static MAC parameters that are carefully optimized for the peak traffic loads.
- In a scenario where the packet rates vary across nodes and fluctuate over time, pTunes satisfies given end-to-end latency and reliability requirements at peak traffic load while simultaneously prolonging the network lifetime at lower traffic loads.
- During periods of controlled wireless interference, pTunes reduces packet loss by 80 % compared to static MAC parameters that are carefully optimized for the applied traffic load without interference, thus satisfying given end-to-end reliability requirements.
- By adapting the MAC parameters, pTunes helps the routing protocol recover from critical network changes, reducing the number of parent switches and settling quickly on a stable routing topology. This reduces packet loss by 70% in a scenario where multiple nodes that are important for tree routing fail simultaneously.

2.5.1 Setting and Metrics

Testbed. Our testbed spans one floor in an ETH building [LFZ+13b, DBK+07]. Figure 2.5 shows the positions of the 44 TelosB nodes distributed in several offices, passages, and storerooms; two nodes are located outside on the rooftop. The sink is connected to a laptop computer that acts as the base station. Paths between nodes and the sink are between 1 to 5 hops in length. Nodes transmit at the highest power setting of 0 dBm, using channel 26 to limit the interference with co-located Wi-Fi.

Metrics. Our evaluation is based on the metrics defined in Section 2.2.1. To measure network lifetime, we use Contiki's energy profiler to obtain

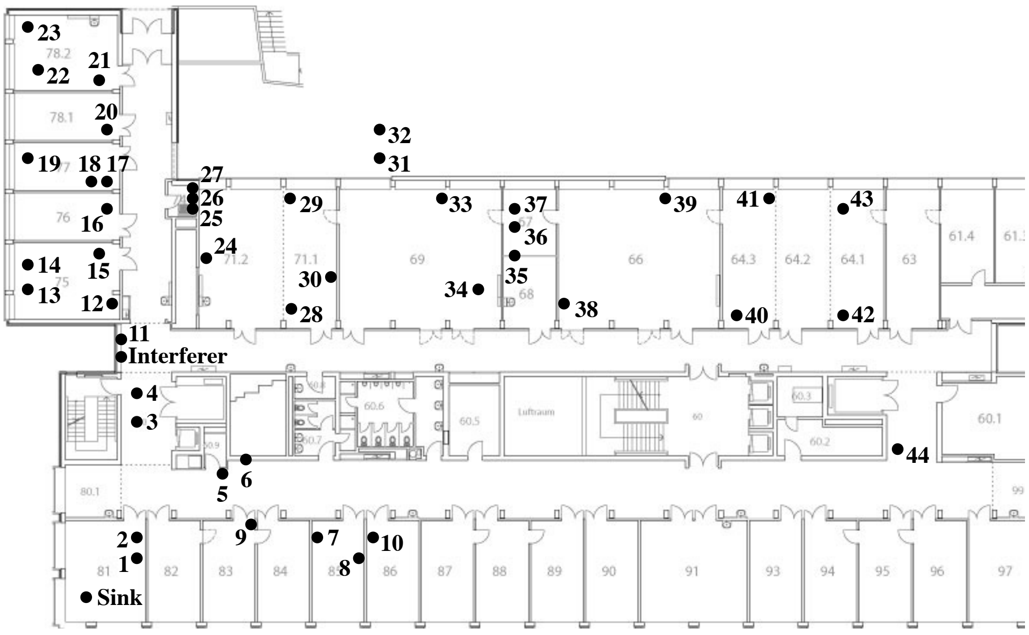


Figure 2.5: Layout of the testbed used to experimentally evaluate pTunes. *Nodes 31 and 32 are located outside on the rooftop; the interferer is only used in Section 2.5.6.*

the fractions of time the radio is in receive, transmit, and idle mode. Then, we compute *projected* node lifetimes using (2.11) and current draws from the CC2420 data sheet, assuming batteries constantly supply 2000 mAh at 3 V. When pTunes is enabled, the measured network lifetime includes the energy overhead of pTunes collection and dissemination, performed every $T_c = 1$ minute in all experiments. We measure end-to-end reliability based on the sequence numbers of data packets received at the sink. To measure end-to-end latency, we exploit Glossy's time synchronization service and timestamp data packets at the source.

Requirements. We consider typical requirements of real-world data collection applications: maximize network lifetime while providing a certain end-to-end reliability [CMP+09, TPS+05]. We also enforce a constraint on end-to-end latency, accounting for applications that require timely delivery [CCD+11].

$$\begin{aligned} &\text{Maximize} && T(\mathbf{c}) \\ &\text{Subject to} && R(\mathbf{c}) \geq 95\,\% \ \text{ and } \ L(\mathbf{c}) \leq 1\,\text{s} \end{aligned} \tag{2.44}$$

pTunes solves (2.44) at runtime to determine optimized MAC parameters. If there exists no solution because either constraint in (2.44) is unsatisfiable (e.g., due to low link qualities), pTunes maximizes R without constraints. This serves to exemplify the capabilities of pTunes; other policies can be implemented based on the optimization triggers we provide.

Table 2.2: Static MAC configurations of X-MAC and LPP optimized for various performance trade-offs and traffic loads given our specific testbed and setup.

	Name	Parameter Values $[T_{on}, T_{off}, N]$	Performance Trade-Off (R, L, T)
X-MAC	S1	[16 ms, 100 ms, 8]	(high, low, low)
	S2	[11 ms, 250 ms, 5]	(medium, medium, medium)
	S3	[6 ms, 500 ms, 2]	(low, high, high)
	S4	[6 ms, 100 ms, 3]	optimized for IPI = 30 s
	S5	[11 ms, 350 ms, 2]	optimized for IPI = 300 s
	S6	[16 ms, 20 ms, 10]	(very high, very low, very low)
LPP	S7	[116 ms, 100 ms, 8]	(high, low, low)
	S8	[266 ms, 250 ms, 5]	(medium, medium, medium)
	S9	[516 ms, 500 ms, 2]	(low, high, high)

Methodology. We compare pTunes with several static MAC configurations optimized for a variety of different traffic loads and application requirements, as listed in Table 2.2. We found these MAC configurations using pTunes and extensive experiments on our testbed. Existing MAC adaptation approaches, on the other hand, consider only per-link and per-node metrics [PFJ10, BYAH06] or focus solely on energy [JBL07, MWZT10, CWW10], rendering the comparison against pTunes purposeless.

2.5.2 Model Validation

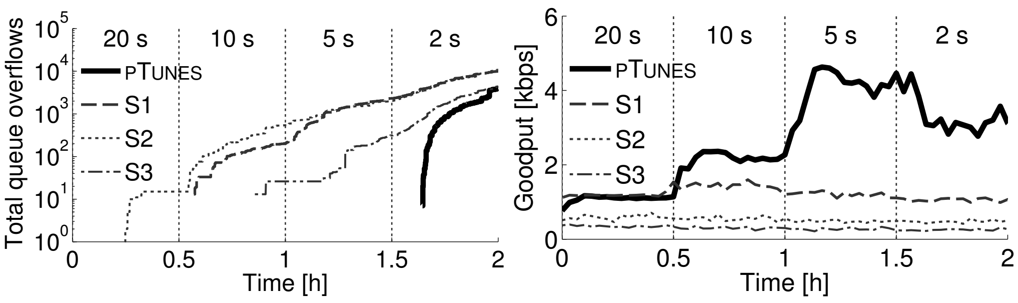
Before evaluating pTunes under traffic fluctuations, wireless interference, and node failures, we validate our models and assumptions from Section 2.2 on real nodes.

Scenario. We run experiments in which we let pTunes periodically estimate the application-level metrics based on the collected network state, and compare the model estimation $e(M_i)$ against the actual measurement $m(M_i)$ by computing the absolute *model error* $\delta(M_i) = m(M_i) - e(M_i)$ for each metric $M_i \in \{R, L, T\}$. Using δ we assess the model accuracy depending on MAC configuration and network state.

To evaluate the dependency on the former, we use three static MAC configurations for each protocol (S1–S3 and S7–S9 in Table 2.2). We also perform one run with pTunes enabled, using a TimedTrigger to adapt the MAC parameters every 10 minute. To evaluate the dependency on network state, in each run we progressively decrease the inter-packet interval (IPI) at all nodes, from 300 s to 180, 60, 30, 20, 10, 5, and 2 s. In this way, we also validate our models against different probabilities of

Table 2.3: Average absolute errors of the network-wide performance model in testbed experiments, with pTunes and six static MAC configurations. *Our X-MAC and LPP models are highly accurate in all metrics.*

	X-MAC				**LPP**			
	S1	**S2**	**S3**	**pTunes**	**S7**	**S8**	**S9**	**pTunes**
$\boldsymbol{\delta(R)}$ **[%]**	-0.68	-0.18	0.09	0.24	4.77	-0.22	0.49	0.41
$\boldsymbol{\delta(L)}$ **[s]**	0.37	0.04	0.18	0.05	-0.12	0.07	0.04	0.08
$\boldsymbol{\delta(T)}$ **[d]**	0.25	0.64	0.65	-0.50	0.37	-0.91	0.96	-0.73



(a) Total number of queue overflows.

(b) Goodput at the sink.

Figure 2.6: Total number of queue overflows across all nodes and goodput at the sink with X-MAC as the traffic increases, using pTunes and three static MAC configurations. pTunes *triples the goodput and avoids the occurrence of local packet queuing until the network capacity is fully exhausted.*

successful transmission p_l: a shorter IPI increases contention and thus lowers the link success rates. We conduct repeatable experiments by enforcing the same static routing topology across all runs.

Results. Table 2.3 lists average model errors in R, L, and T for X-MAC and LPP. We see that both models are highly accurate in all metrics. For example, with pTunes enabled, our LPP models estimate R, L, and T with average absolute errors of 0.41 %, 0.08 s, and -0.73 d. Note that node dwell times, which are included in the measurements but ignored in the model of L, introduce only a negligible error since pTunes aims at avoiding packet queuing, as explained next.

2.5.3 Impact on Bandwidth and Queuing

Based on the experiments above, we study also the impact of the MAC configuration on bandwidth and local packet queuing. To this end, we analyze queuing statistics collected from the nodes and the goodput measured at the sink (application packets carry 69 bytes of data).

Results. Figure 2.6 plots total queue overflows and goodput for X-MAC as the IPI decreases. We can see from Figure 2.6(a) that pTunes avoids queue overflows up to IPI = 2 s, whereas S1–S3 fail to prevent overflows already at longer IPIs. The increasing traffic requires more and more bandwidth, leading to local packet queuing and ultimately to queue overflows when the bandwidth becomes insufficient. Unlike S1–S3, pTunes tolerates such increasing bandwidth demands by automatically adjusting the MAC parameters to provide higher bandwidth. By doing so, pTunes avoids the occurrence of packet queuing until even the MAC parameters providing the highest bandwidth (S6 in Table 2.2), based on the settings and X-MAC implementation we use, are insufficient.

This is also confirmed by looking at the goodput seen by the sink, which is shown in Figure 2.6(b). First, we note that pTunes achieves a more than three-fold increase in goodput over MAC configurations S1–S3 at IPI = 5 s. When queuing occurs also with pTunes at IPI = 2 s, goodput drops from 4.6 kbps to 3.1 kbps, because increased contention leads to more transmission failures and queue overflows. This confirms that the network capacity is fully exhausted at this point. To keep satisfying the requirements in such situations, an application needs to employ higher-layer mechanisms, such as a rate-controlled transport layer that reduces the transmission rate in response to congestion [PG07].

2.5.4 Lifetime Gain

In real deployments, it is common practice to overprovision the MAC parameters based on the highest expected traffic load [KGN09]. The goal is to provide sufficient bandwidth during periods of peak traffic, for example, when an important event causes nodes to temporarily generate more sensor data. However, because such traffic peaks are usually rare and short compared to the total system lifetime, overprovisioning the MAC parameters results in a significant waste of energy resources [LM10]. We now analyze how pTunes helps alleviate this problem.

Scenario. We conduct two experiments in which nodes gradually increase the IPI from 10 s to 20 s, 30 s, 60 s, 3 minute, 5 minute, and 20 minute. In the first experiment, we use pTunes exactly once at the very beginning to determine MAC parameters optimized for the initial IPI of 10 s, and then keep this overprovisioned MAC configuration until the end of the experiment. In the second experiment, we let pTunes adapt the MAC parameters, using a TimedTrigger with a period of 10 minute; pTunes maximizes T subject to $R \geq 95\,\%$ and no constraint on L. We enforce the same static routing topology in both experiments to factor out effects related to routing topology changes, an aspect we consider in Secs. 2.5.5

Table 2.4: Lifetime gains achieved by pTunes relative to static MAC parameters that are carefully optimized for peak traffic load, depending on baseline traffic load and the fraction of time at peak traffic load. pTunes *achieves up to three-fold lifetime gains in settings with extremely rare traffic peaks and low baseline traffic.*

Fraction of time	**X-MAC**				**LPP**			
at peak traffic	**Baseline IPI [min]**				**Baseline IPI [min]**			
(IPI = 10 s)	**1**	**3**	**5**	**20**	**1**	**3**	**5**	**20**
75%	1.05	1.17	1.24	1.43	1.14	1.27	1.35	1.57
50%	1.14	1.36	1.50	1.88	1.24	1.50	1.65	2.08
25%	1.21	1.55	1.75	2.33	1.33	1.72	1.95	2.60
0%	1.29	1.74	2.01	2.77	1.42	1.94	2.24	3.11

and 2.5.7. We then compute the *lifetime gain* as the ratio between the measured network lifetime with and without pTunes.

Results. Table 2.4 lists lifetime gains for X-MAC and LPP, *including* the energy overhead of pTunes collection and dissemination phases. We see that the lifetime gain achieved by pTunes increases as (*i*) the system spends less time at peak traffic (75–0 % from top to bottom), and (*ii*) the difference between the shortest, overprovisioned IPI of 10 s and the longest, baseline IPI increases (1–20 minute from left to right). For instance, for a baseline traffic at IPI = 20 minute and extremely rare traffic peaks at IPI = 10 s, the lifetime gain is close to 2.77 for X-MAC and close to 3.11 for LPP compared to static MAC parameters overprovisioned for peak traffic.

The above experimental results reveal that pTunes enables significant lifetime gains, not least due to its energy-efficient system support (see Section 2.3). The following sections examine how pTunes trades possible gains in network lifetime for satisfying end-to-end reliability and latency requirements under varying network conditions.

2.5.5 Adaptation to Changes in Traffic Load

Traffic fluctuations are characteristic of many sensor network applications, where the data rate often depends on time-varying external stimuli. The following experiments investigate the benefits pTunes brings to these applications.

Scenario. All nodes send packets with IPI = 5 minute for 5 hours. However, during two periods of 30 minute each, two clusters of 10 and 5 spatially close nodes (14–23 and 40–44 in Figure 2.5) send packets with IPI = 10 s, emulating the detection of an important event that deserves reporting more sensor data.

We run three experiments with X-MAC and dynamic routing topologies using Contiki Collect. In the first two experiments, we use static MAC configurations S1 and S5: S1 provides high bandwidth when nodes send more packets, and S5 extends network lifetime at normal traffic (see Table 2.2). In the third experiment, we let pTunes adapt the MAC parameters according to (2.44). We couple a TimedTrigger with a NetworkStateTrigger as follows. When nodes transmit at low rate, the TimedTrigger starts the solver every 10 minute. As soon as the NetworkStateTrigger detects the beginning of a traffic peak, it starts the solver immediately and adapts the period of the TimedTrigger to 5 minute, setting it back to 10 minute at the end of a peak. In this way, pTunes reacts promptly to traffic changes, and adapts more frequently during traffic peaks when nodes report important sensor data.

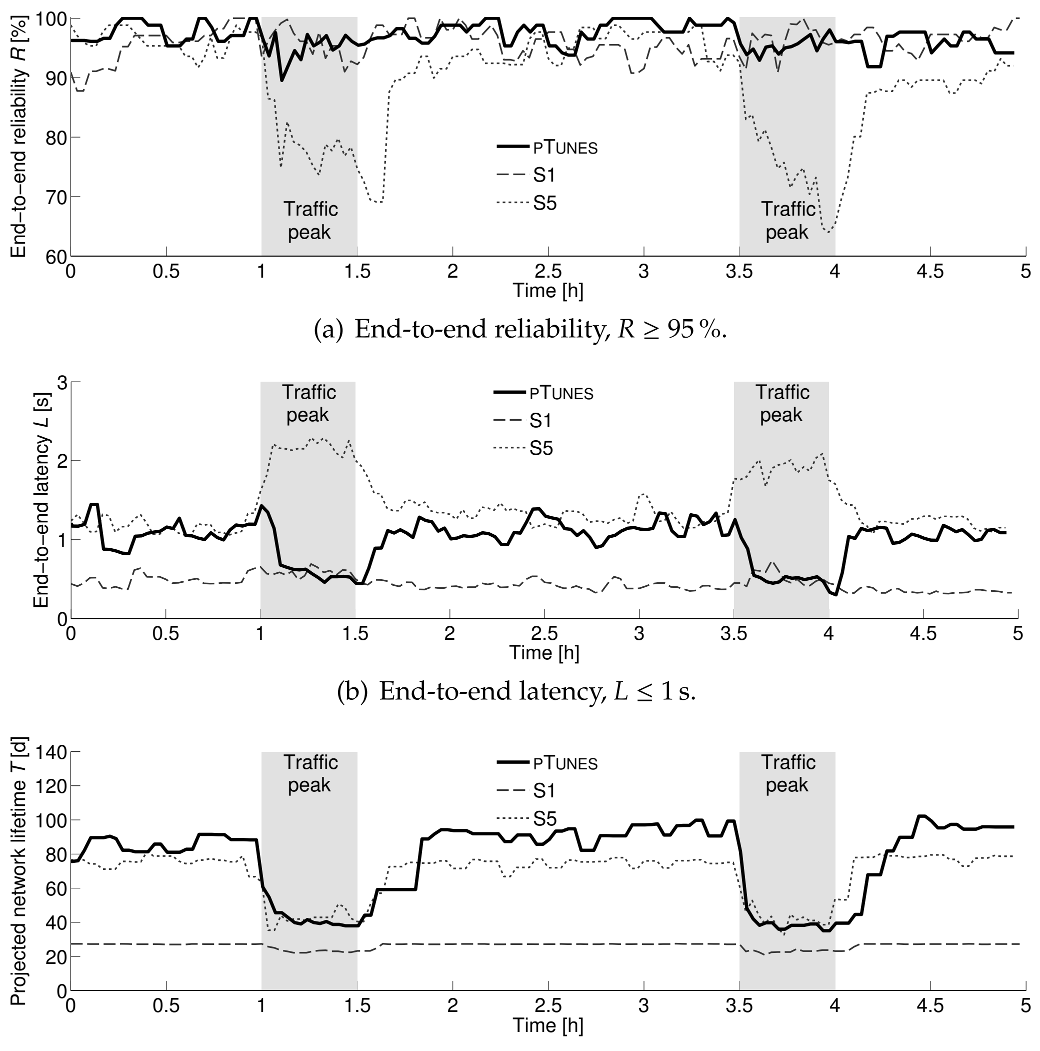
Results. Figure 2.7 plots performance over time in the three experiments. We see that S5 approximately satisfies the reliability and latency requirements when nodes send at low rate, achieving also a high projected network lifetime. However, as soon as the two node clusters start transmitting at high rate, reliability drops significantly below 75 %. This is because S5 does not provide sufficient bandwidth, leading to high contention and ultimately to packet loss. Similarly, S5 violates the latency requirement during traffic peaks, making L exceed 2 s due to queuing and retransmission delays. S1, instead, provides sufficient bandwidth and satisfies the end-to-end requirements. However, network lifetime is always below 30 d: the higher bandwidth comes at a huge energy cost, paid also when a lower bandwidth would suffice.

By contrast, pTunes satisfies the end-to-end requirements under high and low rate. Moreover, when nodes transmit at low rate, the projected network lifetime increases up to 90 d. By adapting the MAC parameters, pTunes always provides a bandwidth sufficient to satisfy the end-to-end requirements without sacrificing lifetime unnecessarily: at the beginning of a traffic peak, pTunes reduces T_{off} from about 300 ms to 120 ms (and slightly adapts T_{on} and N), which explains why reliability stays up and latency is halved. Static MAC configurations lack this flexibility; they can only be optimized for a specific workload and thus fail to trade the performance metrics as the traffic conditions change.

2.5.6 Adaptation to Changes in Link Quality

Unpredictable changes in link quality are characteristic of low-power wireless [ZG03]. Adapting the MAC parameters to these changes is important but non-trivial, as we show next.

Scenario. We use the technique by Boano et al. to generate controllable



(a) End-to-end reliability, $R \geq 95\,\%$.

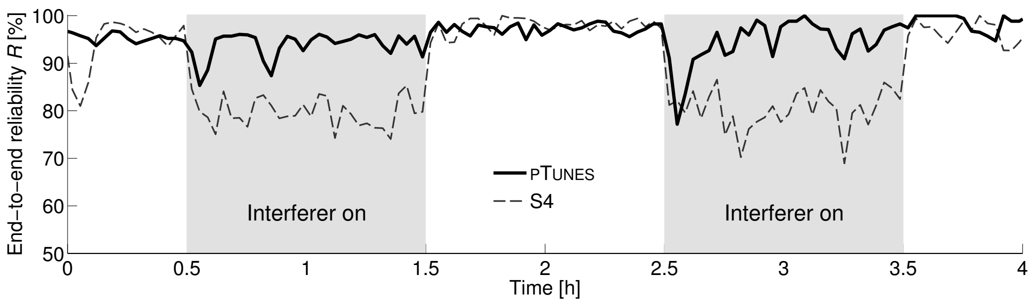
(b) End-to-end latency, $L \leq 1\,\text{s}$.

(c) Projected network lifetime.

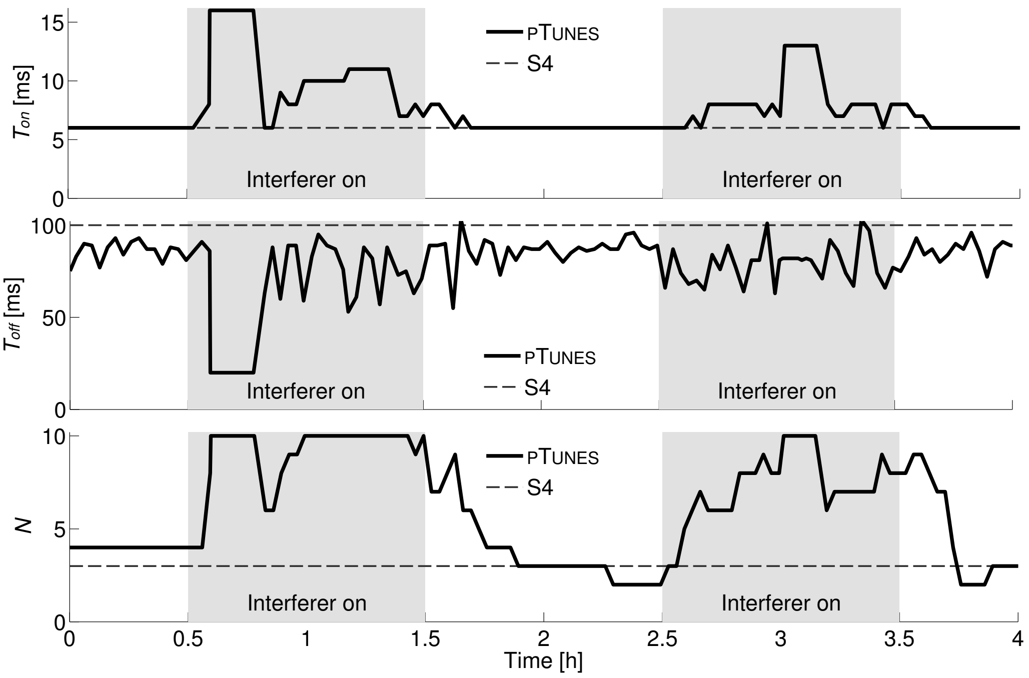
Figure 2.7: Performance of pTunes against two static MAC configurations as the traffic load changes. pTunes *satisfies the end-to-end requirements at high traffic while extending network lifetime at low traffic. Fixed MAC parameters optimized for a specific traffic load fail to meet the application requirements as the traffic conditions change.*

interference patterns [BVT+10], making the link quality fluctuate in a repeatable manner. To this end, we deploy an additional interferer node in a position where it affects the communication links of at least one fourth of the nodes in our testbed, as shown in Figure 2.5. When active, the interferer transmits a modulated carrier on channel 26 for 1 ms at the highest power setting. Then, it sets the radio to idle mode for 10 ms before transmitting the next carrier.

All nodes generate packets with IPI = 30 s for 4 hours. The interferer is active during two periods of 1 hours each. In a first experiment, we use static MAC configuration S4, optimized for IPI = 30 s (see Table 2.2).



(a) End-to-end reliability, $R \geq 95\,\%$.



(b) Trace of X-MAC parameters.

Figure 2.8: End-to-end reliability and trace of X-MAC parameters with pTunes as the link qualities in the network change. pTunes *reduces packet loss by 80 % during periods of controlled wireless interference in comparison with static MAC parameters optimized for the applied traffic load without interference.*

We enable pTunes in a second experiment, using a TimedTrigger with a period of 1 minute to adapt the MAC parameters according to (2.44). We deliberately enforce a static routing tree to separate effects related to link quality changes from those related to topology changes. We investigate the latter in detail in Section 2.5.7.

Results. Figure 2.8 shows end-to-end reliability and the trace of X-MAC parameters. Looking at Figure 2.8(a), we see that S4 and pTunes satisfy the reliability requirement when the interferer is off. When the interferer is on, reliability starts to drop below 95 %. However, as soon as pTunes collects network state, it detects a decrease in link quality and adapts the X-MAC parameters accordingly. In particular, as shown in Figure 2.8(b), pTunes increases N from 3 or 4 to values between 6 and 10. T_{on} is also increased (from 6 ms to 10–16 ms) to further help satisfy the reliability requirement. Moreover, pTunes decreases T_{off} (from 100 ms to 20–90 ms) to provide more bandwidth and combat increased channel contention, which is a consequence of numerous retransmission attempts over low-quality links. Indeed, these low-quality links make (2.44) temporarily unsatisfiable (while $T_{on} = 16$ ms in the first interference phase), triggering pTunes to instead maximize R as explained in Section 2.5.1. As a result of these decisions, pTunes achieves an average end-to-end reliability of 95.4 % also in presence of interference.

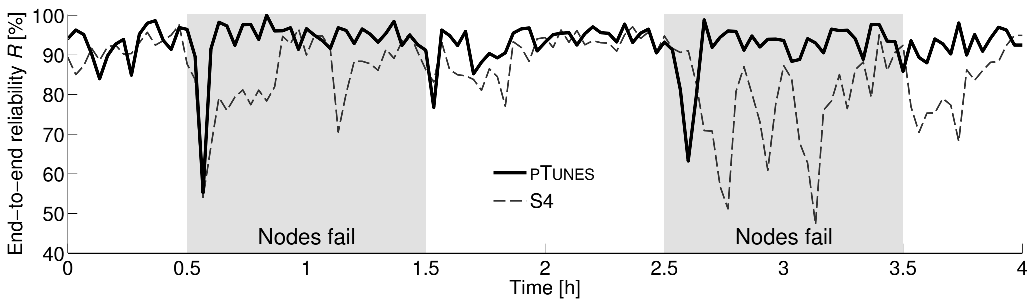
S4, instead, fails to meet the reliability requirement when the interferer is active: reliability ranges between 70 % and 80 %, and never recovers while the interferer is on. In total, 2252 packets are lost with S4 during interference. pTunes reduces this number to 418—a reduction of more than 80 %.

2.5.7 Interaction with Routing

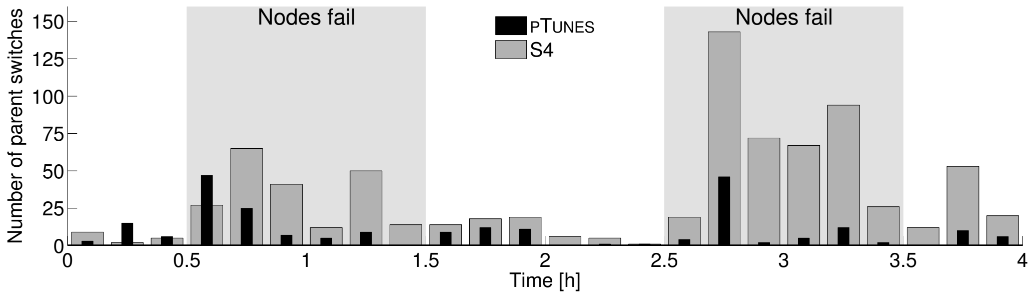
Several studies emphasize the significance of cross-layer interactions to the overall system performance [DPR00]. We study this aspect between best-effort tree routing and parameter adaptation of an underlying low-power MAC protocol with pTunes. To do so, during each of the following experiments, we temporarily remove multiple core routing nodes important for forwarding packets. In this way, we emulate node failures, which are common in deployed systems [BGH+09], and force the routing protocol to find new routes.

Scenario. We run two 4-hour experiments with Contiki Collect and X-MAC. After 30 minute, we turn off eight nodes within the sink's neighborhood that forward most packets in the network (1–8 in Figure 2.5). We turn them on again after 60 minute, and repeat the on-off pattern after 1 hours. Nodes generate packets with IPI = 30 s. In the first experiment, we use static MAC configuration S4, optimized for this traffic load (see Table 2.2). In the second experiment, we enable pTunes and use a TimedTrigger to solve (2.44) every minute.

Results. Figure 2.9(a) shows end-to-end reliability over time, accounting for packets from nodes that are currently turned on. During the first 30 minute, both S4 and pTunes satisfy the reliability requirement. However, when nodes are removed, reliability starts to drop below 70 %. Many packets are indeed lost since children of removed nodes fail to transmit packets: the routing protocol needs to find new routes.



(a) End-to-end reliability, $R \geq 95\,\%$.



(b) Distribution of parent switches.

Figure 2.9: End-to-end reliability and distribution of parent switches when eight core routing nodes fail simultaneously. pTunes *helps the routing protocol recover from node failures by settling quickly on a stable routing topology, thus reducing packet loss by 70 % compared with static MAC parameters optimized for the applied traffic load.*

We see from Figure 2.9(a) that end-to-end reliability recovers much faster when pTunes is enabled. During the two periods when eight nodes are removed, S4 fails to deliver in total 2673 packets from the remaining 35 nodes. pTunes reduces this number to 813—a reduction of 70 %.

To further investigate this behavior, we plot in Figure 2.9(b) the distribution of parent switches. pTunes reduces the total number of parent switches compared to S4 (from 631 to 165), and shifts them to the beginning of the periods in which nodes are removed. At this point, pTunes quickly realizes a significant drop in link quality, reported by nodes whose parent disappeared. pTunes thus increases T_{on} and N to improve reliability, and decreases T_{off} to provide more bandwidth for retransmissions and route discovery.

As a result of increasing the maximum number of retransmissions per packet N, transmission attempts of nodes with a dead parent fail with a higher number of retries. This causes the corresponding ETX values to drop more severely than with S4 (which has a lower N), and so nodes switch much faster to a new parent. Moreover, the MAC parameters provided by pTunes help deliver packets over the remaining links. Delivering more packets also enables the routing protocol to quickly detect route inconsistencies and eventually settle on a stable topology. As the topology stabilizes, pTunes gradually relaxes the MAC parameters (reduce T_{on} and N, increase T_{off}) to extend network lifetime.

These results demonstrate that, by adapting the MAC parameters, pTunes helps the routing protocol recover faster from critical network changes. Protocols like CTP [GFJ+09] and Arbutus [PH10] also utilize feedback from unicast transmissions to compute the ETX. In addition, CTP uses data path validation to detect possible loops based on ETX values embedded in data packets [GFJ+09]. Our findings with Contiki Collect, which uses similar techniques, suggest that these protocols could also benefit from pTunes.

Additionally, the results demonstrate the advantage of decoupling network state collection from application packet routing, as we argue in Section 2.3.2. As long as the network remains connected, Glossy provides up-to-date network state to the base station with very high reliability [FZTS11]. Changes in the routing tree do no affect network flooding: information about faulty links is collected even when the routing protocol fails to deliver packets from nodes whose parent died, allowing pTunes to react promptly and thus effectively.

2.6 Discussion

Designing a MAC adaptation framework involves striking a balance between goals typically at odds with each other. We discuss some of the trade-offs we make in pTunes and the implications of our decisions.

Feasibility *vs.* scalability. We adopt a centralized approach rather than a likely more scalable distributed solution; in return for this, pTunes allows users to express their requirements in terms of network-wide metrics, which better reflect the way domain experts are used to state performance objectives compared to per-node or per-link metrics. In fact, distributing the tasks of collecting global information, computing MAC parameters optimized for network-wide objectives, and coordinating the consistent installation of new parameters would hardly be feasible, if at all, on resource-constrained devices. Instead, pTunes exploits the better resources of a central base station, which is already present in many real deployments [RC08], and achieves simplicity of in-network functionality by moving most of its intelligence from the nodes into the base station.

Flexibility *vs.* optimality. We focus on existing MAC protocols rather than on the design or adaptation of cross-layer solutions (e.g., coupling link and network layer) which may, in principle, achieve better performance; in return for this, pTunes allows system designers to choose the MAC and routing protocol independently from existing code bases. In comparison, cross-layer solutions tend to enjoy little generality and flexibility, as they are often designed for very specific scenarios (e.g., periodic, low-rate data collection [BvRW07]).

Robustness *vs.* optimality. We determine network-wide parameters rather than per-node parameters, which may better match the current role of a node in the routing tree (e.g., with respect to traffic load); in return for this, the parameters pTunes provides are much more robust to changes in the routing topology. It is not unlikely that, even in the most benign environment, slight variations in the link qualities trigger drastic changes in the routing topology. For instance, Ceriotti et al. observe that nodes serving many children suddenly become leaves in the routing tree [CMP+09]. In such a case, per-node MAC parameters become inappropriate and must be quickly updated. Similar situations can happen frequently, even several times per minute [GGL10], which would render per-node parameter adaptation impractical.

As a consequence of the design decisions above, pTunes represents one particular point in a multi-dimensional design space. Corresponding to this point is a large fraction of deployed low-power wireless networks comprising tens of nodes, leveraging protocols such as X-MAC and LPP, and yet failing to meet the application requirements often due to communication issues ultimately related to inadequate MAC parameter choices and lack of adaptiveness [RC08, KGN09]. pTunes is directly and immediately applicable in these settings.

2.7 Related Work

pTunes uses a model to predict how changes in the MAC parameters affect the network-wide performance given the current network state. Based on iterative runtime optimization, it selects MAC parameters such that the predicted performance satisfies the application requirements. This approach is similar to the concept of model predictive control (MPC) [GPM89], with the differences that pTunes computes only the next step of the control law and uses no information about past control steps or measured system responses.

Several recent systems incorporate centralized control in their design, much like pTunes does. For example, Koala implements a network-wide routing control plane, where the base station computes end-to-end paths used for packet forwarding [MELT08]. RACNet uses centralized token passing to sequence data downloads [LLL+09]. In RCRT, the sink detects congestion and adapts the rates of individual sources [PG07]. PIP determines schedule and channel assignment for each flow centrally at the base station [RCBG10]. Like pTunes, these systems exploit global knowledge and ample resources of the base station to achieve high performance and manageability.

Looking at the large body of prior work on adaptive low-power MAC protocols, we find solutions embedding adaptivity or separating adaptivity from the protocol operation.

In the former category, for instance, Woo and Culler propose an adaptive rate control mechanism, where nodes inject more packets if previous attempts were successful and fewer packets if they failed [WC01]. Van Dam and Langendoen introduce an adaptive listen period in T-MAC [DL03] to overcome the drawbacks of the fixed duty cycle of S-MAC [YHE02]. El-Hoiydi and Decotignie adapt radio wake-ups in WiseMAC to shorten the LPL preamble [EHD04]. More recently, Hurni and Braun propose MaxMAC, which schedules additional X-MAC wake-ups at medium traffic and switches to pure CSMA at high traffic [HB10]. Such hard-coded adaptivity mechanisms can be highly effective in specific scenarios, but lack general applicability and bear no direct connection to the high-level application demands. pTunes is more general by adding parameter adaptation atop existing MAC protocols, thus leveraging available implementations, and by explicitly incorporating user-provided application requirements.

Polastre et al. instead separate adaptivity from the protocol operation and present a model of node lifetime for B-MAC [PHC04]. Jurdak et al. use this model to dynamically recompute check interval and preamble length, showing substantial energy savings [JBL07]. Buettner et al. demonstrate energy savings in X-MAC by adapting the wake-up interval to traffic load for one sender-receiver pair [BYAH06]. Meier et al. [MWZT10] and Challen et al. [CWW10] extend network lifetime by adjusting the wake-up interval to traffic load in a static routing tree. Park et al. present numerical results that indicate the potential of adaptation policies for IEEE 802.15.4 MAC protocols, based on per-link and per-node metrics [PFJ10]. pTunes builds on these foundations but extends them in several ways. First, pTunes considers multiple network-wide metrics and adapts multiple MAC parameters. Second, our modeling is more realistic by accounting for packet loss and ARQ mechanisms, and more flexible by isolating protocol-dependent from protocol-independent functionality. Third, we evaluate pTunes in real-world scenarios, including dynamic routing trees, wireless interference, and node failures.

2.8 Summary

This chapter presented pTunes, a novel framework that provides runtime parameter adaptation for low-power MAC protocols. pTunes automatically translates an application's end-to-end performance requirements into MAC parameters that meet these requirements and achieve very good performance across a variety of scenarios, ranging from low traffic to high traffic, from good links to bad links, and wireless interference to node failures. pTunes thus greatly aids in meeting soft requirements of real-world low-power wireless applications by eliminating the need for time-consuming, and yet error-prone, manual MAC configuration before every single deployment and when the network conditions change.

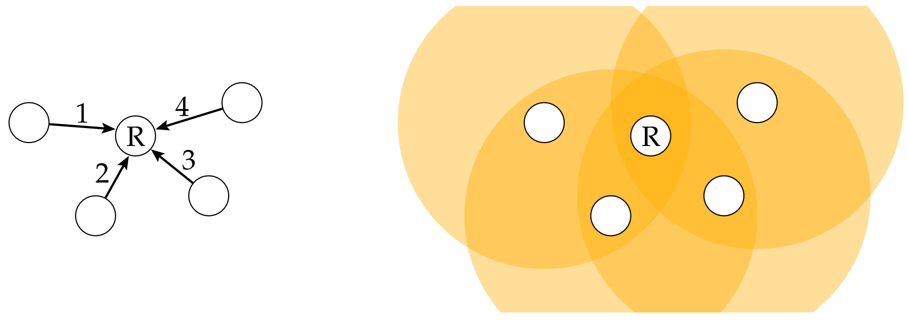
3

Modeling Protocols Based on Synchronous Transmissions

Low-power wireless networks facilitate advanced CPS applications that use wirelessly interconnected sensors and actuators to monitor and act on the physical world, such as environmental control, assisted living, and intelligent transportation [Lee08]. Effectively employing low-power wireless in these applications requires a thorough understanding of the behavior of the protocols that power the network operation. For example, the ability to estimate the energy consumption is crucial to self-sustaining systems based on energy scavenging [MTBB10], and certain guarantees on packet delivery are key to dependable wireless automation [BJ87].

Unfortunately, the current literature falls short in modeling multi-hop low-power wireless protocols. Two aspects concur:

- Low-power wireless transmissions are subject to a number of unpredictable environmental factors, including wireless interference, presence of obstacles and persons, and temperature and humidity changes [HRV+13]. As a result, low-power wireless links suffer from *unpredictable packet loss* that varies in time and space [SDTL10]. This, in combination with failure-prone devices (e.g., due to battery depletion or damage), makes the *network topology highly dynamic*.
- To tame this unpredictability, existing multi-hop communication protocols gather substantial information about the network state, such as link quality estimates [GFJ+09] and the filling levels of packet queues [RGGP06]. Protocols use this information, for example, to form multi-hop routing paths [GFJ+09] and to adapt packet



(a) Link-based transmissions. (b) Synchronous transmissions.

Figure 3.1: Link-based transmissions (LT) versus synchronous transmissions (ST). *Using ST, multiple nodes transmit simultaneously toward the same receiver R, as opposed to pairwise LT 1, 2, 3, and 4 from each sender to R.*

transmission rates [RGGP06]. However, the network state must be *updated at runtime* against the topology dynamics.. For scalability reasons, the network state is often *distributed* across the nodes, which operate *concurrently* with *little or no coordination*.

These reasons render multi-hop low-power wireless protocols intricate and difficult to model [GB12]. As a result, existing models often stop at the link layer, achieving model errors in the range of 2–7 % in real experiments (see Chapter 2). Only a few attempts have been made to model also higher-layer functionality [GB12, BSB+12, GCB08, YZDPHg11]; however, validation of these models is limited to numerical simulations, which lack precisely those real-world dynamics that complicate the modeling.

A new breed of communication protocols has emerged over the past few years that utilize *synchronous transmissions (ST)* [LW09, DDHC+10, FZTS11, WHM+12, LFZ13a, FZMT12, DCL13, CCT+13]. As illustrated in Figure 3.1, unlike single transmissions over sender-receiver links in (a), using ST *multiple* nodes transmit *simultaneously* towards the same receiver in (b). Because of two physical-layer phenomena of low-power wireless communications, constructive baseband interference [DDHC+10] and capture effects [LW09], ST vastly improve the one-hop packet reliability compared with *link-based transmissions (LT)* [DDHC+10].

As we further discuss in Section 3.1, the salient features of ST enable multi-hop communication protocols that require very little network state and outperform LT-based protocols. The open question is whether ST also simplify the accurate modeling of these protocols. To answer this question, this chapter puts forward two key contributions:

1. We investigate in Section 3.2 to what extent the *Bernoulli assumption* applies to ST. The assumption stipulates that subsequent packet receptions and losses at a receiver adhere to a sequence of i.i.d.

Bernoulli trials. Models of communication protocols often make this assumption to simplify the specification [PHC04, ZFM+12], but prior work suggests that this is often invalid for LT [CWPE05, SDTL10]. Up to now, nothing is known about ST in this regard. By studying a specific flavor of ST, *Glossy* network floods [FZTS11], through experiments on a 139-node testbed, we show that the Bernoulli assumption is largely valid for ST, and way more than for LT.

2. We build upon these findings to demonstrate that modeling an ST-based protocol is in fact simpler and yields significantly higher accuracy than models of LT-based protocols. We do so by considering LWB [FZMT12], a representative protocol, described in Section 3.3, of a growing number of solutions [WHM+12, LFZ13a, DCL13, CCT+13] that build upon Glossy. Specifically, we present in Section 3.4 sufficient conditions for providing probabilistic guarantees on LWB's end-to-end packet reliability, and in Section 3.5 a discrete-time Markov chain (DTMC) model to estimate LWB's expected long-term energy consumption. Results from our validation based on real-world experiments in Section 3.6 indicate that the end-to-end reliability guarantees are correctly matched, and that the estimates of the energy model are within 0.25 % of the real measurements. This error margin is unparalleled in the low-power wireless literature we are aware of.

3.1 Background and Related Work

This chapter builds on recent work in low-power wireless communications. In this section, we provide the necessary background on multi-hop protocols exploiting different flavors of ST, contrast these with the existing literature on LT, and review related modeling efforts. We conclude with an outlook on how this chapter fills the gaps in the current literature.

3.1.1 Synchronous Transmissions

Little work exists to deeply understand the behavior of ST in low-power wireless. For example, Son et al. [SKH06] conduct an experimental study of the *capture effect*, a physical layer phenomenon that allows a receiver to correctly decode a packet despite interference from other transmitters. In low-power wireless, this is typically due to *power capture*, which occurs when the received signal from a node is 3 dB stronger than the sum of the signals from all other nodes [SKH06]. Several protocols exploit the capture effect, for example, to implement fast network flooding [LW09] and efficient all-to-all communication [LFZ13a].

Precisely overlapping transmissions of *identical* packets enable another phenomenon in low-power wireless: *constructive baseband interference* of IEEE 802.15.4 symbols. This enables a receiver to correctly decode the packet also in the absence of capture effects, significantly boosting the transmission reliability. Using resource-constrained devices, however, the required timing accuracy of ST is difficult to achieve. One way to address this challenge is by using hardware-generated acknowledgments, a mechanism that has been employed to better resolve contention in low-power MAC protocols [DDHC+10, CT11].

Glossy, instead, uses a careful software design to make ST of the same packet precisely overlap, thus taking advantage of both constructive baseband interference and capture effects for efficient network flooding and time synchronization with microsecond accuracy [FZTS11]. Several protocols extend and improve Glossy, for example, in dense networks [WHM+12], for distributing large data objects [DCL13], for point-to-point communication [CCT+13], and for in-network processing and all-to-all data sharing [LFZ13a]. LWB, which we use to examine the impact of ST on modeling multi-hop protocols, efficiently supports multiple traffic patterns by globally scheduling Glossy floods [FZMT12].

3.1.2 Link-based Transmissions

In contrast to ST, a large body of work exists on understanding the behavior of LT [BKM+12]. Srinivasan et al. [SDTL10], for example, conduct an empirical study of IEEE 802.15.4 transmissions to provide guidelines for fine-grained design decisions such as the scheduling of link-layer packet retransmissions. The β factor [SKAL08] measures the link burstiness over time, which may be used by a protocol to determine how long to pause after a transmission failure to prevent unnecessary retransmissions. Cerpa et al. [CWPE05] examine both short- and long-term temporal aspects to improve simulation models and for enhancing point-to-point routing. Dually, the κ factor [SJC+10] measures the degree of correlation of packet receptions across different receivers—hence exploring LT's spatial diversity—which can possibly be used to design better opportunistic routing and network coding schemes.

Despite the significant knowledge about LT, obtaining full-fledged models of LT-based multi-hop protocols is very difficult [GB12]. Several attempts stop at the MAC layer, where distributed interactions span only one hop and hence reasoning is still manageable. For example, pTunes provides runtime tuning of MAC parameters based on application-level performance goals, leveraging MAC protocol models (see Chapter 2). Similarly, Polastre et al. [PHC04] present a model of node lifetime for

B-MAC, and Buettner et al. [BYAH06] model reliability and energy in X-MAC. Gribaudo et al. [GCB08] use interacting Markovian agents to model a generic sender-initiated low-power MAC protocol [ZFM+12]; they also acknowledge that the opportunistic operation of this class of protocols greatly complicates the modeling using standard techniques.

The dynamics of the network topology render the modeling of higher-layer functionality, where interactions typically extend across multiple hops, very complex. As a result, accurate models of the end-to-end or network-wide performance are largely missing. Some exceptions model the Collection Tree Protocol (CTP) [GFJ+09] to improve its performance in industrial scenarios [YZDPHg11], analyze swarm intelligence algorithms for sensor networks based on the Markovian agent model [BSB+12], apply diffusion approximation techniques to estimate the end-to-end packet travel times assuming opportunistic packet forwarding rules [Gel07], or model generic multi-hop functionality through population continuous-time Markov chains [GB12]. Nevertheless, the validation of these models is limited to numerical simulations, which lack precisely those real-world dynamics of low-power wireless links that make accurate protocol modeling so complex and difficult in the first place.

3.1.3 Outlook

Motivated by the lack of a deeper understanding of ST, in the remainder of this chapter we provide a thorough account on the behavior of ST and its impact on the modeling of emerging ST-based multi-hop protocols. To this end, we start by analyzing in Section 3.2 to what extent a key, yet sometimes illegitimate assumption in modeling low-power wireless protocols applies to ST. We base this study upon Glossy's specific incarnation of ST [FZTS11], because it serves as the communication primitive for a growing class of multi-hop protocols [WHM+12, FZMT12, DCL13, CCT+13, LFZ13a]. We then apply the corresponding findings while closely examining LWB [FZMT12], one specific such protocol we illustrate in Section 4.2 that exceeds the performance, reliability, and versatility of prior LT-based protocols. In doing so, we analyze end-to-end packet reliability in Section 3.4 and energy consumption in Section 3.5—two key performance indicators in low-power wireless [GFJ+09].

3.2 Bernoulli Assumption

Because wireless networks are very complex, researchers make simplifying assumptions about their behavior when reasoning about a protocol. One common assumption is the *Bernoulli assumption* [SDTL10]. Let the reception of packets sent in a sequence be a random event with success or failure as the only possible outcomes. The Bernoulli assumption stipulates that a receiver observes a sequence of i.i.d. Bernoulli trials. In practical terms, success means a packet is received (with probability p), and failure means a packet is lost (with probability $1 - p$).

However, several studies have shown that the Bernoulli assumption is not always valid in low-power wireless, because links have temporally correlated receptions and losses when they occur close in time (i.e., on the order of a few tens of milliseconds) [CWPE05, SDTL10]. In this section, we show empirically that the Bernoulli assumption is (*i*) in fact highly valid for ST in Glossy, and (*ii*) more appropriate in Glossy than for LT.

We first describe how we collect large sets of packet reception traces on a real-world sensor network testbed. Next, we discuss our analysis of these traces for weak stationarity, which is a necessary condition for further statistical analysis. We then construct a statistical test for packet reception independence based on the sample autocorrelation metric, and use this test to assess the validity of the Bernoulli assumption in our traces.

3.2.1 Experimental Methodology

We perform 80 hours of packet reception measurements on Indriya, a large testbed of 139 TelosB nodes deployed across three floors in the School of Computing at the National University of Singapore [DCA11]. Indriya provides a mixture of dense and sparse node clusters, as well as realistic interference from the presence of people and co-located Wi-Fi.

We conduct two types of experiments that match the building blocks of multi-hop communication in ST- and LT-based protocols:

1. *ST-Type.* We select 70 nodes that are equally distributed across the three floors on Indriya and let them, one at a time, initiate 50,000 Glossy network floods. According to Glossy's operation, the remaining 138 nodes blindly relay the flooding packet, eventually delivering it to all nodes in the network.

2. *LT-Type.* All 139 nodes available on Indriya broadcast, one at a time, 50,000 packets, while the remaining nodes passively listen. As LT are bound by the transmission range of the broadcasting node, only its one-hop neighbors can receive the packet.

In both types of experiments, packets are 20 bytes long and carry a unique sequence number. The sender transmits at a fixed inter-packet interval (IPI) of 20 ms, which corresponds to the typical minimum interval between consecutive Glossy floods in ST-based protocols [WHM+12, CCT+13, DCL13, FZMT12]. All other nodes record received and lost

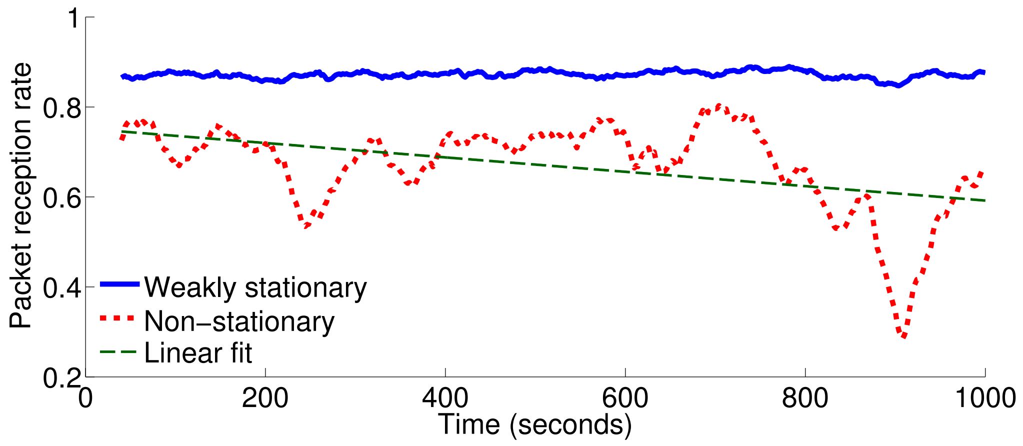


Figure 3.2: Example of a weakly stationary and a non-stationary trace. Packet reception rate is a moving average with a window size of 2,000 packets (40 s).

packets based on the sequence number. We use IEEE 802.15.4 channel 26 to reduce the influence of Wi-Fi interference, whose extent we cannot control and is difficult to assess afterwards [HRV+13]. We repeat both types of experiments for two transmit powers: 0 dBm is the maximum transmit power of a TelosB node, and -15 dBm is the lowest transmit power at which the network on Indriya remains connected. The resulting network diameters are 5 and 11 hops, respectively.

We record in total more than 1,200,000,000 events, grouped into packet reception traces of length n =50,000. We represent every collected trace as a discrete-time binary time series $\{x_i\}_{i=1}^{n}$, where x_i is 1 if the i-th packet was received and 0 if it was lost. This time series representation forms the basis for our statistical analysis.

3.2.2 Weak Stationarity

A necessary condition for well-founded statistical analyses of time series is *weak stationarity* [BD91]. A weakly stationary time series has constant mean, constant variance, and the autocovariance between two values depends only on the time interval between those values. We investigate whether our traces conform to these criteria based on the *packet reception rate* (PRR), computed on a trace as a moving average of the fraction of received packets over a window of 2,000 packets (40 s).

Visual inspection of our traces reveals obvious violations of these criteria. For example, Figure 3.2 shows the PRR for a stationary and a non-stationary trace. The latter has several abrupt changes and a significant trend in the mean, as evident from the linear fit. To avoid biases in our

Table 3.1: Number of non-stationary and weakly stationary traces.

Type	Transmit power	Total	Non-stationary	Weakly stationary
ST	0 dBm	9660	47	9613
ST	-15 dBm	9660	256	9404
LT	0 dBm	4189	1418	2771
LT	-15 dBm	1777	588	1189

analysis, we need to identify and exclude such non-stationary traces.

While there is a number of formal tests for stationarity, they often fail in practice due to their inability to detect general non-stationarity [Det13]. Thus, similar to [YMKT99], we apply two empirical tests to identify non-stationary traces. To test for trends in the mean, we compute a linear fit using ordinary least squares and declare a trace as non-stationary if the PRR changes by 0.015 or more over the entire trace of 50,000 packets (1,000 s). Then, we test for non-constant variance by checking whether the PRR decreases or rises by more than 0.05 within a window of 2,000 packets (40 s), which we interpret as an indication of non-stationarity. Table 3.1 summarizes the sets of traces before and after applying the two empirical tests for non-stationarity; we removed about 9 % of non-stationary traces.

3.2.3 Validating the Bernoulli Assumption

To confirm or refute the Bernoulli assumption for a given trace, we use the *sample autocorrelation*, which measures the linear dependence between values of a weakly stationary time series as a function of the interval (lag) between those values. As we explain below, the Bernoulli assumption is valid if the values in the time series are independent already at lag 1.

For a discrete-time binary time series $\{x_i\}_{i=1}^{n}$ of length n, the sample autocorrelation $\hat{\rho}$ at lag $\tau = 1, 2, \ldots, n-1$ is

$$\hat{\rho}(\tau) = \begin{cases} \hat{\gamma}(\tau)/\hat{\gamma}(0) & \text{if } \hat{\gamma}(0) \neq 0 \\ 0 & \text{if } \hat{\gamma}(0) = 0 \end{cases} \tag{3.1}$$

where $\hat{\gamma}(\tau)$ is the estimated autocovariance given by

$$\hat{\gamma}(\tau) = \frac{1}{n} \sum_{i=1}^{n-\tau} (x_{i+\tau} - \overline{x})(x_i - \overline{x})$$

and $\overline{x} = 1/n \sum_{i=1}^{n} x_i$ is the sample mean.

The sample autocorrelation in (3.1) ranges between -1 and 1. Negative values indicate anti-correlation in packet reception: as more packets are received (lost), the next packet reception is more likely to fail (succeed).

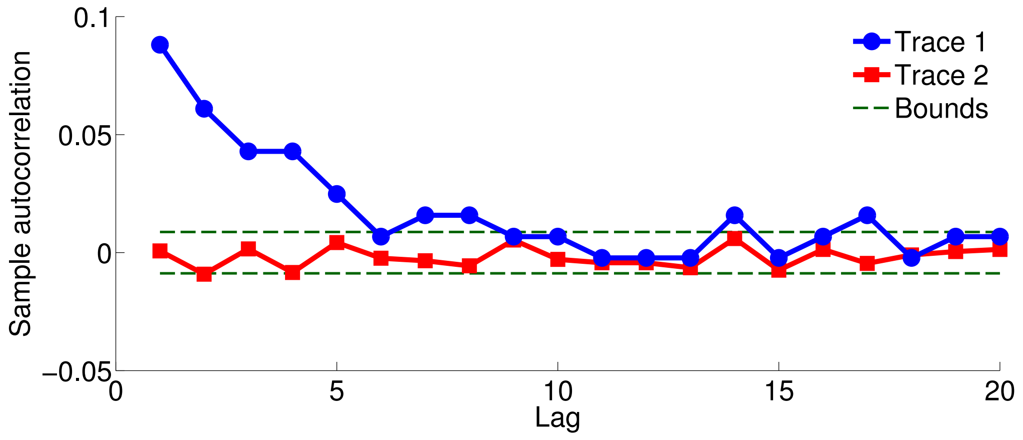


Figure 3.3: Sample autocorrelation up to lag 20 for two of our collected packet reception traces. *The Bernoulli assumption holds for Trace 2, because its sample autocorrelation falls within the confidence bounds starting from lag 1.*

Positive values indicate positive correlation: packet receptions (losses) tend to be followed by more packet receptions (losses).

Values close to zero indicate independence among packet receptions at a given lag, assuming the x_i are i.i.d. Bernoulli random variables. Let $\{x_i\}_{i=1}^{n}$ be a realization of an i.i.d. sequence $\{X_i\}_{i=1}^{\infty}$ of random variables with finite variance. It can be shown that, for a large number of samples n, about 95 % of the sample autocorrelation values should lie within the confidence bounds $\pm 1.96/\sqrt{n}$ [BD91]. Based on this, we define the *correlation lag* as the smallest lag at which the sample autocorrelation lies within $\pm 1.96/\sqrt{n}$. Like [YMKT99], we consider the Bernoulli assumption valid if the correlation lag is 1. Formally: *Given a time series $\{x_i\}_{i=1}^{n}$, the Bernoulli assumption holds at the 0.05 significance level if $|\hat{\rho}(1)| \leq 1.96/\sqrt{n}$.*

As an example, Figure 3.3 plots $\hat{\rho}$ for two of our traces of length $n = 50,000$. The dashed lines at $\pm 1.96/\sqrt{50,000} \approx \pm 0.0088$ represent the confidence bounds. The figures shows that Trace 1 is dependent up to lag 5, but starting from lag 6 the autocorrelation values become insignificant except for a few stray points. By contrast, Trace 2 has an insignificant autocorrelation already at lag 1, indicating that there is no dependence between consecutive packets nor between any other packets in the trace. Thus, the Bernoulli assumption holds for Trace 2 but not for Trace 1.

3.2.4 Results

Based on the above reasoning, we analyze the validity of the Bernoulli assumption for our weakly stationary traces (see Table 3.1). Figure 3.4 shows the percentage of traces with correlation lag greater than 1 (for

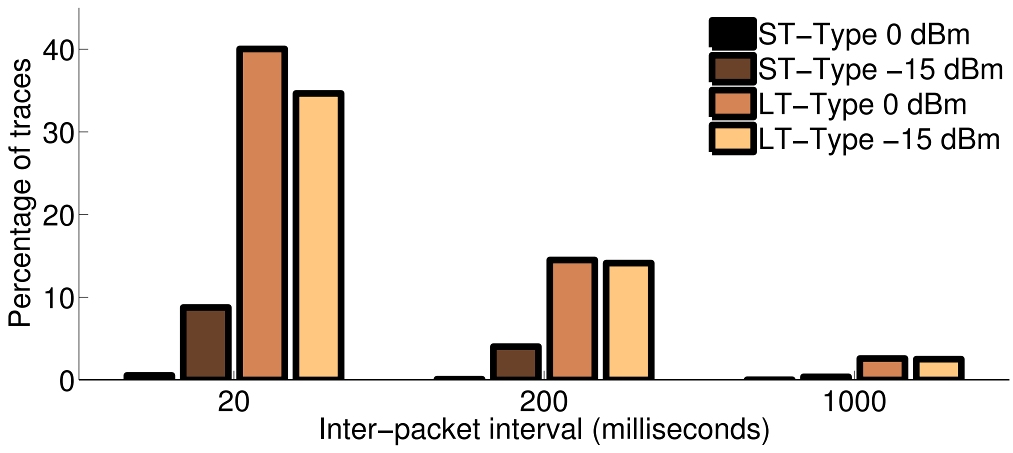


Figure 3.4: Percentage of weakly stationary traces for which the Bernoulli assumption does not hold, for different IPIs and transmit powers. *The Bernoulli assumption is highly valid for ST, and significantly more legitimate for ST than for LT.*

which the Bernoulli assumption does not hold) when examining different IPIs in our traces. We see that the Bernoulli assumption is significantly more legitimate for synchronous transmissions (ST-Type) than for link-based transmissions (LT-Type). For example, at the highest transmit power of 0 dBm, the Bernoulli assumption holds for more than 99 % of the ST-Type traces irrespective of the IPI, whereas it holds only for 60 % of the LT-Type traces at the smallest IPI of 20 ms.

We also see that at the lower transmit power there are more ST-Type traces for which the Bernoulli assumption does not hold. This is mostly because nodes have fewer neighbors and hence benefit less from sender diversity [RHK10]. Indeed, at -15 dBm about 24 % of nodes have at most four well-connected neighbors, which makes their reception behavior approach the one of LT. This is also confirmed by a significant negative Pearson correlation of -0.31 between the number of well-connected neighbors and the percentage of traces for which the Bernoulli assumption does not hold. Finally, Figure 3.4 shows that the autocorrelation decreases as the IPI increases, and becomes negligible at IPI = 1 s also for LT-Type traces. This observation is in line with prior studies on low-power wireless links [SDTL10], thus validating our methodology.

In summary, our results show that the Bernoulli assumption holds to a large extent for ST due to sender diversity [RHK10]. This implies that packet receptions in Glossy can be considered largely independent, so a *single* parameter is sufficient to precisely characterize the probability of receiving a packet. By contrast, packet receptions in LT are often not independent, which necessitates more complex models, such as high-

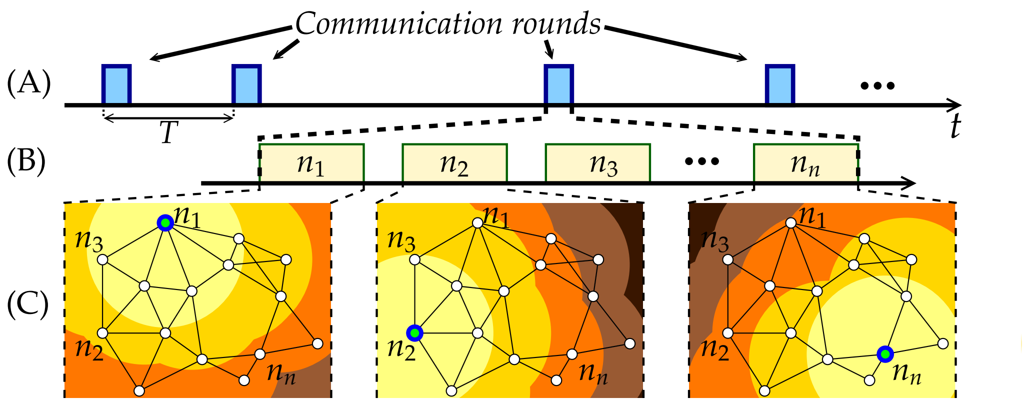


Figure 3.5: LWB's time-triggered operation. Communication rounds occur with a possibly varying round period T (A); each round consists of a varying number of communication slots (B); every slot corresponds to a Glossy flood (C).

order Markov chains [YMKT99], to accurately capture their behavior.

The next section describes LWB, a Glossy-based protocol that we use throughout Sections 3.4 and 3.5 to demonstrate how the validity of the Bernoulli assumption for ST enables simple, yet highly accurate models of multi-hop low-power wireless protocols.

3.3 Low-power Wireless Bus

The basic idea behind LWB is to abstract a network's multi-hop nature by employing *only* ST for communication [FZMT12]. To this end, LWB maps *all* communication demands onto Glossy network floods [FZTS11]. Glossy always and blindly propagates every message from one node to all other nodes in the network, effectively creating the perception of a single-hop network for higher-layer protocols and applications. The resulting protocol operation of LWB is similar to a shared bus, where all nodes are potential receives of all messages; delivery to the intended recipients happens by filtering messages at the receivers.

LWB exploits Glossy's accurate time synchronization for a *time-triggered* scheme that arbitrates access to the (wireless) bus. Nodes communicate according to a global *communication schedule*. A dedicated *host* node computes the schedule online based on the current traffic demands and distributes it to the nodes, determining when a node is allowed to initiate a flood.

As shown in Figure 4.1 (A), communication occurs in *rounds* that repeat with a possibly varying *round period* T. All nodes keep their radios off between two rounds to save energy. Every round consists of a possibly

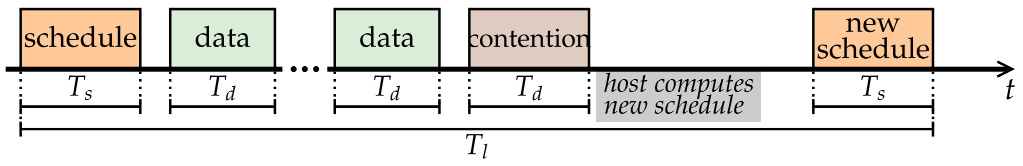


Figure 3.6: Communication slots and activities during a single LWB round.

varying number of *communication slots*, as shown in Figure 4.1 (B). In every slot, at most one node puts a message on the bus (i.e., initiates a flood), while the remaining nodes read the message from the bus (i.e., receive and relay the flooding packet), as illustrated in Figure 4.1 (C).

Figure 3.6 shows the different communication slots within one round of length T_l. Each round starts with a slot of length T_s in which the host distributes the communication schedule. The nodes use the schedule to time-synchronize with the host and to be informed of (*i*) the round period T and (*ii*) the mapping of source nodes to the following data slots of length T_d. A non-allocated *contention slot* of length T_d follows; nodes may contend in this slot to inform the host of their traffic demands. The host uses these to compute the schedule for the next round, which it transmits in a final slot of length T_s.

The host computes the communication schedule by determining a suitable round period T and allocating data slots to the current *streams*. A stream represents a traffic demand, characterized by a *starting time* and an *inter-packet interval (IPI)*, as LWB targets the periodic traffic pattern typical of many low-power wireless applications [GGB+10]. A node can source multiple streams and individually add or remove streams at runtime.

3.4 End-to-end Reliability in LWB

End-to-end reliability refers to a protocol's ability to deliver packets from source to destination, perhaps over multiple hops. It is a key performance metric in low-power wireless [GFJ+09], indicating the level of service provided to users. Many applications do require *probabilistic guarantees* on this metric, for example, to allow for post-processing of data in structural health monitoring [CFP+06] or enable stable control loops [SSF+04].

End-to-end reliability, however, is subject to unpredictable packet loss [SDTL10]. LT-based protocols, therefore, rely on per-hop retransmissions to achieve a certain end-to-end reliability, yet the necessary number of retransmissions depends on the ever-changing loss rates of the individual links along the routing paths. Further, LT-based protocols constantly adapt the routes in response to such changes, which renders reasoning about end-to-end guarantees extremely complex.

By contrast, ST-based protocols, such as LWB, often have no routes to adapt. This facilitates reasoning about end-to-end guarantees, even across multiple hops. To show this, we extend LWB with a retransmission scheme and derive sufficient conditions to provide reliability guarantees. The key insight we use is that the validity of the Bernoulli assumption for ST greatly simplifies the specification of these conditions.

Packet retransmissions in LWB. We consider a typical data collection setting where the LWB host is also the sink [GGB+10]. We augment LWB with packet retransmissions by modifying the scheduling algorithm used at the host to compute the schedule for the next round. Originally, LWB allocates exactly one data slot for each data packet, regardless of the actual reception at the host [FZMT12]. In our modification, the host first checks whether in the current round it received every data packet assigned a slot. For each lost packet, it reallocates a slot in the next round, in which the source node retransmits the lost packet, provided that fewer than k_{max} slots have already been allocated for it. Then, the host allocates data slots for new packets as before.

Consider now an application that requires a minimum end-to-end reliability $\overline{p_d} > 0$ on data packets. We derive sufficient conditions on the minimum k_{max} and overall available bandwidth to provide such guarantee in a probabilistic sense.

Sufficient condition #1: retransmissions. The validity of the Bernoulli assumption for ST in Glossy allows us to consider consecutive retransmissions as independent events. Thus, based on our retransmission scheme, the probability that the host receives a packet from stream s within $k_s \geq 1$ (re)transmissions is $1-(1-p_{d,s})^{k_s}$, where $p_{d,s}$ is the probability that the host receives a packet from stream s in one slot (i.e., during one Glossy flood). To provide the desired guarantee $\overline{p_d} > 0$, we require

$$1 - (1 - p_{d,s})^{k_s} \geq \overline{p_d} \tag{3.2}$$

where $0 < \overline{p_d} < 1$ and $0 < p_{d,s} < 1$. Then, (3.2) holds if

$$k_s \geq \frac{\log(1 - \overline{p_d})}{\log(1 - p_{d,s})} \tag{3.3}$$

Thus, the host must allocate k_s slots to each packet of stream s to provide an end-to-end packet reliability of *at least* $\overline{p_d}$.

Because LWB needs to set an integer upper bound k_{max} on the number of data slots allocated to each packet, the reliability guarantee can only be provided if for all existing streams s

$$\lceil k_s \rceil \leq k_{max} \tag{3.4}$$

Example. Assume one stream with $p_{d,s} = 0.9$, and the host allocates at most $k_{max} = 2$ slots for each packet. In this case, a reliability guarantee of $\overline{p_d} = 0.99$ can be provided, because $1 - (1 - 0.9)^2 = 0.99$. To guarantee $\overline{p_d} = 0.9999$, we need to increase k_{max} to $\lceil \log(1 - 0.9999)/\log(1 - 0.9) \rceil = 4$.

Sufficient condition #2: bandwidth. The bandwidth available in LWB is a function of how often communication rounds unfold: the shorter the round period T, the more data slots are available, yielding increased overall bandwidth. Due to platform-specific constraints on timings and size of the schedule packet, however, at most B data slots can be allocated in a round.

The original scheduling policy minimizes energy while providing enough bandwidth to all traffic demands whenever possible. Specifically, given N streams, the host first computes

$$T_{opt} = \frac{B}{\sum_{s=1}^{N}(1/\mathrm{IPI}_s)} \tag{3.5}$$

where $1/\mathrm{IPI}_s$ is the number of data slots allocated to stream s per time unit, because without retransmissions LWB allocates exactly one slot for each packet. Then, the host obtains the new round period by computing $T = \lceil \max(T_{min}, \min(T_{opt}, T_{max})) \rceil$. The lower bound T_{min} ensures that T is longer than the duration of a round T_l, and the upper bound T_{max} ensures that the nodes stay time-synchronized with the host. If $T_{opt} < T_{min}$, the network is *saturated*, that is, the maximum bandwidth provided by LWB is insufficient to support the current traffic demands. If saturation occurs, the host sets $T = T_{min}$ and informs the nodes.

On the other hand, if a packet must be transmitted $k_s \geq 1$ times to provide a guarantee on the end-to-end packet reliability, every stream s requires k_s/IPI_s data slots per time unit; and all N streams together require $\sum_{s=1}^{N}(k_s/\mathrm{IPI}_s)$. Therefore, to account for packet (re)transmissions, we modify (3.5) as follows

$$T_{opt} = \frac{B}{\sum_{s=1}^{N}(k_s/\mathrm{IPI}_s)} \tag{3.6}$$

As described above, T_{opt} cannot be smaller than the minimum round period T_{min}. Therefore, the total bandwidth is sufficient for k_s packet transmissions only if

$$\sum_{s=1}^{N} \frac{k_s}{\mathrm{IPI}_s} \leq \frac{B}{T_{min}} \tag{3.7}$$

Only if both conditions (3.4) and (3.7) are satisfied, it is guaranteed that packets are delivered with at least probability $\overline{p_d}$.

Example. Consider streams s_1 and s_2 that generate packets with $\mathrm{IPI}_1 = 8\,\mathrm{s}$ and $\mathrm{IPI}_2 = 12\,\mathrm{s}$, and deliver packets to the host with probabilities

$p_{d,1} = 0.99$ and $p_{d,2} = 0.9$. The host allocates up to $k_{max} = 16$ slots per packet and up to $B = 5$ slots per round. The minimum round period is $T_{min} = 2\,\text{s}$. Can LWB guarantee an end-to-end packet reliability of $\overline{p_d} = 0.9999$ for both streams? The condition in (3.4) is satisfied for both streams, because the required numbers of slots $k_1 = 2$ and $k_2 = 4$ are smaller than k_{max}. The condition in (3.7) is satisfied as well, because the optimal round period $T_{opt} = \frac{5}{2/8+4/12} \approx 8.57\,\text{s}$ is longer than the minimum round period T_{min}. Thus, $\overline{p_d} = 0.9999$ can be guaranteed for both streams.

3.5 Energy Consumption in LWB

Energy is a primary concern in low-power wireless systems. Thus, the ability to accurately model a protocol's energy consumption is important, for example, to dimension a system's power sources before deployment, or to estimate the remaining system lifetime during operation [GGB+10].

The major factor contributing to a node's energy consumption in low-power wireless is the time spent with the radio on, because the wireless transceiver may draw several orders of magnitude more power than other components [LM10]. Therefore, we use the radio on-time as a proxy for energy in this chapter. The actual energy is obtained by multiplying the radio on-time with the transceivers' power draw when on.

Deriving a model that precisely estimates the radio on-time of a LT-based low-power wireless protocol is, however, difficult. Nodes generally experience different radio on-times depending on their position in the routing topology, and sudden route changes trigger actions that need to be coordinated across different nodes [BKM+12]. ST simplify a protocol's operation by sparing the need for routes, thus making modeling simpler.

We now demonstrate the above for LWB. This is because a single event—the reception of schedule packets from the host—drives most of a node's actions, as illustrated in Section 3.5.1. As shown in Section 3.5.2, we can derive precise radio on-times for each of the protocol's operational states. The validity of the Bernoulli assumption for ST allows us to consider consecutive schedule receptions as independent. This facilitates computing the long-term frequency of visits to these states, as described in Section 3.5.3, ultimately yielding the expected radio on-time.

3.5.1 Operational States of LWB

The radio on-time of a LWB node depends only on the reception of schedules from the host, which allows a node to time-synchronize and to learn the mapping of nodes to slots for the current round. Based on this, a node knows when communication occurs and activates the radio

Table 3.2: Meaning and radio on-times of FSM states in Figure 3.7.

State	Description	Worst-case radio on-time
B_b	**B**ootstrapping: not synced, radio always on	T_l
B_e	**B**ootstrapping: not synced, radio always on	$T - T_l$
R_b	**R**eceived schedule, drift not estimated	$T_g(\overline{m}) + T_s + T_c$
R_e	**R**eceived schedule, drift not estimated	$T_g(\overline{m}) + T_s$
S_b	**S**ynced: received schedule, drift estimated	$T_g(0) + T_s + T_c$
S_e	**S**ynced: received schedule, drift estimated	$T_g(0) + T_s$
M_b	**M**issed schedule at beginning of current round	$T_g(m) + T_s + \delta_{m1} T_c$
M_e	**M**issed schedule at end of previous round	$T_g(m) + T_s$

accordingly. A node that misses a schedule in a round refrains from communicating during that round, since communication outside of the allocated slots (e.g., due to inaccurate time information) may cause packet loss due to collisions with other transmissions.

Clock drift prevents a node from having perfect time information, even when it constantly receives schedule packets [LSW09]. The synchronization error often increases when missing several schedules in a row, as the effects of clock drift accumulate over time. To compensate for these, a LWB node uses predefined *guard times* in order to turn the radio on shortly before a round is bound to begin, and increases them in discrete steps as it misses more schedules in a row. After more than $\overline{m}$ consecutive missed schedules, a node permanently keeps the radio on until it receives a schedule and time-synchronizes again.

This behavior is reflected in the finite state machine (FSM) in Figure 3.7, which models the behavior of a LWB node depending on received (r) or missed ($\neg$ r) schedule packets. Table 3.2 lists the meaning of each state and the corresponding worst-case radio on-time. Key to the model is whether the schedule packet is received (or missed) at the end of the previous round or at the beginning of the current round. To differentiate the two cases, a node reaches a state labeled X_b following schedules sent at the beginning of a round, and a state labeled X_e following schedules sent at the end of a round. We now derive the radio on-times reported in Table 3.2 for every state in the FSM.

3.5.2 Radio On-time of LWB States

A node starts in state B_e with the radio turned on until it receives a schedule at the beginning of a round ($B_e \rightarrow R_b$) and synchronizes with the host. As shown in Figure 4.1, communication rounds occur every T.

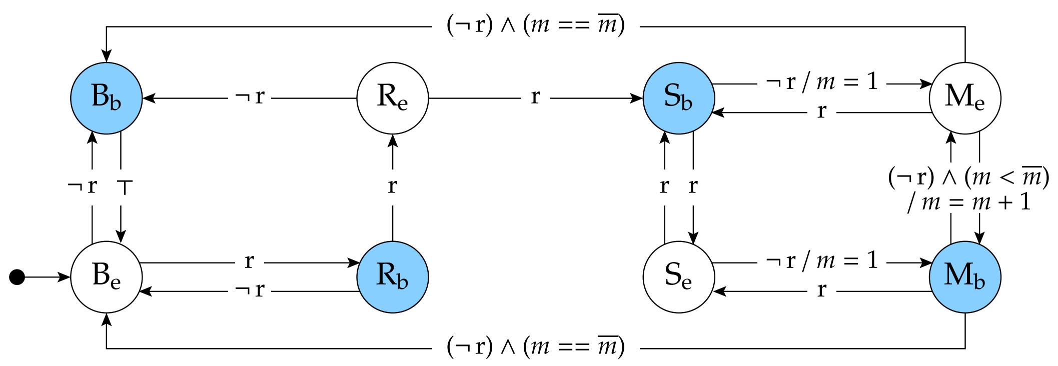


Figure 3.7: FSM modeling the behavior of a LWB node depending on received (r) and missed (¬ r) schedules. *A LWB node reaches states labeled X_b after receiving or missing schedules sent at the* ***b****eginning of a round, and states labeled X_e after receiving or missing schedules sent at the* ***e****nd of a round. The number of consecutively missed schedules m is updated on every transition to M_b or M_e. When m reaches the predefined threshold $\overline{m}$, a LWB node returns to one of the two bootstrapping states B_e or B_b.*

Therefore, a node remains in B_e for at most $T - T_l$, where T_l is the duration of a round illustrated in Figure 3.6. If a node misses such schedule ($B_e \rightarrow B_b$), it keeps the radio on for T_l before it tries again to receive a schedule at the beginning of a round ($B_b \rightarrow B_e$).

As shown in Table 3.2, in all non-bootstrapping states the radio on-time includes the length of a schedule slot T_s and two additional terms: the guard time to compensate for synchronization errors and the radio on-time due to communication.

Guard times. A platform-specific guard time function $T_g(m)$ specifies the guard time before a schedule slot, based on the number of consecutive missed schedules m, with $0 \leq m \leq \overline{m}$. $T_g(m)$ is non-decreasing in m, since the synchronization error typically increases with more missed schedules, as discussed before. For example, if schedules are always received, a node alternates between states S_b and S_e using the smallest guard time $T_g(0)$, but switches to a longer guard time $T_g(1)$ if it misses the schedule at the beginning of the current round ($S_e \rightarrow M_b$) or at the end of the previous round ($S_b \rightarrow M_e$).

An exception to this processing occurs when a node has insufficient information to compute the drift of the local clock compared to the clock of the host. This is the case in states R_b and R_e, when a node has received only one schedule sent at the beginning of a round since the last bootstrapping state (see Figure 3.7). As a result, it cannot estimate the drift of the local clock, because at least two time references are required [LSW09]. Therefore, a node prudently uses the largest possible guard time $T_g(\overline{m})$ in state R_b or R_e, as shown in Table 3.2.

Radio on-time due to communication. The radio on-time for data and contention slots is $T_c = (d_r + d_k)T_d$, where d_r is the expected number of data slots per round, d_k is the average number of contention slots per round, and T_d is the length of data and contention slots (see Figure 3.6). As shown in Table 3.2, T_c is accounted for only when a node participates in a round, after receiving the schedule at the beginning of the current round (state $\mathrm{R_b}$ or $\mathrm{S_b}$) or at the end of the previous round (state $\mathrm{M_b}$ with $m = 1$; the Kronecker delta δ_{m1} in Table 3.2 equals 1 if $m = 1$ and 0 otherwise). We now derive d_k and d_r.

The host schedules one contention slot every $T_k > T$ to save energy under stable traffic conditions [FZMT12]. The average number of contention slots per round is thus $d_k = T/T_k$. The average number of data slots per round d_r depends on whether the network is saturated or not. If saturated, the host allocates all available B slots in every round, so $d_r = B$. Without saturation, given that each data packet can be (re)transmitted up to k_{max} times, on average $d_s \leq k_{max}$ data slots are allocated to each data packet of stream s. The expected number of data slots allocated per time unit to stream s is d_s/IPI_s. The general expression for d_r is thus

$$d_r = \min(B, T \sum_{s=1}^{N} d_s/\mathrm{IPI}_s) \tag{3.8}$$

As described in Section 3.4, the host allocates data slots for each packet of stream s across multiple rounds until either it receives the packet or k_{max} slots are allocated. Thus, the expected number of data slots d_s allocated to a packet of stream s depends on the probability $p_{d,s}$ that the host receives from stream s and on the maximum number of transmissions k_{max}. For $0 < p_{d,s} \leq 1$, it can be shown that the host allocates

$$d_s = \frac{1 - (1 - p_{d,s})^{k_{max}}}{p_{d,s}} \tag{3.9}$$

slots on average to each packet of stream s. Note that for $p_{d,s} = 1$ the host receives the packet always at the first attempt and $d_s = 1$, whereas d_s approaches k_{max} as $p_{d,s}$ goes to 0.

Exploiting the Bernoulli assumption validated in Section 3.2, we estimate next how frequently a LWB node is expected to visit each operational state in the long run. This information, together with the radio on-times we just derived for every state, allows us to estimate the expected radio on-time per round.

3.5.3 Expected Radio On-time of a LWB Round

According to Section 3.2, packet reception with Glossy-based ST can be modeled with high confidence as a Bernoulli trial. We thus characterize

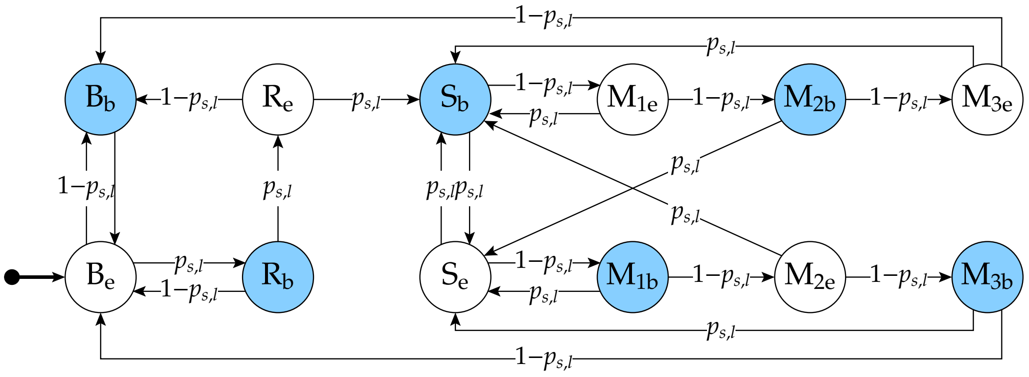


Figure 3.8: DTMC corresponding to FSM in Figure 3.7 for a node that receives schedules from the host with probability $p_{s,l}$ and starts bootstrapping anew after missing more than $\overline{m} = 3$ schedules in a row.

the reception of schedules at a node through a single parameter $p_{s,l}$, the success probability of Glossy-based ST from the host. As a result, the FSM in Figure 3.7 translates into a discrete-time Markov chain (DTMC) where an event r (or$\neg$ r), corresponding to a successful (failed) reception of the schedule, occurs with probability $p_{s,l}$ (or $1-p_{s,l}$).

Our LWB implementation retains the original setting for $\overline{m}$, in which a node returns to one of the bootstrapping states after missing more than $\overline{m} = 3$ consecutive schedules. Figure 3.8 shows the corresponding DTMC. States $\{M_{1b}, M_{2b}, M_{3b}\}$ are equivalent to state M_b in the original FSM for $m = \{1, 2, 3\}$; the same applies to states $\{M_{1e}, M_{2e}, M_{3e}\}$ and state M_e. Note that the DTMC in Figure 3.8 is periodic with period 2: the host sends schedules at the beginning and at the end of a round, thus a node always visits state X_e after state X_b and vice versa.

Knowing the radio on-time for each LWB state as per Table 3.2, we can compute the expected radio-on time per round

$$T_{on} = 2(\mathbf{t_{on}} \cdot \boldsymbol{\pi}) \tag{3.10}$$

where $\mathbf{t_{on}} = (t_{on\mathrm{B_b}}, t_{on\mathrm{B_e}}, \ldots, t_{on\mathrm{M_{3e}}})$ is a vector containing the radio on-times of all DTMC states, $\cdot$ is the dot product, and $\boldsymbol{\pi}$ is the DTMC's stationary distribution. The factor 2 is because, during a round, a node always visits two states of the DTMC as the host transmits two schedules per round.

We can obtain $\boldsymbol{\pi}$ by determining the normalized left eigenvector with eigenvalue 1 of the DTMC's transition matrix. For $\overline{m} = 3$, we have $\boldsymbol{\pi} = (\pi_{\mathrm{B_b}}, \pi_{\mathrm{B_e}}, \ldots, \pi_{\mathrm{M_{3e}}})$, where π_s denotes the long-run frequency of visits to state $s \in \{\mathrm{B_b}, \mathrm{B_e}, \ldots, \mathrm{M_{3e}}\}$. Figure 3.9 plots the stationary distribution $\boldsymbol{\pi}$ against actual values for the probability $p_{s,l}$. For example, we can see that when $p_{s,l}$ approaches 1, a node visits more often states S_b and S_e, as it consistently receives schedule packets, whereas it visits more often

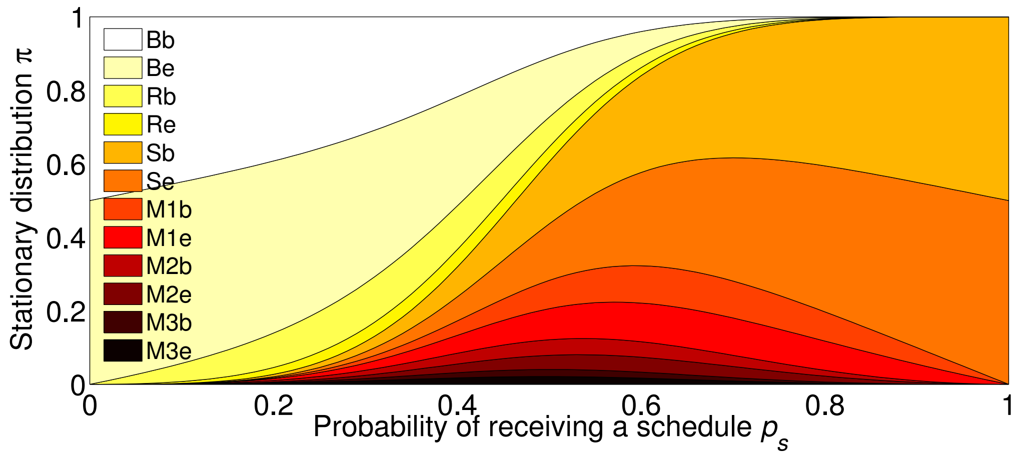


Figure 3.9: Stationary distribution π of the DTMC in Figure 3.8 against the probability of receiving a schedule $p_{s,l}$.

states B_b and B_e when $p_{s,l}$ is close to 0. Typical values for $p_{s,l}$ measured in practice, however, range above 0.99 with Glossy [FZTS11], making it very unlikely that a node ever returns to a bootstrapping state.

The DTMC in Figure 3.8 and the expression in (3.10) confirm our hypothesis that ST simplify the modeling of multi-hop protocols. This is due to (*i*) the validity of the Bernoulli assumption for Glossy-based ST, and (*ii*) the absence of routes in LWB. In the following, we demonstrate that the resulting energy model is not only simple but also highly accurate.

3.6 Validation

To verify accuracy and practicality, we compare estimates of our models with real measurements. Prior to deployment, analyzing the sufficient conditions for a given end-to-end reliability against foreseeable network conditions can, for example, drive the node placement to increase connectivity; moreover, exercising the energy model for different network traffic settings can help designers dimension the power sources. At run-time, monitoring the expected network performance allows system operators to proactively perform maintenance tasks (e.g., replacing nodes or batteries), and to adjust system parameters in response to changes in the network conditions. Our model validation mimics these scenarios.

3.6.1 Settings and Metrics

We use the FlockLab testbed, which consists of 30 TelosB nodes deployed both inside and outside a university building [LFZ+13b]. We use the highest transmit power of 0 dBm, yielding a network diameter of 4 hops. To reduce sources of packet loss we cannot control, we use IEEE 802.15.4 channel 26 to minimize interference from co-located Wi-Fi networks. Instead, we artificially induce packet loss during ad-hoc experiments. In all experiments, data packets carry a payload of 15 bytes.

We extend the original LWB implementation with packet retransmissions, as described in Section 3.4, and measure: (*i*) the *end-to-end reliability*, the fraction of generated data packets successfully delivered at the host; and (*ii*) the *radio on-time per round*. We measure the former based on packet sequence numbers received at the host and the latter using established software-based methods [DOTH07]. Unless otherwise stated, we set the maximum number of transmissions per packet k_{max} to 50.

Most of our model's inputs are implementation constants: the guard times $T_g(\{0,1,2,3\}) = \{1,3,5,20\}$ ms, the lengths of schedule and data slots $T_s = 15$ ms and $T_d = 10$ ms, the length of a round $T_l = 1$ s, and the maximum number of data slots per round $B = 45$. However, a few inputs are precisely known only at run-time:

- The probability of receiving a schedule $p_{s,l}$. A node n estimates $p_{s,n}$ locally based on past schedule receptions, and reports it to the host by piggybacking it on data packets.
- The round period T and the expected number of data slots per round d_r. Both are determined by the scheduling policy, and depend on the streams' IPIs and the probability $p_{d,s}$ that the host receives a packet from stream s. The host estimates $p_{d,s}$ like nodes estimate $p_{s,n}$.
- The radio on-times during schedule and data slots $T_{s,n}$ and $T_{d,n}$, which are generally shorter than the lengths of schedule and data slots T_s and T_d, because nodes may turn off the radio before the end of a slot depending on their position in the network [FZMT12]. Each node n estimates $T_{s,n}$ and $T_{d,n}$ locally, and reports them to the host as it does for the probability $p_{s,l}$ of receiving a schedule.

When using our models offline, one can consider conservative estimates for $p_{s,n}$, $p_{d,s}$, $T_{s,n}$, and $T_{d,n}$ based on coarse-grained deployment information or data from exploratory experiments [BKM+12]. From these and the streams' IPIs defined in the requirements specification, one can determine T and d_r based on the scheduling policy. During system operation, one can refine these values using up-to-date run-time estimates. Specifically, in our experiments, nodes maintain two counters to estimate $p_{s,n}$: the number of received r_s and the number of expected e_s schedule packets. Nodes embed r_s/e_s into data packets and halve both counters whenever e_s reaches a threshold, which behaves similarly to an exponentially weighted moving average (EWMA). The nodes estimate $T_{s,n}$ and $T_{d,n}$ in a similar way, and so does the host to estimate $p_{d,s}$.

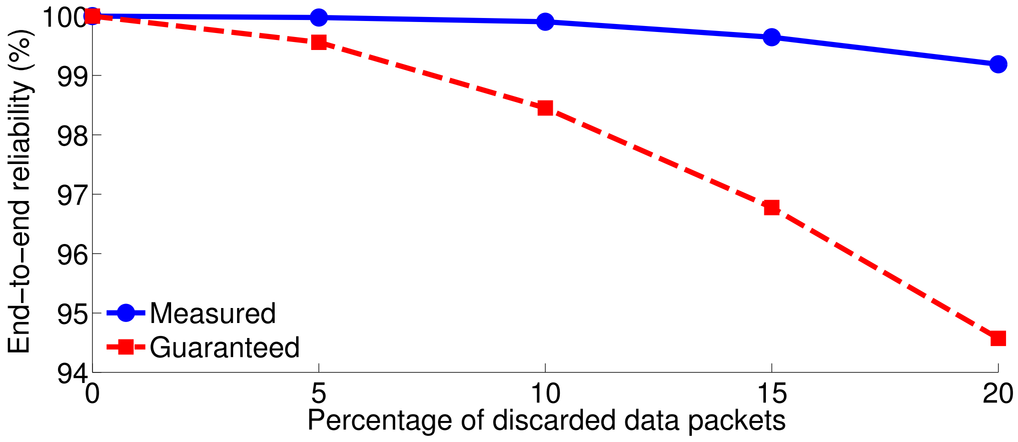
We run experiments on FlockLab to assess the accuracy of our intentionally simple parameter estimations. To test different network conditions, three nodes at the edge of the testbed randomly discard up to 50 % of schedule packets. Such high loss rates are very unlikely given Glossy's typical reliability of more than 99 % [FZTS11]. Nevertheless, we find that our parameter estimates are accurate to within less than 1 % across all settings, including the case of 50 % missed schedules.

3.6.2 End-to-end Reliability

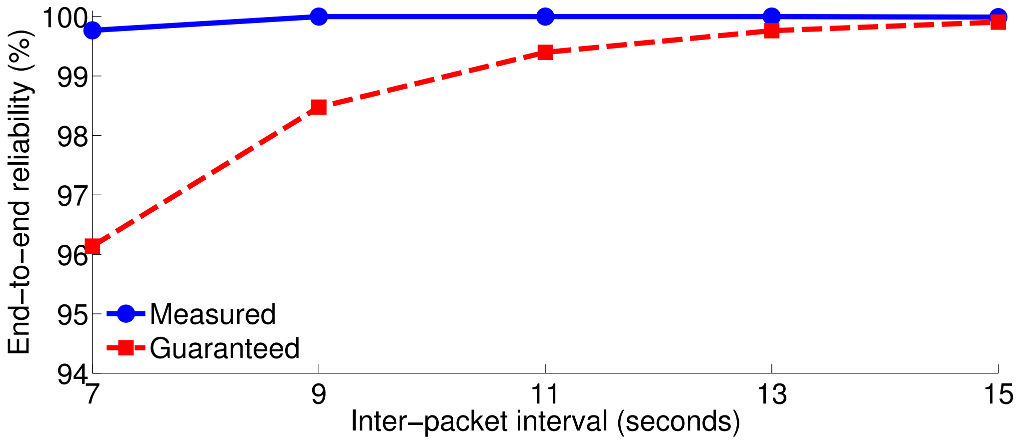
We study how significant network unreliability and changes in traffic load may affect the guarantee on end-to-end packet reliability. To test the former, we let 29 nodes generate packets with IPI = 7 s, while the host discards between 0 % and 20 % of the received data packets to emulate network unreliability. The round period is $T = 6$ s. To analyze changes in traffic load, all nodes generate packets with an increasing IPI in different runs: from 7 s to 15 s in steps of 2 s, while the host discards 5 % of the data packets. The round period is set to $T = 10$ s. Both settings mirror conditions found in real deployments [GGB+10]. The experiments take 1.5 hours, and the maximum number of transmissions per packet is $k_{max} = 3$.

Figure 3.10 shows for both experiments the measured end-to-end reliability and the maximum end-to-end reliability that can be guaranteed according to our analysis in Section 3.4. As for the measured values, the slight drop in Figure 3.10(a) is because the maximum number of transmissions $k_{max} = 3$ is insufficient to deliver all packets to the host, whereas the slight drop in Figure 3.10(b) at the smallest IPI is due to insufficient bandwidth.

While the results confirm that our specific LWB executions never provide an end-to-end packet reliability lower than the guaranteed values, the figure also shows that the latter drop more severely than the measured ones. This is expected, as the analysis for guarantees on end-to-end packet reliability entails over-provisioning the number of data slots allocated to a packet, although often only a small fraction thereof is actually needed to receive it. In the experiments where we emulate network unreliability, for example, we compute from (3.7) that the host can allocate at most $k_s = 1.81$ slots for a packet of each stream s. Based on (3.2), this value guarantees an end-to-end reliability of $\overline{p_d} = 94.57\,\%$ when 20 % of data packets are discarded, as shown in Figure 3.10(a). However, according to



(a) Varying percentage of artificially discarded data packets at the host.



(b) Varying inter-packet interval (IPI) of generated data packets at the nodes.

Figure 3.10: Measured end-to-end reliability of LWB and maximum end-to-end reliability that can be guaranteed according to the analysis of Section 3.4.

(3.9) packets are actually received after $d_s = 1.24$ slots on average.

The conclusion is that the gap between the real executions and the guaranteed values provides, in a sense, information *in advance* about how worse the system may possibly perform under worst-case assumptions. Thus, system operators can take appropriate countermeasures before a problem, although highly unlikely, manifest in the measurements, to effectively satisfy the application requirements at all times.

3.6.3 Energy Consumption

We evaluate the accuracy of our energy model in 5-hour experiments: 29 nodes generate packets with IPI = 6 s, and the round period is T = 6 s. In our model, the probabilities of receiving schedule and data packets

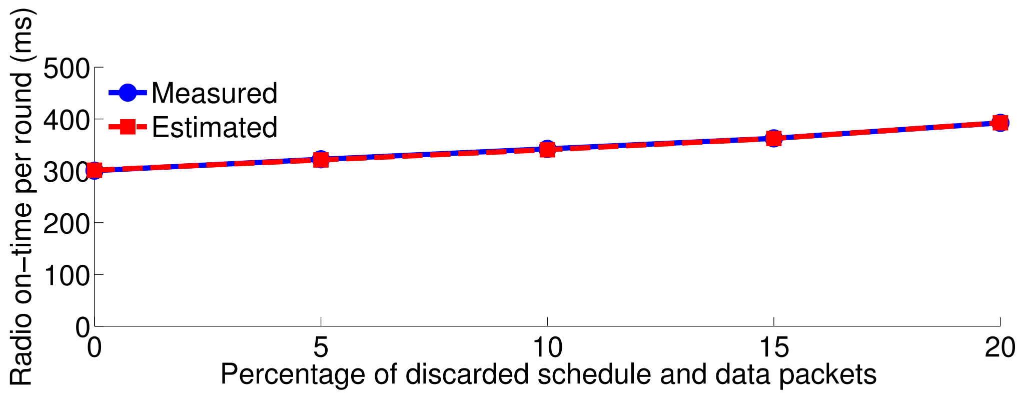


Figure 3.11: Estimated and measured radio on-time per round when artificially discarding schedule and data packets. The average model error is 0.25 %.

$p_{s,n}$ and $p_{d,s}$ are the most critical inputs. To test the model output against different probability values, we let three nodes at the edge of the testbed randomly discard between 0 % and 20 % of schedules, while the host also discards the same percentage of data packets. Exercising the energy model against different IPIs and round periods T simply scales the model output proportionally.

Figure 3.11 plots the estimated and measured radio on-time averaged over the three nodes. Overall, the results show that our energy model is highly accurate, with an average relative error of 0.25 %. In comparison, recent work on modeling LT-based multi-hop protocols reports relative errors in energy consumption between 2 and 7 % [ZFM+12]—one order of magnitude larger than ours. Considering also that our work spans a *complete* multi-hop protocol rather than individual components, as discussed in Section 4.1, this confirms our initial hypothesis that ST enable highly accurate protocol modeling.

We maintain that this is mainly due to the accuracy of the parameter estimation, as discussed in Section 3.6.1, and to the validity of the DTMC model. To verify the latter, we additionally run a 3-hour experiment with three nodes at the edge of the testbed discarding 50 % of schedule packets. Using FlockLab's tracing facility [LFZ+13b], we precisely measure the fractions of time these nodes spend in each of the twelve DTMC states shown in Figure 3.8. Figure 3.12 shows these measurements next to what the DTMC model predicts for $p_{s,l} = 0.5$; for better visibility, we merge the corresponding states at the beginning and at the end of a round into single states, leaving six instead of twelve states in the plot. We see that expectations and measurements indeed match very well, and would do so even better for longer experiments as the long-term behavior of the system emerges.

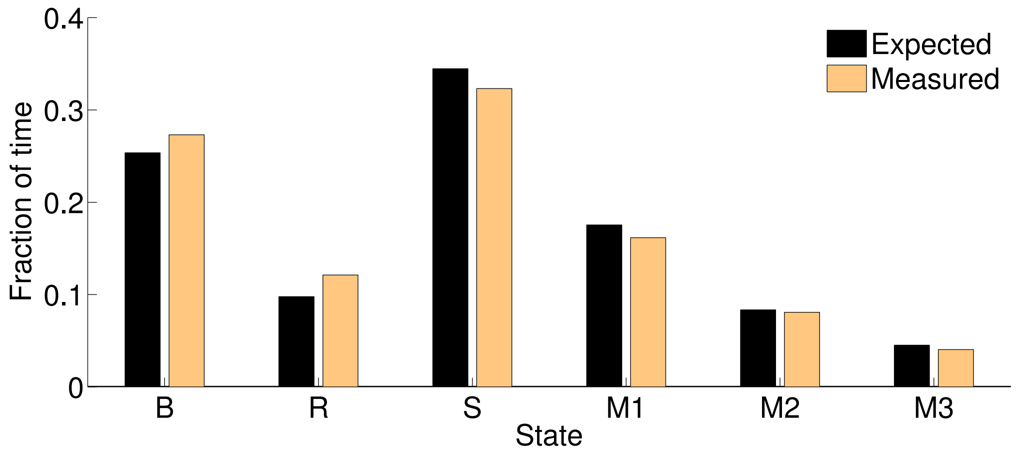


Figure 3.12: Fraction of time in FSM states when discarding 50 % of schedules, measured on three nodes and predicted by the DTMC model. For illustration the corresponding states at the beginning and at the end of a round are merged.

3.7 Summary

In this chapter, we studied whether ST enable simple yet accurate modeling of multi-hop low-power wireless protocols. Our experimental results show that the Bernoulli assumption is highly valid for ST in Glossy and more legitimate for ST than LT. We exploit these findings in the modeling of LWB's end-to-end reliability and long-term energy consumption. Our validation using real-world experiments confirms that accurate models of ST-based protocols are feasible, demonstrating a model error in energy of 0.25 %. We believe our contributions represent a key stepping stone in the development and analysis of ST-based protocols.

Furthermore, our results are important for closed-loop control over lossy wireless networks. Many control algorithms can be designed to tolerate a small fraction of packet loss without sacrificing control performance and stability. This, however, is based on the assumption that packet receptions are statistically independent, that is, the few losses do not happen as a longer burst of multiple consecutive losses [SSF$^+$04]. We show that due to the validity of the Bernoulli assumption for ST, such adverse packet loss bursts virtually never occur when using Glossy to communicate packets throughout the network. Given this beneficial property of Glossy and LWB's bus-like operation, which is conceptually similar to wired fieldbusses used in safety-critical embedded systems, suggests that LWB could be a good candidate protocol to support CPS applications with high reliability and real-time requirements.

4

Blink: Real-time Communication in Multi-hop Low-power Wireless

The tangible benefits of low-power wireless technology in CPS and especially in automation and control are, by now, widely acknowledged. Key industry players argue: "... the possibilities are endless: wireless technology will unlock value in one's process chain far beyond merely avoiding the wiring costs" [hon]. Such benefits include improved control safety and process reliability, lower installation and maintenance costs, and unprecedented flexibility in selecting sensing and actuation points [ÅGB11, Sve]. Example applications range from level control of dangerous liquids to protect against environmental threats [hon] through rapid prototyping of automation solutions in retrofitting buildings [ABD+11] to minimally invasive monitoring of safety-critical assets [HSE]. Because of lower costs and ease of installation, battery-powered embedded platforms with low-power wireless radio transceivers and computationally constrained MCUs are preferred in real deployments [hon, Sve].

A trait common to all these applications is that packets must be delivered within *hard end-to-end deadlines* [But11], for example, to ensure the stability of the controlled processes [ÅGB11]. Hard deadlines entail that packets not meeting their deadlines have no value to the application and hence count as lost. Support for this type of *real-time traffic* is mainstream in wired fieldbusses [CAN, Fle], but hard to attain in a low-power wireless network. This is due to, for example, the dynamics of low-power wireless links [BKM+12], the need for multi-hop communication to cover large areas, and the resource scarcity of the employed devices.

Prior Work

Existing efforts to address these challenges can be broadly classified depending on whether they require local or global knowledge as an input for computing packet schedules.

Local knowledge. In SPEED [HSLA05], each node continuously monitors other nodes within its direct radio range, for example, to detect transient congestion. Using only this node-local information, each node computes and follows its own transmission schedule. On a conceptual level, this fully localized approach of SPEED is also adopted by numerous MAC-layer solutions [GHLX09, KLS+01, LBA+02, WC01] and by systems designed to support specific traffic patterns [HLR03].

Albeit these solutions scale very well because of their localized nature, they *cannot* support applications that rely on traffic with hard real-time deadlines. As described in Section 4.1, those requirements are indeed specified from an end-to-end perspective. However, a device that reasons based on local information and that can only influence its surroundings is oblivious of the global picture and has limited means to affect it.

Global knowledge. By contrast, the WirelessHART [har], ISA 100 [isa], and IEEE 802.15.5e TSCH [tsc] standards and other solutions [OBB+13, SNSW10] compute communication schedules based on global information about the *network state*: the instantaneous conditions at the physical layer that possibly enable communication between any two nodes in the network. Global network state information thus takes the form of a connectivity graph, where the weight of edge A $\rightarrow$ B represents the quality of the link from node A to node B, for example, in terms of the packet reception rate seen by B when receiving packets from A.

Using this global network state as an input, in WirelessHART and similar solutions a central network manager computes and distributes communication schedules tailored to each node, thereby forming end-to-end routing paths from sources to destination(s). Then, each node follows its own schedule locally. This approach has two fundamental problems:

1. Global network state is time-varying due to fluctuating low-power wireless links [SDTL10], environmental influences [BKM+12], and device outages or device mobility [XTLS08]. Any such change in the network state must first be detected and then communicated to the central manager for updating the connectivity graph and possibly re-computing and re-distributing communication schedules to each device. While this happens new changes may occur, requiring to re-iterate the same processing over and over again. Meanwhile, packets are lost because of inconsistent routing paths or miss their deadlines because of stale communication schedules.

2. The need to compute *per-node* communication schedules also causes severe scalability problems in deep multi-hop networks. Existing works map the problem of scheduling real-time traffic in a wireless multi-hop network to the problem of scheduling tasks on a *multiprocessor* machine [SXLC10]. As a result, in WirelessHART networks for example, computing optimal schedules takes time at least exponential in the network diameter [SXLC10]. Although some attempts have been made to address this issue [CWLG11, SXLC10, ZSJ09], WirelessHART schedulers are hardly practical in networks that extend across more than three hops [CWLG11].

Because of these fundamental problems, any solution relying on global network state information cannot support hard real-time applications either. This fact is also acknowledged by major industry players who contributed to the WirelessHART standard: "... none of the technologies provide any hard guarantees on deadlines, which is needed if you should dare to use the technology in critical applications" [Per].

Thus, the problem of providing hard real-time guarantees in multi-hop low-power wireless networks remains unsolved.

Contribution and Road Map

This chapter introduces Blink, the first real-time low-power wireless protocol that provides hard guarantees on end-to-end packet deadlines in large multi-hop networks, while simultaneously achieving low energy consumption. Blink seamlessly handles dynamic changes in the network state and in the application's real-time requirements, and readily supports scenarios with multiple controllers and actuators.

The approach we adopt in Blink is radically different from prior art: Blink uses *global* knowledge to compute a single *global* schedule that applies to all nodes in the network. The key idea is to detach Blink's operation from the global network state, whose rapid variations would lead to the same problems we observe in prior solutions. To this aim, an enabling factor is our choice of the LWB [FZMT12] as Blink's underlying communication support. As described in Section 4.2, LWB is an existing best-effort protocol that exclusively employs network-wide Glossy floods for communication [FZTS11]. Crucially, Glossy's operation allows us not to consider the time-varying global network state information as an input to the scheduling problem in Blink.

Building upon this foundation, we describe Blink's overall design in Section 4.3, while Section 4.4 details Blink's efficient protocol operation, which rests upon three key contributions:

- *Problem mapping.* In LWB all nodes are time-synchronized and communicate according to the same global schedule. Moreover, Glossy allows us to abstract away the network state, as if it was a virtual single-hop network. Because of these two observations, we can treat the entire network as a single resource that runs on a single clock. Unlike previous approaches, we can thus map the real-time scheduling problem in Blink to the problem of scheduling tasks on a *uniprocessor*, making it easier to solve than prior art [SXLC10].

- *Real-time scheduling policies.* We conceive scheduling policies based on the EDF principle [LL73]. Using these policies, Blink computes online a communication schedule that provably meets all deadlines of packets released by a set of admitted real-time packet streams, while minimizing the network-wide energy consumption within the limits of the underlying LWB communication support. At the same time, Blink tolerates dynamic changes in both the network state and the set of streams.

- *Data structures and algorithms.* We design and implement a highly efficient priority queue data structure and algorithms that utilize it to enable EDF scheduling on resource-constrained devices. Based on these, we demonstrate the first implementation of EDF on low-power embedded platforms. This is notable per se: due to its run-time overhead, EDF has seen little adoption even on commodity hardware, despite its realtime-optimality [But05, SAÅ+04].

Section 4.5 reports on the evaluation of our Blink prototype on two testbeds of up to 94 nodes [BVJ+10, LFZ+13b], three state-of-the-art MCUs, and using a timing-accurate instruction-level emulator [EOF+09]. Our results show that Blink meets almost 100 % of packet deadlines; the few deadline misses are due to packet loss, which cannot be fully avoided over a lossy wireless channel. Further, by using our dedicated data structures and algorithms, Blink achieves speed-ups of up to 4.1× compared to a conventional implementation of our scheduling policies on state-of-the-art MCUs, which prove to be instrumental to the viability of EDF-based real-time scheduling on certain low-power embedded platforms.

We discuss trade-offs and limitations of our current Blink prototype in Section 4.6 and conclude in Section 4.7.

4.1 Problem Statement

The scheduling problem is a function of the application requirements, the characteristics of the deployment, and the devices employed. We discuss these aspects next.

Applications. CPS are increasingly deployed as a means to embed sophisticated feedback loops into the physical environment [SLMR05]. CPS devices achieve this by tightly orchestrating computing, control, and physical elements through sensors and actuators. Example application domains range from static installations in industrial and building automation, process control, or smart grids [ÅGB11, Whi08] to settings involving highly mobile autonomous computing elements [MMWG14].

These applications place hard real-time requirements on end-to-end communication. Embedded devices periodically stream sensor data or control signals under given timing constraints. Every device may source multiple such streams. The data is then used for monitoring or to feed time-critical control loops. These control loops may execute right on the actuators that affect the environment, or on a few dedicated controller devices that periodically distribute control signals to the actuators, again subject to hard real-time constraints.

Let Λ denote the set of all *n streams* in the network. Each stream $s_i \in \Lambda$ *releases* one packet at a regular periodic interval P_i, called the *period* of stream s_i. The *start time* S_i is the time when stream s_i releases the first packet. Every packet released by stream s_i must be delivered to the destination(s) within the same *relative deadline* D_i. The next packet is only released after the *absolute deadline* of the previous packet, so deadlines are less than or equal to periods (i.e., $D_i \leq P_i$). We often refer to the absolute deadline of a stream as a shorthand for the absolute deadline of the most recent packet released by the stream. Overall, each stream $s_i \in \Lambda$ is characterized by its *profile* $\langle S_i, P_i, D_i \rangle$. If there are k streams with the exact same profile, we also write $k\langle \cdot, \cdot, \cdot \rangle$.[1]

The actual real-time requirements determining the stream profiles are highly application-dependent. The physical dimension recorded through sensors or the nature of the feedback loop often dictate a stream's period P_i. For example, temperature control in liquid volumes [PL10] demands periods on the order of minutes, and coordinated multi-robot control runs with periods of at most tens of seconds [MMWG14]. On the other hand, compressor speed control requires periods down to a few microseconds [ÅGB11]. Low-power wireless is applicable with the greatest advantages in the former type of applications [Whi08]. The monitoring or control process governs a stream's deadline D_i and starting time S_i. For example, closed-loop control typically requires shorter deadlines than open-loop control [Oga01].

Deployments and platforms. Resource-constrained embedded devices amplify the benefits with regards to flexibility and ease of installation and maintenance [ÅGB11, PL10]. Typical devices feature a 16- or 32-bit MCU, a few kB of data memory, and rely on non-renewable energy sources [ÅGB11, Whi08]. This motivates the use of low-power wireless, which reduces the energy costs but limits the bandwidth and makes the system susceptible to interference and environmental factors [BKM+12].

Deployments consist of tens to hundreds of devices. The devices are typically installed at fixed locations as determined by the application and control requirements. Because of energy constraints that limit the individual radio ranges, designers rely on multi-hop networking to ensure overall connectivity. Nodes may also be added or moved on the fly to optimize measurement locations or prototype improvements over existing installations [XTLS08]. The network can therefore exhibit some degree of mobility. Emerging CPS scenarios, instead, feature partially or completely mobile networks, for example, swarms of arial drones to enable precision agriculture [Vil12]. These characteristics further add to the temporal dynamics of low-power wireless links [BKM+12, SDTL10].

Problem. Thus, a solution to support real-time low-power wireless must meet the application-defined packet deadlines, while also achieving:

- *scalability* in terms of the number of streams, nodes, and the size of the deployment area;

[1]For simplicity, we assume a stream releases one packet at a time. If a stream $\langle S_i, P_i, D_i \rangle$ releases k packets at a time, we implicitly transform this into $k\langle S_i, P_i, D_i \rangle$ streams each releasing one packet.

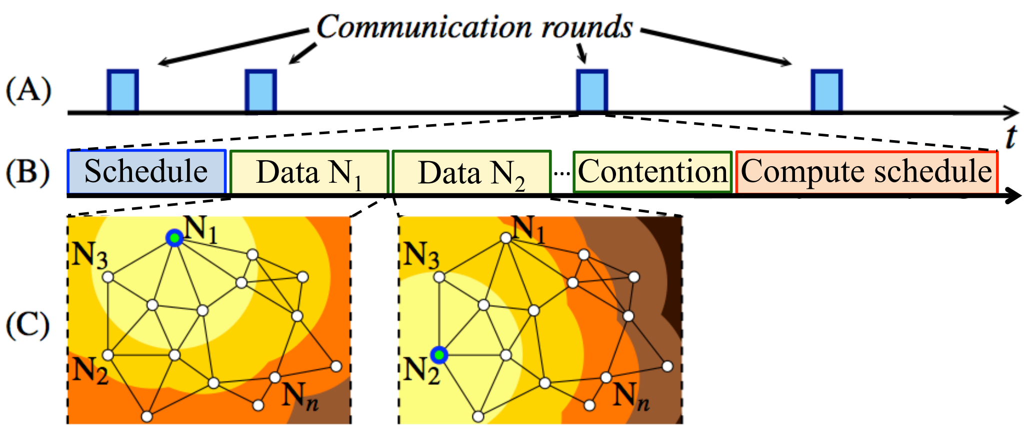


Figure 4.1: Time-triggered operation and sequence of slots in a LWB round.

- *adaptiveness* to accommodate dynamic changes in the set of active streams and the global network state;
- *energy efficiency* to achieve long system lifetimes in the face of limited energy resources;
- *viability* with respect to the limited bandwidth and computational resources of the employed devices.

Subject to these, we can formulate the problem as finding communication schedules such that, given n streams, $n = |\Lambda|$, for every stream $s_i \in \Lambda$, every packet released by stream s_i is delivered within D_i time units.

4.2 Foundation

To detach the operation of Blink from the time-varying network state, we leverage LWB as the underlying communication support [FZMT12]. LWB is an existing non-real-time protocol that, conceptually, turns a multi-hop wireless network into a shared bus, where all nodes are potential receivers of all packets. The nodes are time-synchronized and communicate in a *time-triggered* fashion according to a global communication schedule. To implement the shared bus abstraction, LWB maps *all* communications onto efficient Glossy floods [FZTS11]. Glossy propagates a packet from one node to all other nodes within a few milliseconds. Crucially, in doing so, Glossy's protocol logic is independent of the network state [FZTS11], as opposed to almost all existing low-power wireless protocols.

LWB operation. As shown in Figure 4.1 (A), LWB's operation unfolds in a series of *communication rounds* of *fixed* duration, executed simultaneously by all nodes. Nodes keep their radios off between rounds to save energy. Each round consists of a sequence of non-overlapping *communication slots*, as shown in Figure 4.1 (B). All nodes engage in the communication in every slot: the node to which a slot was assigned places a packet on the bus (i.e., initiates a flood), and all other nodes read the packet from the bus (i.e., receive and relay the packet), as shown in Figure 4.1 (C). At the end of a round, only the intended receivers, which are encoded in the packets, deliver the packet to the application.

Each round starts with a slot allocated to a specific node, called *host*, for distributing the communication schedule, as shown in Figure 4.1 (B). The schedule specifies when the next round starts and which nodes can send application data in the following data slots; there are at most B data slots in a round. Since nodes can only send during a round, B and the time between rounds determine the bandwidth provided by LWB. The shorter this time the more bandwidth is offered to nodes, and vice versa. The time between rounds is upper-bounded to let the nodes update their synchronization state often enough to compensate for clock drifts.

If a node receives the communication schedule, it time-synchronizes with the host and participates in the round; otherwise, it does not take any action until the beginning of the next round. To inform the host about their traffic demands, nodes may compete in a final *contention slot*. Due to capture effects [LF76], with high probability one node succeeds despite contention and its request reaches the host. Based on all traffic demands received thus far, the host computes the schedule for the next round.

A key observation is that the energy *overhead* of LWB is exclusively determined by how often the communication rounds unfold over time. Indeed, it is not necessarily the case that all B data slots in a round are used. This occurs only when an application's instantaneous bandwidth demands align perfectly with LWB's offered bandwidth. Otherwise, some data slots may remain unused, so nodes can turn off their radios during these unused slots to save energy. Thus, the energy *overhead* consists of the energy needed to compute (host) and transmit schedules (all nodes).

Although this purely flooding-based approach to multi-hop communication in LWB may seem wasteful, LWB outperforms prior solutions in end-to-end reliability and energy efficiency; for example, LWB's reliability ranges above 99.9 % in real-world experiments, with energy consumption on par with or even better than state-of-the-art solutions [FZMT12].

Benefits to Blink. Our design choice of building Blink on top of LWB as the communication support brings three assets:

- The use of network-wide Glossy floods for all communications in place of point-to-point transmissions creates, in essence, a *virtual single-hop network* whose operational logic is *independent* of the state of individual wireless links. As a result, the host can take scheduling decisions without considering the network state as an input.

- The previous point, together with LWB's time-triggered operation, allow us to abstract a multi-hop low-power wireless network as a single resource that runs on a single clock. This, in turn, allows us to map the real-time scheduling problem in Blink to the well-known problem of *uniprocessor* task scheduling, making it simpler to solve than the multiprocessor formulation found in prior works [SXLC10].

- LWB offers useful system-level functionality. It supports different traffic patterns, such as one-to-many, many-to-one, and many-to-many, which makes it an appropriate choice for scenarios involving multiple actuators or controllers. Unlike existing solutions [har, isa], LWB also features mechanisms to resume its operation after a host failure, overcoming single point of failure problems [FZMT12].

4.3 Overview

Irrespective of its benefits, we need to address two issues to make LWB ready for real-time. First, the existing LWB scheduler is oblivious of packet deadlines, and only meant to reduce energy consumption [FZMT12]. We must therefore conceive a suitable policy to schedule packets such that all deadlines are met without unnecessarily sacrificing energy efficiency. Second, such a real-time scheduler must execute within strict time limits on a severely resource-constrained platform. As shown in Figure 4.1 (B), the longer it takes to compute the schedule for the next round, the fewer data slots are available given the fixed duration of a round, thus reducing the available bandwidth. We must thus provision an efficient scheduler implementation that is fully cognizant of the platform restrictions.

To this end, our design of Blink is driven by two important goals that we must achieve simultaneously:

1. *Realtime-optimal [SAÅ+04] scheduling*, which entails to admit a new stream if and only if there exists a scheduling policy able to deliver all packets by their deadlines, and to deliver all packets released by the admitted streams by their deadlines.

2. *Minimum network-wide energy consumption*, which in LWB entails to minimize the number of rounds over any given interval, because every round incurs a constant energy overhead regardless of the number of packets sent in the round, as discussed in Section 4.2.

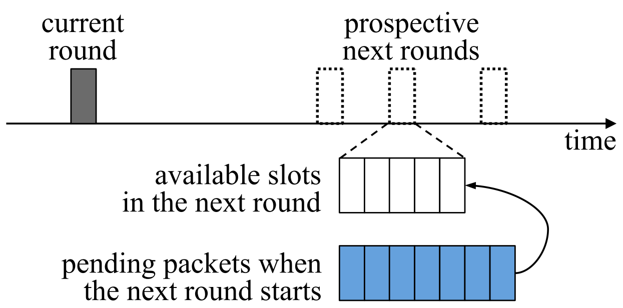

current round
prospective next rounds
time
available slots in the next round
pending packets when the next round starts

Figure 4.2: Illustration of important problems Blink needs to address. *At the end of the current round, Blink must first compute the start time of the next round, and then allocate pending packets to the available slots in a round. Also, before Blink can accept a new stream or update the profile of an existing stream, it must check that the modified stream set is schedulable (i.e., there exists a schedule such that all deadlines can be met).*

Each of these goals arguably leads to a difficult problem on its own, and to a significant challenge if considered together. Taking on this challenge, we require functionality in Blink that solves three interrelated problems:

1. *Start of round computation.* Shown at the top of Figure 4.2, at the end of the current round Blink must decide when the next round should start. This may happen between the time the current round ends and the maximum time allowed between subsequent rounds in LWB, as explained in Section 4.2. Per our two goals, Blink must make this decision so as to meet all deadlines of packets released by the admitted streams, while simultaneously minimizing energy consumption. Intuitively, the earlier the next round starts, the better it is in terms of meeting deadlines; on the contrary, the earlier a round starts, the more energy is consumed in the long run.

2. *Slot allocation.* Once the start time of the next round is computed, given a number of packets waiting to be transmitted, Blink must decide which and how many of these packets will be sent in the next round, as illustrated at the bottom of Figure 4.2. As mentioned in Section 4.2, there may be fewer pending packets than the B data slots available in a round. In case there are more pending packets than available data slots, however, Blink needs to prioritize pending packets of different streams in some meaningful way.

3. *Admission control.* The set of streams and/or their profiles can change over time, for example, when the application requirements change or nodes are added or removed. Thus, Blink must check at runtime whether adding a new stream or updating the profile of an already admitted stream leads to a modified stream set that is *schedulable*, meaning that our solution to the two problems above can deliver

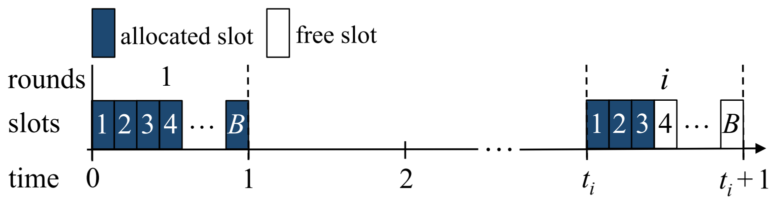


Figure 4.3: Discrete-time model of LWB used throughout Section 4.4. *Each round is of unit length and comprises B data slots, each of which is either allocated to a packet or free. Here, the i-th round with three allocated slots is scheduled to start at time t_i and thus ends at time $t_i + 1$, where t_i is a non-negative integer.*

all packets of all streams by their deadlines. In essence, Blink needs to make sure that given the modified stream set it will never be the case that all streams together demand a bandwidth that exceeds the maximum available bandwidth over any interval of time.

To realize these functionality, Blink builds on LWB as communication support as well as on novel scheduling policies that are provably realtime-optimal and minimize the network-wide energy consumption within the limits set by LWB, and efficient data structures and algorithms that let the new real-time scheduling policies run "in a blink" even on a resource-constrained platform. We discuss in the next section how these techniques help us achieve the three necessary functionality above.

4.4 Design and Implementation

In the following, Sections 4.4.1 and 4.4.2 describe our solutions to the slot allocation and start of round computation problems, respectively, assuming the set of streams is schedulable. Section 4.4.3 describes how we ensure this condition through online admission control.

Discrete-time model. Throughout the discussion, we consider a discrete-time model of LWB in which (*i*) each round is atomic and of unit length, and (*ii*) rounds start at an integer multiple of the unit length of a round, as illustrated in Figure 4.3. The reason for (*i*) is that the single MCU on today's low-power wireless platforms is responsible for both application processing and interacting with the radio. These radio interactions are time-critical and occur frequently during a Glossy flood [FZMT12]. Since each slot in a round consists of a Glossy flood, the MCU has very little time for application processing during a round. So to avoid interference, the application must release packets before a round and can only handle received packets after a round. We thus consider rounds atomic. Further, (*ii*) is beneficial in a practical LWB implementation. For example, it allows a node that just got out-of-sync to *selectively* turn on the radio in order to receive the next schedule (and thereby time-synchronize again with the host) rather than keeping the radio on all the time, which consumes more energy but is unavoidable if rounds can start at arbitrary times.

Besides the specific model, our analyses and algorithms in this section enjoy general validity. In particular, they also hold for streams with start times S_i, periods P_i, and deadlines D_i that are not integer multiples of the unit length of a round. For example, fractional packet release times are simply postponed to the next discrete time (by taking the ceiling), and fractional packet deadlines are preponed to the previous discrete time (by taking the floor). Thus, the atomicity of rounds does not prevent any packet from meeting its deadline. This is essential to the validity of our EDF-based scheduling policies, because preemptions in the execution of the underlying resource (i.e., the network, which we abstract as a single resource that runs on a single clock) can only occur at discrete times.

4.4.1 Slot Allocation

For ease of exposition, let us momentarily assume that the start time of the next round was already computed. We now need to determine the concrete schedule for that next round, which raises two questions: With B slots available per round, how many packets should we actually allocate? How should we prioritize pending packets of different streams?

Algorithms. To answer the first question, we note that delaying a packet by not sending it in an otherwise empty slot does not lead to improved schedulability or lower energy *overhead* in Blink. In the following round, the set of pending packets to schedule will be the same or larger, which can only worsen the overall schedulability. Furthermore, as explained in Section 4.2, the energy *overhead* in LWB over any given interval depends on the number of rounds within that interval, not on the number of allocated slots. Thus, it is best to allocate as many pending packets as possible to the available slots in any given round.

As described in Section 4.3, the approach we adopt in Blink allows us to abstract the entire network as a single resource that runs on a single clock. We can thus resort to uniprocessor scheduling policies to answer the second question. Among these, *earliest deadline first (EDF)* is provably realtime-optimal [Der74]; that is, if a set of streams can be scheduled such that all packets meet their deadlines, then EDF also meets all deadlines. This holds also for sets of streams demanding the full bandwidth, whereas other well-known policies, such as rate-monotonic (RM), may fail to meet all deadlines at significantly lower bandwidth demands [LL73]. Finally, as the packets' priorities are computed while the system executes, EDF

Table 4.1: Operations required on the set of streams Λ to efficiently implement EDF in Blink. *The key represents the absolute deadline of a stream's current packet.*

Operation	**Description**
INSERT(s)	Insert a new stream s into stream set Λ
DELETE(s)	Delete stream s from stream set Λ
DECREASEKEY(s, δ)	Propagate an increment of δ in the key of stream s in stream set Λ
FINDMIN()	Return a reference to the stream with the minimum key in stream set Λ
FIRST(t)	Position traverser t at the stream with the minimum key in stream set Λ
NEXT(t)	Advance traverser t to the stream with the next larger key in stream set Λ

can readily deal with dynamic changes in the set of streams [SAÅ+04], which is crucial when the set of streams changes because of, for example, varying application requirements or failures of nodes sourcing streams.

Using EDF in Blink entails allocating the next free data slot in a round to the packet whose deadline is closest to the start time of the round, until the round is full or there are no more pending packets. This seemingly simple logic, however, bears a significant run-time overhead [But05]. To implement EDF efficiently, one should maintain the streams in increasing order of absolute deadline while the latter is being updated from one packet to the next as they are allocated to slots. This overhead is one of the reasons why EDF is rarely used in real systems, such as operating system kernels [But05]. We describe next how we tackle this issue.

Design and implementation in Blink. Key to enabling EDF in Blink is the provision of a data structure that can efficiently maintain the current set of streams in order of increasing absolute deadline. A suitable data structure must support all operations required to manipulate the stream set Λ during EDF-based slot allocation, as listed in Table 4.1.

Besides operations to insert and delete a stream, EDF crucially requires a FINDMIN() operation to retrieve the stream with the earliest absolute deadline, which is to be served first. A *priority queue* is thus the most natural choice, where streams with smaller absolute deadline are given higher priority and hence served first. Moreover, after serving a stream s, the absolute deadline of s needs to be set to the deadline of its next packet. We therefore require a DECREASEKEY(s, δ) operation that propagates an increment of δ (typically the period of stream s) in the absolute deadline

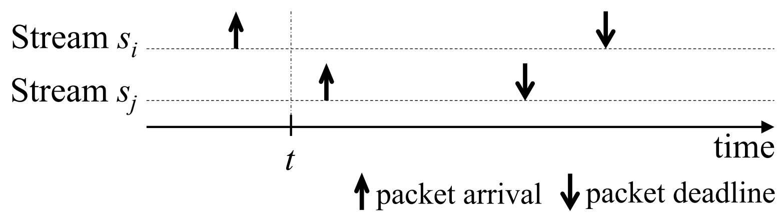


Figure 4.4: Example motivating EDF traversal of the set of stream during slot allocation. *Stream s_j has a higher priority than stream s_i because it has an earlier absolute deadline. Nevertheless, s_j requires no slot in the next round starting at time t, because it releases its next packet only after the start of the next round.*

of stream s in the priority queue. The result of this operation is that the priority of stream s is *decreased* relative to all other streams in the queue.

These four operations are supported by almost all standard priority queue data structures [Bro13]. Nevertheless, because the highest-priority stream returned by FindMin() may release its packet only *after* the start of the next round, as illustrated in Figure 4.4, we also require operations to perform an efficient *EDF traversal* of the stream set while *only* those streams with pending packets are updated. Specifically, it should be possible to position a *traverser t* at the highest-priority stream with First(t), and then to visit all streams in order of increasing absolute deadline through repeated Next(t) calls. During the EDF traversal, the priority of any stream t with a pending packet is updated using DecreaseKey(t, δ).

Finding a data structure that supports all required operations *efficiently* in terms of time and memory is challenging. A review of widely used and highly efficient priority queue data structures, ranging from the classical binary heap to red-black trees used in the Linux scheduler [lin], reveals that the EDF traversal (also known as *in-order traversal*) is the main culprit. In particular, updating a stream using DecreaseKey(t, δ) is likely to alter the relative ordering of streams, which triggers structural changes inside these data structures. Thus, a runtime stack or pointers (e.g., on a *threaded* binary tree) used for the traversal becomes invalid [Pfa], so the traversal must start anew after any such change, which becomes highly inefficient.

While looking for a practical solution to these challenges, we found that the following properties of our specific problem allow us to use a simple yet efficient data structure as the basis for our priority queue:

1. A stream's absolute deadline, henceforth referred to as the *key* of a stream, is a non-negative integer.

2. The key of a stream increases monotonically as it is being updated from one packet to the next.

3. The range of keys in the priority queue at any one time is bounded, as stated in the following theorem.

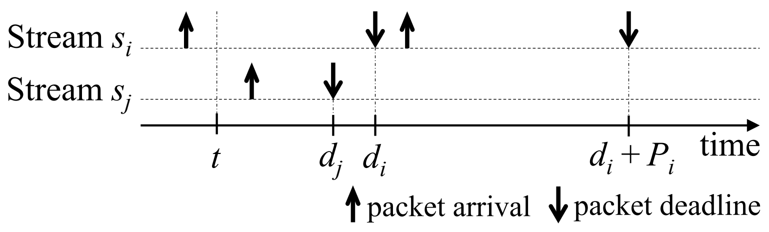


Figure 4.5: Illustration of the proof of Theorem 1.

Theorem 1. *With Λ denoting the set of streams, let $\overline{P}$ be an upper bound on the period P_i of any stream $s_i \in \Lambda$. Then, there are never more than $2\overline{P} - 1$ distinct keys (i.e., absolute deadlines) in the priority queue at any one time.*

Proof. Let d_i be the absolute deadline of stream s_i at some point in time; that is, d_i is the deadline of s_i's current packet. Since s_i's relative deadline D_i can be shorter than its period P_i, the current packet may not yet have arrived at this point in time. To determine the maximum number of distinct keys (i.e., absolute deadlines) in the priority queue at any one time, we must upper-bound the difference between the absolute deadlines of any two streams, that is, maximize $\Delta_{ij} = d_i - d_j$ for any two streams $s_i, s_j \in \Lambda$.

The value of Δ_{ij} is larger when packets with later deadlines are sent before packets with earlier deadlines, due to the order of packet arrival. Let us consider the example in Figure 4.5. Assume at time t the current packet of stream s_i with deadline d_i is sent, while the current packet of stream s_j with an earlier deadline $d_j < d_i$ is yet to be sent. This can happen if and only if s_i's packet arrives strictly before s_j's packet; that is, at time t, s_j's packet is yet to arrive. After sending the packet of stream s_i, its absolute deadline becomes $d_i + P_i$, while the absolute deadline of stream s_j is still d_j. Thus, we have $\Delta_{ij} = d_i + P_i - d_j$. What is the upper bound on Δ_{ij}? As s_i's packet has arrived by time t, we have $d_i \leq t + P_i$. Also, as s_j's packet has not yet arrived by time t, we have $d_j > t$. With these two conditions, we can establish the following bound

$$\Delta_{ij} = d_i + P_i - d_j < t + P_i + P_i - t \leq 2\overline{P}, \tag{4.1}$$

where $\overline{P}$ is an upper bound on the period of any stream. Since absolute deadlines are integers, the strict inequality in (4.1) implies that Δ_{ij} is at most $2\overline{P} - 1$. □

Given the three properties above, we can consider using a *monotone integer priority queue*. Similar observations apply to problems in discrete event simulation [Bro88] and image processing [FaSdAL04] but, to the best of our knowledge, have not been leveraged for real-time scheduling.

Specifically, we use a simple *one-level bucket queue* [Dia69] implemented as a circular array **B** of $2\overline{P}$ doubly-linked lists, as shown in Figure 4.6,

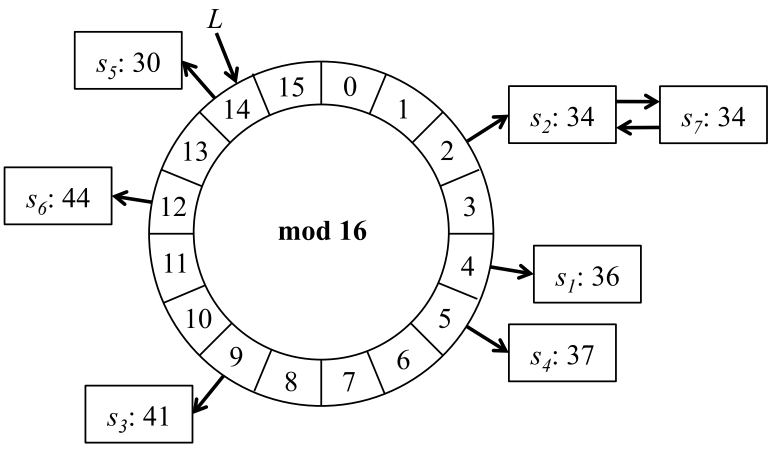


Figure 4.6: Illustration of a one-level bucket queue implemented as a circular array of $\mathbf{2\overline{P}}$ doubly-linked lists. *In this example, the largest period of any stream $\overline{P}$ is 8, so the queue consists of $2\overline{P} = 16$ buckets. The queue contains seven streams ordered by increasing key as follows: $s_5, s_2, s_7, s_1, s_4, s_3, s_6$. Operation* FindMin() *sets index L to 14, because this bucket contains the stream(s) with the smallest key in the queue.*

where $\overline{P}$ is an upper bound on the period of any stream. Stream s_i with key d_i is stored in $\mathbf{B}[d_i \bmod 2\overline{P}]$. Because a stream's relative deadline D_i is no longer than its period P_i, all keys in the bucket queue are always in the range $[d_{min}, d_{min} + 2\overline{P} - 1]$, where d_{min} is the smallest key currently in the queue. Thus, all streams in a bucket have the *same* key. As an example, the keys in the bucket queue shown in Figure 4.6 are in the range $[30, 45]$, and the two streams in bucket $\mathbf{B}[2]$ have the same key, namely 34.

Because buckets are implemented as doubly-linked lists, operations Insert(s), Delete(s), and DecreaseKey(s, δ) take constant time. Insert(s) inserts a stream s with key d into bucket $\mathbf{B}[d \bmod 2\overline{P}]$.[2] Delete($s$) removes stream s from the list containing it. DecreaseKey(s, δ) first performs a Delete(s) and then re-inserts stream s into bucket $\mathbf{B}[(d + \delta) \bmod 2\overline{P}]$.

We implement FindMin() using an index L, which is initialized to 0. If bucket $\mathbf{B}[L]$ is empty, FindMin() increments L (modulo $2\overline{P}$) until it finds the first non-empty bucket; otherwise, it returns the first stream on the list in bucket $\mathbf{B}[L]$. First(t) works similarly, using a second index I. Next(t) moves the traverser t to the next stream on the list in bucket $\mathbf{B}[I]$. When the end of the list is reached, Next(t) increments I (modulo $2\overline{P}$) until it finds the next non-empty bucket and then lets t point to the first stream on the list in bucket $\mathbf{B}[I]$. Unlike the vast majority of priority queues, this logic enables a smooth continuation of an EDF traversal despite stream updates. The three operations run in $O(2\overline{P})$ worst-case time.

[2] If $2\overline{P}$ is a power of two, the modulo operation required to compute the bucket index is equivalent to a bit-wise AND of d with $2\overline{P} - 1$ as the mask, which is highly efficient.

Algorithm 1 Perform EDF-based Slot Allocation

Input Bucket queue representing the current set of streams, where a smaller *absDeadline* implies that the stream has a higher priority, the start time of the next round t_{i+1}, and the upper bound on any stream's period $\overline{P}$.

Output Allocation of packets released by t_{i+1} in EDF order to at most B slots.

```
initialize slot counter $c$ to 0
position traverser $t$ at highest-priority stream using FIRST($t$)
$horizon = t.absDeadline + \overline{P} - 1$
while ($c < B$) and ($t.absDeadline \leq horizon$) do
    if $t.releaseTime \leq t_{i+1}$ then
        allocate a slot to stream $t$ and set $c = c + 1$
        $t.releaseTime = t.releaseTime + t.period$
        $t.absDeadline = t.absDeadline + t.period$
        update $t$'s priority with DECREASEKEY($t$, $t.period$)
    end if
    advance $t$ to next stream in EDF-order using NEXT($t$)
end while
```

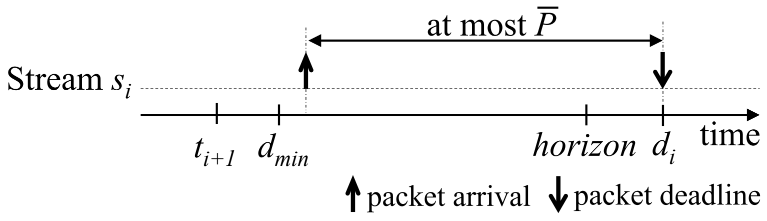


Figure 4.7: Illustration of the second termination criterion of the while loop in Algorithm 1. *Any stream s_i that has an absolute deadline d_i greater than the horizon $= d_{min} + \overline{P} - 1$ releases its current packet after the start of the next round at time t_{i+1}, and hence need not be considered for slot allocation in the next round.*

Using this dedicated priority queue data structure, Algorithm 1 shows the pseudocode to allocate as many pending packets as possible to the B available slots in the next round. The algorithm operates on the current set of streams, maintained in a bucket queue in order of increasing absolute deadline. Starting from the stream with the earliest absolute deadline, it visits streams in EDF-order through repeated NEXT(t) calls. When it sees a stream t with a pending packet, it allocates a slot to stream t and updates its priority within the queue using DECREASEKEY(t, $t.period$). The algorithm stops when all B data slots available in a round are allocated, or when it sees a stream with an absolute deadline larger than the *horizon*.

The second termination criterion using the horizon is needed because there may be less than B pending packets by the time the next round starts. Algorithm 1 determines the horizon initially, $horizon = d_{min} + \overline{P} - 1$, where d_{min} is the earliest absolute deadline of all streams at this time and

$\overline{P}$ is an upper bound on the period of any stream. As shown in Figure 4.7, the next round starts before d_{min}, that is, $t_{i+1} \leq d_{min} - 1$. So stream s_i with absolute deadline $d_i > horizon$ releases its current packet no earlier than

$$d_i - \overline{P} > d_{min} + \overline{P} - 1 - \overline{P} \geq t_{i+1}. \tag{4.2}$$

The strict inequality in (4.2) implies that stream s_i releases its current packet only after the start of the next round. Thus, stream s_i need not be considered for slot allocation in the next round. As Algorithm 1 visits streams in EDF-order, it can terminate when it sees the first such stream.

As described in the following sections, our bucket queue implementation underpins not merely the EDF-based slot allocation step, but rather *all* Algorithms 1, 2, 3, and 4 required for energy-efficient real-time scheduling in Blink. Despite enabling a smooth EDF traversal, the efficiency of our bucket queue implementation stems primarily from two key properties. First, DecreaseKey(s, δ) is a frequently used operation and at the same time extremely efficient in our design due to its constant, short running time. Second, the cost of searching for a non-empty bucket amortizes. To see why, we note that Next(t) needs to increment index I in the worst case $2\overline{P}-1$ times; however, the following n calls to Next(t) require *no* searching at all since all n streams are necessarily in $\mathbf{B}[I]$. With these implementation choices, we empower Blink to schedule hundreds of streams using EDF on resource-constrained devices even when the system is in a continuous state of change, which we show through experiments in Section 4.5.

4.4.2 Start of Round Computation

We now turn to the problem of computing the start time of the next round. To illustrate the problem, we use an example with $B = 5$ slots available per round and the following twelve streams with three distinct profiles: $3\langle 0, 5, 4\rangle$, $4\langle 2, 7, 5\rangle$, and $5\langle 1, 15, 12\rangle$. Figure 4.8 indicates the release times and deadlines of packets generated by these streams in the first 14 time units. We seek an answer to the following question: Using our EDF-based slot allocation policy from Section 4.4.1, when should a round start to meet all deadlines while minimizing energy (i.e., the number of rounds)?

Algorithms. One option, called *contiguous scheduling* (**CS**), is to start the next round immediately after the previous one has ended. **CS** offers the highest possible bandwidth and therefore necessarily meets all deadlines, provided the set of streams is schedulable. However, **CS** wastes energy, because it may trigger more rounds than necessary. Looking at Figure 4.8, we see that 8 out of the first 14 rounds are *empty* (i.e., contain only free slots) when using **CS**, causing unnecessary energy overhead.

Another possibility, referred to as *greedy scheduling* (**GS**), improves on

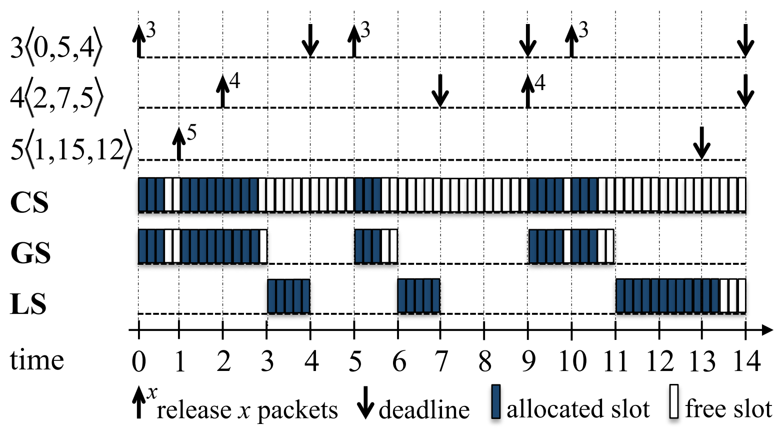


Figure 4.8: Example execution comparing the **CS**, **GS**, and **LS** policies. *CS and GS waste energy by scheduling more rounds than necessary. Instead, LS meets all deadlines while also minimizing energy (i.e., minimizing the number of rounds).*

CS by delaying the next round until there are one or more pending packets ready to be sent. **GS** is realtime-optimal just like **CS** as it schedules packets as soon as possible. Moreover, **CS** can reduce the energy consumption compared with **CS** in certain situations. For instance, in Figure 4.8 using **GS** results in only 6 rounds in the first 14 time units. However, there are still 8 free slots, raising the question whether we can do even better.

The crucial observation is that **GS** starts the next round no matter how "urgent" it is. If there was still some time until the earliest deadline of all pending packets, we could delay the next round further. Meanwhile, we could await more packet arrivals and thus allocate more slots in the next round. This strategy, however, may do more harm than good: Without knowing the future bandwidth demand, we may end up delaying the next round to a time where the number of packets to be sent is larger than the available bandwidth. This situation would inevitably cause deadline misses. To prevent this, we need to forecast the bandwidth demand.

A policy we call *lazy scheduling* (**LS**) is precisely based on this intuition. At the heart of **LS** is the notion of *future demand* $h_i(t)$ that quantifies the number of packets that must be sent (or *served*) between the end of round i and some future time t. The future demand includes all packets that have *both* their release time and deadline no later than time t, and have not been served until the end of round i, as captured by the following definition

$$h_i(t) = \sum_{j=1}^{n} \begin{cases} \left\lfloor (t - d_j)/P_j \right\rfloor + 1, & \text{if } d_j \le t \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

where P_j is the period and d_j is the absolute deadline of stream s_j (or more precisely, d_j is the deadline of s_j's current packet).

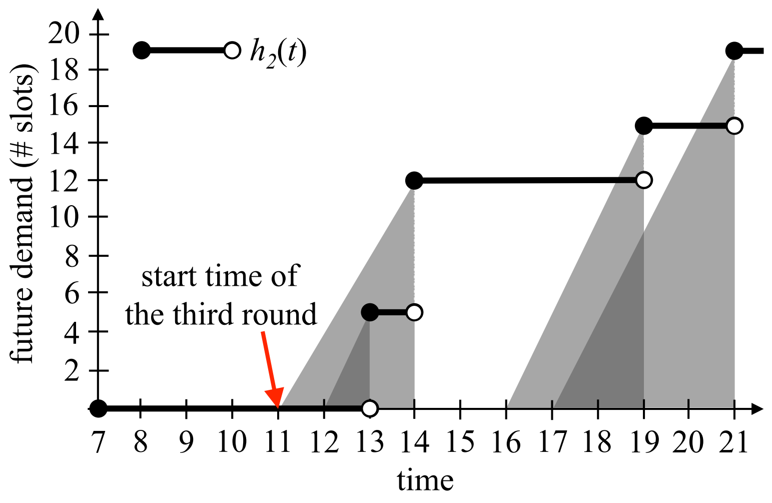


Figure 4.9: Graphical illustration of how **LS** computes the latest possible start time of the third round in Figure 4.8.

LS uses $h_i(t)$ to forecast the bandwidth demand and, based on this, computes the latest possible start time of the next round without missing any deadline. As a concrete example, say we want to compute the start time of the third round t_3 in Figure 4.8 using **LS**. We proceed as follows:

1. *Compute $h_2(t)$.* As illustrated in Figure 4.9, $h_2(13) = 5$ because $t = 13$ is the absolute deadline of the $5\langle 1, 15, 12\rangle$ streams whose packets are still pending at the end of the second round; $h_2(14) = h_2(13) + 7 = 12$, since $t = 14$ is the deadline of the 7 packets released by the other streams at $t = 9$ and $t = 10$; and so on.

2. *Determine a set of latest possible start times $\{t_3^i\}$.* For instance, $h_2(13) = 5$ packets must be served no later than time 13. With $B = 5$ slots available in each round, serving this demand takes $\lceil h_2(13)/B \rceil = 1$ round. Thus, we get a first latest possible start time $t_3^1 = 13 - 1 = 12$. We indicate this in Figure 4.9 by casting a shadow back on the time axis. Further, $h_2(14) = 12$ packets must be served before time 14, which takes $\lceil h_2(14)/B \rceil = 3$ rounds. So, a second latest possible start time of the third round is $t_3^2 = 14 - 3 = 11$. The same reasoning repeats, thus identifying more latest possible start times.

3. *Take the minimum of the computed latest possible start times as t_3.* Based on the reasoning in step 2., pushing the start of the third round beyond the beginning of the shady area at $\min\{t_3^i\} = 11$ in Figure 4.9 would cause deadline misses. Indeed, if we had served $h_2(13) = 5$ packets only between times 12 and 13, we would be left with $h_2(14) - h_2(13) = 7$ packets to serve between times 13 and 14, but these do not fit in a round with $B = 5$ slots. Alternatively, an earlier

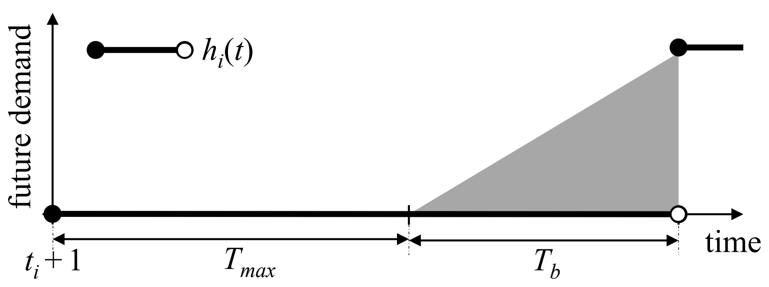


Figure 4.10: Illustration of how far **LS** needs to look into the future when computing the start time of the next round.

start time could, in the long run, lead to more rounds than needed, thereby wasting energy. Thus, the third round should start at $t_3 = 11$.

The example illustrates the main reasoning behind **LS**. Nevertheless, we still need to address two questions: Which times t do we really need to inspect in steps 1. and 2.? How far do we need to look into the future?

To answer the first question, we observe that the future demand $h_i(t)$ is a step function: the value of $h_i(t)$ increases only at times of deadlines, as visible in Figure 4.9. Thus, to speed up the computation, we can safely skip all intervals between steps where $h_i(t)$ is constant.

The answer to the second question involves two observations. First, as described in Section 4.2, we can delay the start of the next round by at most T_{max} time units after the start of the previous round since LWB requires to update the nodes' synchronization state sufficiently often to compensate for clock drift [FZMT12]. To find out whether we can indeed delay the next round by T_{max}, we need to evaluate $h_i(t)$ for at least T_{max} time units after the end of the previous round at $t_i + 1$, as illustrated in Figure 4.10. Second, we also have to consider any demand arising *after* $t = t_i+1+T_{max}$ that could possibly prevent us from delaying the next round by T_{max}. Thus, we must evaluate $h_i(t)$ for another T_b time units beyond $t = t_i + 1 + T_{max}$, also illustrated in Figure 4.10. The value of T_b is known as the *synchronous busy period* [Spu96]. Informally, this is the minimum time needed to serve the maximum demand that a given stream set can possibly create.[3] By looking up to $t = t_i + 1 + T_{max} + T_b$ into the future, we ultimately ensure that all deadlines are met.

The following theorem formally specifies the way **LS** computes the latest possible start time of the next round without missing any deadline.

[3]The maximum demand arises when all streams release their packets at the same time. **CS** essentially serves this demand "as fast as possible." T_b is then the time between the simultaneous arrival of packets from all streams and the first idle time where no packet is pending under **CS** scheduling.

Theorem 2. *Let T_b be the synchronous busy period of the stream set Λ, T_{max} the largest time by which the start of the next round can be delayed after the start of the previous one, and B the number of slots available in a round. Using **LS**, the start time of each round t_i for all $i = 0, 1, \ldots$ is computed as*

$$t_{i+1} = \min(t_i + T_{max}, T_i), \tag{4.4}$$

where $t_0 = -1$ and T_i is given by

$$T_i = \min_{t \in \mathcal{D}_i} \left(t - \left\lceil \frac{h_i(t)}{B} \right\rceil \right). \tag{4.5}$$

$\mathcal{D}_i$ denotes the set of deadlines in the interval $[t_i + 1, t_i + T_{max} + T_b + 1]$ of packets that are unsent until the end of round i; $h_i(t)$ is the future demand as per (4.3).

Proof. Because of time synchronization constraints imposed by the LWB communication support [FZMT12], the start of the next round at time t_{i+1} can be delayed at most T_{max} after the start of the previous round at time t_i. This implies the first component of the min-operation in (4.4).

We now show that the second component of the min-operation in (4.4) ensures that all packets meet their deadlines. The number of packets that must be sent between the end of round i and some time $t \geq t_i + 1$ is given by the future demand $h_i(t)$. The available bandwidth in the interval $[t_{i+1}, t]$ is $B(t - t_{i+1})$, where B is the number of slots available per round. To ensure that all packets meet their deadlines, the future demand $h_i(t)$ must not exceed the available bandwidth for any time $t \geq t_i + 1$, that is,

$$B(t - t_{i+1}) \geq h_i(t). \tag{4.6}$$

Dividing both sides by the positive quantity B,

$$t - t_{i+1} \geq h_i(t)/B. \tag{4.7}$$

Since $m \geq x$ if and only if $m \geq \lceil x \rceil$ for any integer m and real number x,

$$t - t_{i+1} \geq \lceil h_i(t)/B \rceil. \tag{4.8}$$

Rearranging terms,

$$t_{i+1} \leq t - \lceil h_i(t)/B \rceil. \tag{4.9}$$

In particular,

$$t_{i+1} \leq \min_{t \geq t_i + 1 \wedge h_i(t) > 0} \left(t - \lceil h_i(t)/B \rceil \right). \tag{4.10}$$

The min-operation in (4.10) is to be performed for every time t larger than $t_i + 1$ at which the future demand $h_i(t)$ is greater than zero. We can restrict this in two ways. First, we need to apply the min-operation only at every time t in the interval $[t_i + 1, t_i + T_{max} + T_b + 1]$, where T_b

is the synchronous busy period of the stream set Λ. We prove this by contradiction. Let for some $\hat{t} > t_i + T_{max} + T_b + 1$,

$$\hat{t} = \underset{t \geq t_i + 1 \wedge h_i(t) > 0}{\arg\min} \left(t - \lceil h_i(t)/B \rceil\right). \tag{4.11}$$

Let the quantity $\hat{t} - \left\lceil h_i(\hat{t})/B \right\rceil$ be equal to the start time of the next round t_{i+1} and strictly less than $t_i + T_{max}$,

$$t_{i+1} = \hat{t} - \left\lceil h_i(\hat{t})/B \right\rceil < t_i + T_{max}. \tag{4.12}$$

Rearranging terms,

$$\left\lceil h_i(\hat{t})/B \right\rceil = \hat{t} - t_{i+1}. \tag{4.13}$$

Since $\lceil x \rceil = m$ if and only if $m - 1 < x \leq m$ for any integer m and real number x,

$$\hat{t} - t_{i+1} - 1 < h_i(\hat{t})/B. \tag{4.14}$$

Multiplying both sides by the positive quantity B,

$$(\hat{t} - t_{i+1} - 1)B < h_i(\hat{t}). \tag{4.15}$$

We can interpret (4.15) as follows. The future demand $h_i(\hat{t})$ exceeds the bandwidth available in the interval $[t_{i+1} + 1, \hat{t}]$. This means that if one were to contiguously serve a demand as large as $h_i(\hat{t})$, the required time would exceed the length of the interval $[t_{i+1} + 1, \hat{t}]$. We can thus consider interval $[t_{i+1} + 1, \hat{t}]$ a *busy period* of length $\hat{t} - t_{i+1} - 1 > \hat{t} - (t_i + T_{max}) - 1 > T_b$, because we have $t_{i+1} < t_i + T_{max}$ according to (4.12). However, T_b is the length of the synchronous busy period, which is the longest possible busy period [Spu96]. This contradicts the supposition on the existence of $\hat{t}$.

Second, we need to perform the min-operation in (4.10) only at times when $h_i(t)$ has discontinuities. In fact, $h_i(t)$ is a right-continuous function with discontinuities only at times that coincide with packet deadlines. Thus, we can restrict the domain of the min-operation to all deadlines $\mathcal{D}_i$ in the interval $[t_i + 1, t_i + T_{max} + T_b + 1]$ of packets that are unsent until the end of round i. Since (4.10) yields the largest possible t_{i+1} in the case of equality, we obtain the second component of the min-operation in (4.4)

$$T_i = \min_{t \in \mathcal{D}_i} \left(t - \lceil h_i(t)/B \rceil\right). \tag{4.16}$$

Finally, because EDF is realtime-optimal [Der74, LL73], the necessary condition in (4.16) is also a sufficient one. $\square$

We are now in the position to state the main result about the **LS** policy, showing that it meets the two goals set out in Section 4.3.

Theorem 3. *The **LS** policy is real-time optimal and minimizes the network-wide energy consumption within the limits set by the LWB communication support.*

Proof. A *schedule* $\mathbf{S}$ specifies for each round i its start time t_i and the set of packets to be sent in the round. Let $\mathbf{S}^{\mathbf{LS}}$ denote the schedule computed by **LS**. We prove this theorem by contradiction; that is, we show that there cannot be any schedule $\mathbf{S}' \neq \mathbf{S}^{\mathbf{LS}}$ such that $\mathbf{S}'$ is realtime-optimal and some round starts *later* in $\mathbf{S}'$ than in $\mathbf{S}^{\mathbf{LS}}$. If we show this, it follows that amongst all realtime-optimal schedules, $\mathbf{S}^{\mathbf{LS}}$ delays the start of each round the most and thus minimizes the network-wide energy consumption using LWB.

Let $t_i^{\mathbf{LS}}$ and t_i' denote the start times of the i-th round in $\mathbf{S}^{\mathbf{LS}}$ and $\mathbf{S}'$, and let $h_i^{\mathbf{LS}}$ and h_i' denote the future demands after the end of the i-th round in $\mathbf{S}^{\mathbf{LS}}$ and $\mathbf{S}'$, respectively.

Assume some round in $\mathbf{S}'$ starts later than in $\mathbf{S}^{\mathbf{LS}}$. Let the m-th round be the first such round, that is,

$$m = \min\{i \mid t_i' > t_i^{\mathbf{LS}}\}. \tag{4.17}$$

In $\mathbf{S}^{\mathbf{LS}}$, the m-th round starts at $t_m^{\mathbf{LS}}$ since, according to Theorem 2, at least one of the following two conditions holds:

1. $t_m^{\mathbf{LS}} - t_{m-1}^{\mathbf{LS}} = T_{max}$, where T_{max} is the largest interval between the start of consecutive rounds supported by LWB [FZMT12],

2. $h_{m-1}^{\mathbf{LS}}(t^*) > B(t^* - t_m^{\mathbf{LS}} - 1)$ for some time $t^* > t_m^{\mathbf{LS}}$.

Assume condition 1. holds. Then, from the definition of m in (4.17), we have $t_m' - t_{m-1}' > t_m^{\mathbf{LS}} - t_{m-1}^{\mathbf{LS}} = T_{max}$. This violates the constraint that the time between the start of consecutive rounds in $\mathbf{S}'$ does not exceed T_{max}.

Assume condition 2. holds. Then, the interval $[t_m^{\mathbf{LS}}, t^*]$ is a busy period in $\mathbf{S}^{\mathbf{LS}}$, so the number of packets sent in this interval, denoted $\eta^{\mathbf{LS}}(t_m^{\mathbf{LS}}, t^*)$, is lower-bounded as

$$\eta^{\mathbf{LS}}(t_m^{\mathbf{LS}}, t^*) > B(t^* - t_m^{\mathbf{LS}} - 1). \tag{4.18}$$

On the other hand, since $t_m' > t_m^{\mathbf{LS}}$ according to the definition of m in (4.17), the number of packets transmitted in $\mathbf{S}'$ in the interval $[t_m', t^*]$, denoted $\eta'(t_m', t^*)$, is upper-bounded as

$$\eta'(t_m', t^*) \leq B(t^* - t_m') \leq B(t^* - t_m^{\mathbf{LS}} - 1). \tag{4.19}$$

From (4.18) and (4.19) follows a strict inequality

$$\eta^{\mathbf{LS}}(t_m^{\mathbf{LS}}, t^*) > \eta'(t_m', t^*). \tag{4.20}$$

Algorithm 2 Compute Start Time of Next Round According to **LS**

```
Input  A bucket queue based copy of the current set of streams, where a smaller
    absDeadline means that the stream has a higher priority, the start time of the
    current round $t_i$, and the synchronous busy period $T_b$.
Output  The start time of the next round $t_{i+1}$ according to the LS policy.
  initialize futureDemand to 0 and minSlack to $\infty$
  set $s$ to highest-priority stream using $s$ = FindMin()
  $t = s.absDeadline$
  while $t \leq t_i + T_{max} + T_b + 1$ do
      $futureDemand = futureDemand + 1$
      $s.absDeadline = s.absDeadline + s.period$
      update priority of $s$ using DecreaseKey($s$, $s.period$)
      set $s$ to highest-priority stream using $s$ = FindMin()
      if $s.absDeadline > t$ then
          $minSlack = \min((t - t_i)B - futureDemand, minSlack)$
      end if
      $t = s.absDeadline$
  end while
  $t_{i+1} = t_i + \min(\lfloor minSlack/B \rfloor, T_{max})$
```

We also know that $\eta^{\mathbf{LS}}(0, t_m^{\mathbf{LS}}) \geq \eta'(0, t'_m)$, because each round $1, 2, \ldots, m-1$ starts no earlier in $\mathbf{S}^{\mathbf{LS}}$ than in $\mathbf{S}'$, and in $\mathbf{S}^{\mathbf{LS}}$ as many pending packets as possible are sent in each round. Combining this with (4.20), we have

$$\eta^{\mathbf{LS}}(0, t^*) > \eta'(0, t^*). \tag{4.21}$$

This shows that $\mathbf{S}^{\mathbf{LS}}$ tightly meets all deadlines at time t^*, while sending more packets than $\mathbf{S}'$. Since $\mathbf{S}^{\mathbf{LS}}$ prioritizes packets using EDF, which is realtime-optimal [Der74, LL73], $\mathbf{S}'$ necessarily misses a deadline at or before time t^*. This contradicts the assumption that $\mathbf{S}'$ is realtime-optimal.

For either condition that impacts the choice of $t_m^{\mathbf{LS}}$, we have shown that the assumptions on $\mathbf{S}'$ are contradicted. □

Design and implementation in Blink. The primary systems challenge in leveraging **LS**'s optimality is to efficiently compute the future demand $h_i(t)$. The corresponding expression in (4.3) is obtained by applying well-known concepts from the real-time literature [SRS98]. We implemented this conventional method on a TelosB [PSC05] and observed prohibitive running times due to many time-consuming divisions. This behavior is expected also on other resource-constrained platforms that lack hardware support for accelerating divisions. Experiments in Section 4.5.3 show that the approach we describe next can outperform the conventional method even on recent, powerful platforms, including a 32-bit ARM Cortex-M4.

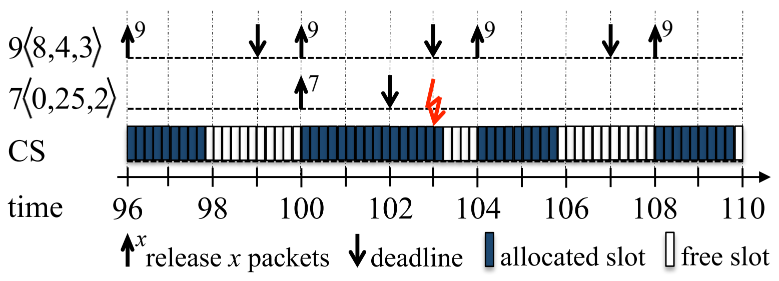


Figure 4.11: Example of a stream set that is not schedulable. *The streams demand 16 slots between times 100 and 103, but there are only 15 slots available in this interval.*

Instead of explicitly computing costly divisions, we can determine $h_i(t)$ by performing efficient operations on the bucket queue of streams. Based on this idea, Algorithm 2 shows the pseudocode to compute the start time of the next round t_{i+1} according to Theorem 2. The algorithm operates on a deep copy of the current set of streams, maintained in a bucket queue in order of increasing absolute deadline. It fictitiously serves streams in EDF-order (as if it would allocate slots to pending packets), using variable *futureDemand* to keep track of the number of streams it has served thus far. The algorithm also maintains a variable *minSlack*, which ultimately determines how far we can delay the start of the next round.

By avoiding divisions and using our bucket queue implementation, our implementation of Algorithm 2 can achieve several-fold speed-ups over the analytic method. As described in Appendix 4.A, we use the same techniques to efficiently compute the duration of the synchronous busy period T_b, which is crucial to **LS** and admission control discussed in the next section, and demonstrate similar speed-ups over an existing iterative method [SRS98]. Experimental results in Section 4.5.3 indicate that these improvements in processing time are instrumental to the viability of EDF-based scheduling in Blink on certain low-power wireless devices.

4.4.3 Admission Control

So far we assumed the stream set is schedulable, yet this is not always the case. As Figure 4.11 exemplifies, for $B = 5$ slots available per round, the set consisting of $9\langle 8, 4, 3\rangle$ and $7\langle 0, 25, 2\rangle$ streams is not schedulable. The streams require altogether 16 slots in the interval between time 100 and time 103; however, there are only 15 slots available in this time interval, which causes one packet to miss its deadline. We describe next how we prevent such situations by checking *prior* to the addition of a new stream whether the resulting set of streams is still schedulable.

Algorithms. As illustrated by the example above, deadlines are missed if, over some time interval, the demand exceeds the available bandwidth. As explained before, the **CS** policy offers the highest possible bandwidth. Therefore, admission control under all scheduling policies discussed in Section 4.4.2 amounts to checking if **CS** can meet all deadlines.

We must perform this check over an interval in which the demand is highest. The intuition is that if the available bandwidth is sufficient even in this extreme situation, we can safely admit the new stream. Precisely identifying when this situation occurs is, however, non-trivial, in that the different start times and periods of the streams may defer this situation until some arbitrary time. For example, for the stream set in Figure 4.11, it is not until time $t = 100$ that an interval of highest demand begins.

To tackle this problem, we deliberately create an interval of maximum demand by forcing all streams to release a packet at time $t = 0$. Using the concept of synchronous busy period T_b, we then check if **CS** can meet all deadlines in the interval $[0, T_b]$. From this intuition follows a theorem, whose proof descends from existing results [Spu96]:

Theorem 4. *For a set of streams Λ with arbitrary start times S_i, let Λ' be the same set of streams except all start times are set to zero. With B slots available in each round, Λ is schedulable if and only if*

$$\forall t \in \mathcal{D},\ h_0(t) \le t \times B, \tag{4.22}$$

where $\mathcal{D}$ is the set of deadlines in the interval $[0, T_b]$ of packets released by streams in Λ', $h_0(t)$ is the number of packets that have both release time and deadline in $[0, t]$, and $t \times B$ is the bandwidth available in the interval $[0, t]$.

Design and implementation in Blink. An efficient implementation of Theorem 4 faces the same challenges as mentioned before in Section 4.4.2. The closed-form expression found in the literature [But11]

$$h_0(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor \tag{4.23}$$

involves many costly divisions, so using (4.23) can result in a performance hog on certain resource-constrained platforms.

For this reason, we perform admission control again by performing efficient bucket queue operations instead of costly divisions. Algorithm 3 takes as input a deep copy of the current set of streams, including the new stream to be admitted, and the new synchronous busy period T_b. All streams start at time $t = 0$ to trigger an interval of maximum demand, so the absolute deadline of each stream s_i is initialized to the relative deadline D_i. Using a bucket queue to keep streams in EDF-order, the algorithm repeatedly updates the absolute deadline of the highest-priority stream

Algorithm 3 Perform Admission Control

```
Input  A bucket queue based copy of the current set of streams including the new
   stream, with $s.absDeadline$ initialized to $D_i$ for all streams $s_i$ (smaller $absDeadline$
   implies higher priority), the utilization as well as the deadline-based utilization
   of that stream set, and the new synchronous busy period $T_b$.
Output  Whether the new stream is to be admitted or rejected.
  if utilization exceeds 1 then
      return "reject"
  end if
  if deadline-based utilization does not exceed 1 then
      return "admit"
  end if
  initialize $demand$, $availableBandwidth$, and $t$ to 0
  while $demand \leq availableBandwidth$ do
      set $s$ to highest-priority stream using $s$ = FindMin()
      if $s.absDeadline > t$ then
          if $s.absDeadline > T_b$ then
              return "admit"
          end if
          $availableBandwidth = t \times B$
          $t = s.absDeadline$
      end if
      $demand = demand + 1$
      $s.absDeadline = s.absDeadline + s.period$
      update priority of $s$ using DecreaseKey($s$, $s.period$)
  end while
  return "reject"
```

as if it were executing **CS**. In doing so, the algorithm keeps track of the number of deadlines seen until time t, which correspond to the demand. If this quantity exceeds the *availableBandwidth* in the interval $[0, t]$ for any t in the interval $[0, T_b]$, the new stream cannot be admitted.

Algorithm 3 contains two optimizations that help improve its average-case performance. First, it checks if the end of interval $[0, T_b]$ has been reached and updates the *availableBandwidth* only when t has advanced. This avoids unnecessary processing when multiple packet deadlines coincide. Second, the algorithm performs two simple checks before the loop. The new stream can be rejected without further processing if the *utilization*, defined as $\frac{1}{B}\sum_{i=1}^{n}\frac{1}{P_i}$, exceeds one [LL73]. Further, since $D_i \leq P_i$ for any stream s_i, the new stream can be admitted if the *deadline-based utilization*, defined as $\frac{1}{B}\sum_{i=1}^{n}\frac{1}{D_i}$, does not exceed one [SRS98]. We update both types of utilizations as streams are added and removed at runtime.

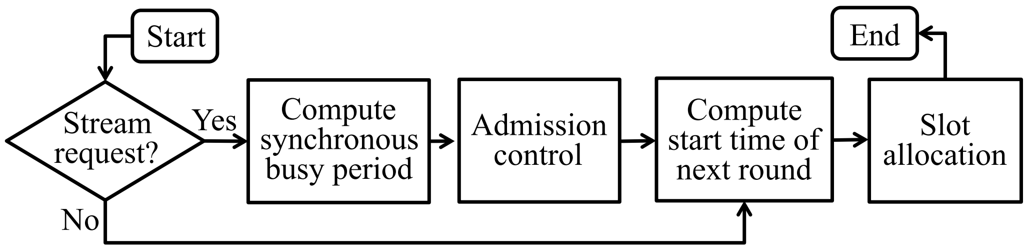

Start
Stream request?
Yes
No
Compute synchronous busy period
Admission control
Compute start time of next round
Slot allocation
End

Figure 4.12: Main steps in Blink's real-time scheduler executed by the host at the end of each round (see Figure 4.1). *The algorithm to compute the start time of the next round depends on whether **CS**, **GS**, or **LS** is used. In case of **LS**, the synchronous busy period T_b is carried over to subsequent rounds unless the set of streams changes.*

4.4.4 Blink in Practice

At the end of each round, the algorithms described above unfold as shown in Figure 4.12. With a pending request for a new stream s, the scheduler computes the (new) synchronous busy period for the stream set $\Lambda \cup \{s\}$ and checks if s can be admitted. In any case, the scheduler computes the start time of the next round and then allocates slots to pending packets. In the worst case, a single scheduler execution needs to proceed through all four steps in Figure 4.12. Experiments in Section 2.5 show that our implementation can schedule hundreds of streams with a wide range of realistic bandwidth demands within the time bounds of Blink prototype.

We implement the processing in Figure 4.12 in a Blink prototype on top of the Contiki [DGV04] operating system. Our prototype targets the TelosB platform available on large public testbeds, which comes with a MSP430 MCU. We use the default settings in LWB [FZMT12], including the fixed 1 second duration of a round that corresponds to the time unit used throughout Section 4.4. We set the number of data slots in a round to $B = 51$, which leaves about 100 ms to compute the schedule at the end of a round. We slightly modify the time synchronization mechanism of LWB to support frequently varying intervals between rounds in Blink.

4.5 Evaluation

In this section, we evaluate Blink along four lines: (*i*) the ability to deal with dynamic changes in the set of streams, (*ii*) the level of real-time service provided to applications in terms of meeting packet deadlines, (*iii*) the energy efficiency of our scheduling policies, and (*iv*) the scalability properties of our implementation of the **LS** policy. We find that:

- Blink promptly adapts to dynamic changes in the set of streams without unnecessarily increasing energy consumption.

- Blink meets on average 99.97 % of the packet deadlines; misses are entirely due to unavoidable packet loss in a wireless setting.
- **LS** improves the energy consumption by up to a factor of 2.5× over **CS** and **GS**, depending on the profiles of the streams.
- Using our bucket queue implementation, our implementation of the **LS** policy executes up to 4.1× faster than a conventional **LS** implementation on state-of-the-art MCUs.

We note that dynamic changes in the network state because of, for example, link quality changes, wireless interference, node mobility, and node failures, are already effectively dealt with by the underlying LWB communication support, as experimentally shown in [FZMT12].

4.5.1 Adaptivity to Changes in the Set of Streams

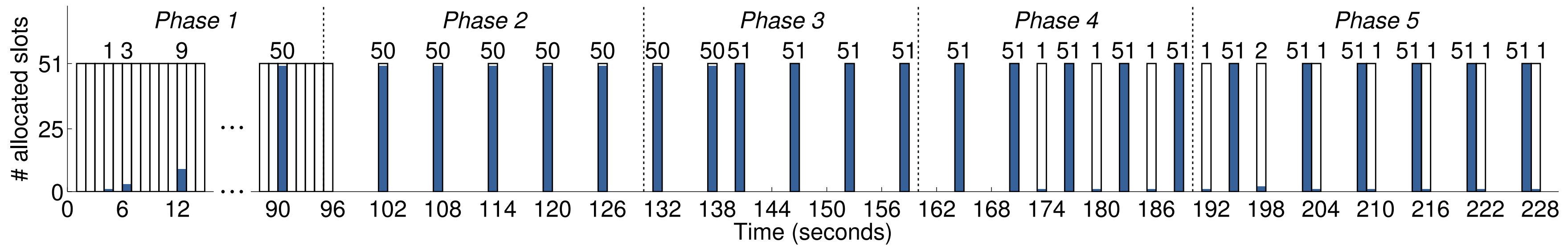
In our first experiment, we demonstrate how Blink dynamically adapts to runtime changes in the set of streams. We use 29 TelosB nodes on the FlockLab testbed [LFZ+13b], which has a diameter of 5 hops. One node acts as the host to run the scheduler, and three randomly chosen nodes serve as destinations. The remaining 25 nodes act as sources generating $2\langle 0, 6, 6\rangle$ streams each. Eventually, we also let two of these sources request and update a third and then a fourth stream with different profiles.

Execution. Figure 4.13(a) shows the number of slots allocated in each round in the first 4 minutes of the experiment, while Figure 4.13(b) shows a breakdown of the execution time of the **LS** scheduler in each round.

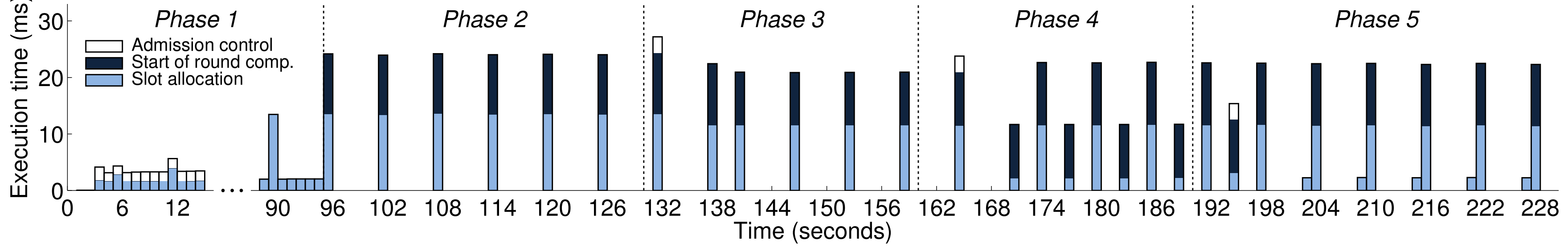
In *Phase 1*, the system is bootstrapping after powering on the devices. Blink schedules rounds contiguously to enable all nodes to quickly time-synchronize and submit their stream requests. This happens for the very first time after 3 seconds, as visible in Figure 4.13(b) from the increase in processing time to perform admission test and slot allocation. During the following rounds, the host gradually receives all initial stream requests and, consequently, admission test and slot allocation take longer.

In *Phase 2*, because no new stream request has recently arrived, Blink adapts its functioning to the normal operation and starts to dynamically compute the start time of rounds based on the **LS** policy. Figure 4.13(b) shows that this takes about 10 ms. Given the profiles of admitted streams, Blink schedules a round every 6 seconds, postponing rounds until right before the packets' deadlines, which minimizes energy consumption.

At the beginning of *Phase 3*, a request for a new stream $\langle 0, 6, 3\rangle$ arrives. Admission control executes as visible in Figure 4.13(b) at time $t = 131$



(a) Number of allocated slots in each round, for $B = 51$ available slots per round.



(b) Breakdown of scheduler execution time in each round.

Figure 4.13: A real trace of Blink dynamically scheduling streams with varying real-time requirements on the FlockLab testbed. *After bootstrapping the network, Blink uses* ***LS*** *to save energy, while meeting all deadlines of the changing stream sets.*

seconds. Blink admits the new stream and accounts for it starting from $t = 140$ seconds. Rounds are scheduled again every 6 seconds and with all $B = 51$ available slots allocated. Unlike most existing systems, Blink has accommodated a new stream without jeopardizing the existing ones, and still maintains the minimum energy consumption. In WirelessHART, for example, changes in the set of streams are more disruptive, likely affecting existing streams [ZSJ09], and thus take much longer to accommodate.

In *Phase 4*, another requests for a stream $\langle 0, 6, 6 \rangle$ arrives and passes admission control. Blink allocates the first slot to the new stream starting from $t = 170$ seconds. However, now there are 52 pending packets, 1 more than the $B = 51$ available slots. Due to this, Blink schedules the following rounds every 3 seconds, with the number of allocated slots alternating between 51 and 1. This shows that Blink seamlessly copes with dynamic changes in the stream set that result in drastic changes in its operation.

In *Phase 5*, the node that requested a second stream in *Phase 3* extends that stream's deadline from 3 to 6 seconds. Thus, the current 52 streams all have the same deadline and period. Again, because 52 packets do not fit into a single round, Blink schedules a complete round with 51 allocated slots 2 seconds before the packets' deadlines, followed by another round for the remaining packet. This shows that a seemingly minor change in the real-time requirements of one stream can have a significant impact on how rounds unfold over time, which Blink handles quickly and effectively.

4.5.2 Meeting Packet Deadlines at Minimum Energy Cost

We next evaluate Blink's ability to meet packet deadlines and the energy consumption when using the **LS** policy compared with **CS** and **GS**.

Metrics and settings. We use two key performance metrics in real-time low-power wireless [SXLC10]. First, the *deadline success ratio* measures the fraction of packets that meet their deadlines, indicating the level of real-time service provided to the application. We compute this figure based on sequence numbers embedded into packets and timestamps taken at both communication end points. Second, the *radio duty cycle* is defined as the fraction of time a node has the radio on, which is widely used as a proxy for energy consumption [GFJ+09]. This metric indicates the energy cost Blink incurs to provide a given level of real-time service. We measure radio duty cycles in software using Contiki's power profiler [DOTH07].

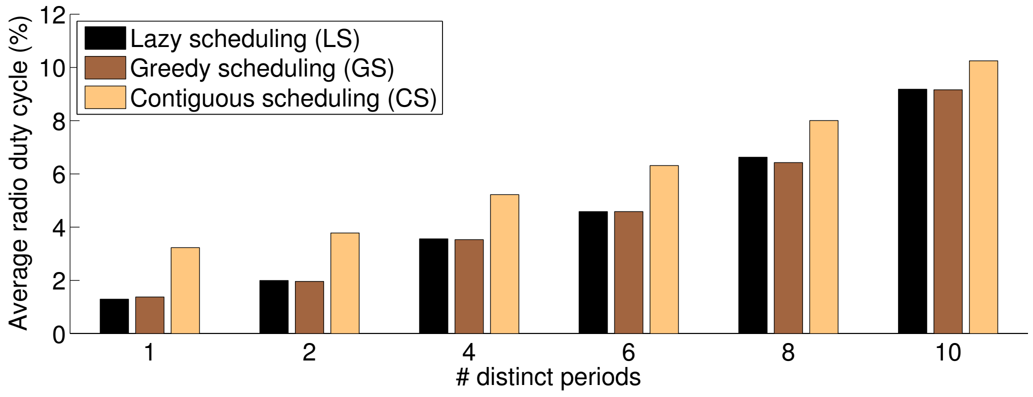
We conduct several experiments with 94 TelosB nodes on the w-iLab.t testbed [BVJ+10], which has a diameter of 6 hops. We let 90 nodes act as sources, one as the host, and three as sinks, mimicking a scenario with multiple controllers or actuators [PL10]. Each source has two streams, so there are in total 180 streams generating packets with a 10-byte payload.

We conduct two types of experiments. First, we set all starting times S_i to zero and vary the number of distinct periods across different runs, resembling configurations used when combining primary and secondary control [Oga01]. In this way, we generate varying demands between 2.9% and 19.4% of the available bandwidth. Then, we set the period P_i of all streams to 2 minutes and vary the number of distinct starting times from 1 to 120. This emulates situations where, for example, sources are added to a running system over time with no explicit alignment in the packet release times [PL10]. Deadlines are equal to periods in all runs. Each run lasts for 50 minutes. To be fair across runs, we give nodes enough time to submit their stream requests and start measuring only after 20 minutes.

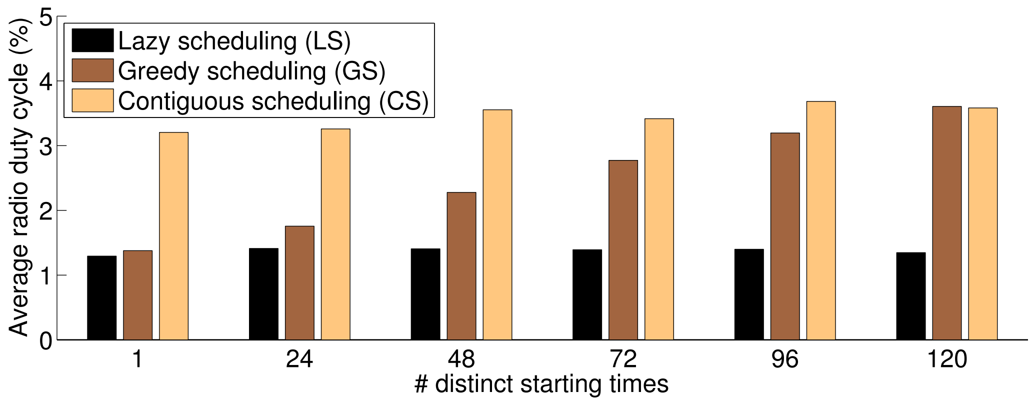
Results. The average deadline success ratio is 99.97 %, with a minimum of 99.71 % in one of the 36 runs. These figures are noteworthy in at least two respects. First, most modern control applications, including the ones we mentioned in Section 4.1, can and do tolerate such small fraction of packets not meeting their deadlines [SSF$^+$04]. We thus demonstrate that Blink can effectively operate in several of these scenarios. Moreover, we verify that deadline misses are entirely due to packet losses over the wireless channel, a phenomenon that is orthogonal to packet scheduling (see Chapter 3) and that cannot be completely avoided. Overall, these experiments confirm the reasoning and theoretical results from Section 4.4.

Looking at radio duty cycle, Figure 4.14(a) shows the average across all 94 nodes for **LS**, **CS**, and **GS**. We see that the energy costs generally increase with the number of distinct periods, since the bandwidth demand increases as well. Differences among the policies stem from scheduling fewer rounds. **LS** and **GS** perform similarly here: Since all streams start at the same time and because of the choice and distribution of periods, **LS** has little opportunity to spare more rounds than **GS**. Nonetheless, both **LS** and **GS** significantly improve over **CS**—they need 2.5× less energy when all streams have the same period. This gap shrinks to 1.2× with 10 distinct periods, mostly because the energy overhead of unnecessary rounds plays a less important role at higher bandwidth demands.

Figure 4.14(b) shows the average radio duty cycle as the number of distinct starting times increases. The bandwidth demand is constant, so **CS** consumes the same energy across all settings. This time, however, **LS** and **GS** perform differently. The energy costs of **GS** increase as the number of distinct starting times increases, because packets are released at increasingly different times and hence **GS** schedules more and more rounds. **LS**, instead, benefits from aggregating packets over subsequent release times and sending them in the same round. As a result, the energy costs of **LS** remain low and constant across all settings, whereas the energy costs of **GS** increase and eventually approach those of **CS**.



(a) Varying number of distinct periods.



(b) Varying number of distinct start times.

Figure 4.14: Average radio duty cycle of Blink with **LS**, **GS**, and **CS** across 94 nodes and 6 hops in the w-iLab.t testbed. *Depending on the stream set, **LS** achieves up to 2.5× reduction in energy consumption compared with the **GS** and **CS** policies.*

Overall, these results show that **LS** is most energy-efficient irrespective of the stream set, achieving severalfold improvements over **GS** and **CS** in some settings. However, in settings with only a few distinct periods and starting times, **GS** might also be an option from an energy standpoint and also because it reduces latency by sending packets as soon as possible.

4.5.3 Scheduler Execution Time

Finally, we look at the scalability of our implementation of the **LS** policy. This is an important aspect since the scheduler's execution time is a key parameter affecting the available bandwidth. As detailed in Section 4.3 and shown in Figure 4.1 (B), the longer the scheduling takes, the fewer data slots are available given the fixed duration of a round.

Method: overview. In the worst case, a single scheduler execution must proceed through all four steps in Figure 4.12. A careful analysis of the respective Algorithms 1, 2, 3, and 4 reveals that the execution time of the **LS** scheduler increases with the number of streams n, the largest possible period of a stream $\overline{P}$, the total bandwidth demand of the streams denoted u, and the synchronous busy period T_b of the set of streams.

Precisely quantifying how the combination of these four parameters determines the total running time of the **LS** scheduler is non-trivial. We therefore opt for an empirical approach that confidently approximates the worst-case execution time of the **LS** scheduler. Specifically, n and $\overline{P}$ are application-specific, yet the memory available on a given platform determines their maximum value. For example, in our Blink prototype, memory scales linearly with n and $\overline{P}$, and for $\overline{P} = 255$ seconds it supports up to $n = 200$ streams on a TelosB. Let N denote the maximum number of streams supported for a given upper bound the streams' period $\overline{P}$. Quantities u and T_b, instead, vary depending on the streams' profiles. Thus, to approximate the worst-case execution time of the **LS** scheduler, we compute, for a given bandwidth demand u, N stream profiles with periods no larger than $\overline{P}$ that maximize the synchronous busy period T_b.

Method: determining worst-case stream profiles. To this end, we solve two integer linear programs (ILPs). The decision variables are the periods of the streams, which are integers from the interval $[1, \overline{P}]$. We encode the periods through variables $x_1, x_2, \ldots, x_{\overline{P}}$, where x_i is the number of streams with period i. The start times of streams are zero, and deadlines are equal to periods. First, we minimize the bandwidth demand u of N streams, given their synchronous busy period T_b, by solving the following ILP:

$$\begin{aligned}
\underset{\{x_1, x_2, \ldots, x_{\overline{P}}\}}{\text{minimize}} \quad & \sum_{i=1}^{\overline{P}} x_i / i \\
\text{subject to} \quad & \sum_{i=1}^{\overline{P}} x_i = N \\
& \sum_{i=1}^{\overline{P}} x_i \lceil 1/i \rceil > B \\
& \sum_{i=1}^{\overline{P}} x_i \lceil 2/i \rceil > 2B \\
& \vdots \\
& \sum_{i=1}^{\overline{P}} x_i \lceil (T_b - 1)/i \rceil > (T_b - 1)B \\
& \sum_{i=1}^{\overline{P}} x_i \lceil T_b/i \rceil \leq T_b B
\end{aligned}$$

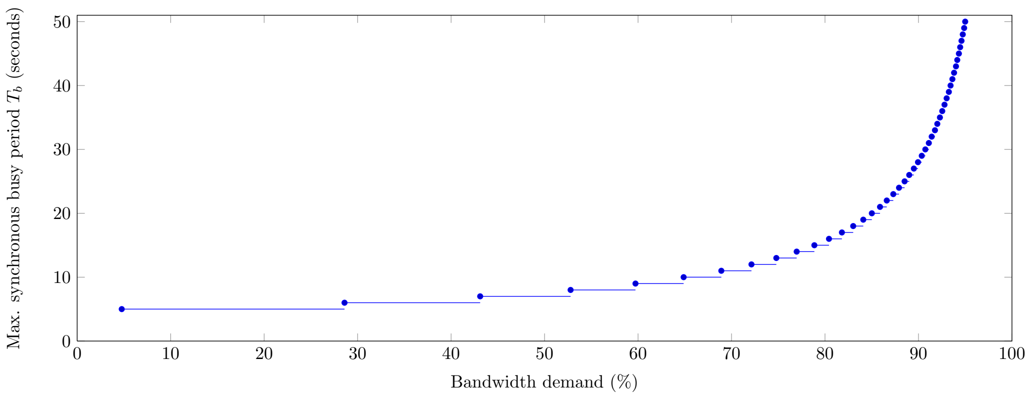


Figure 4.15: The maximum synchronous busy period T_b as a function of the bandwidth demand, for $N = 200$ streams, a maximum stream period of $\bar{P} = 255$ seconds, and $B = 51$ data slots available per round.

The objective function is the bandwidth demand u. The inequality constraints ensure that at the end of each interval $[0, t]$, $t \in \{1, 2, \ldots, T_b - 1\}$, there is at least one pending packet, while there is no pending packet at the end of interval $[0, T_b]$.[4] Solving this ILP for different values of T_b, we obtain a function $f(u)$ that gives the maximum T_b for a given bandwidth demand u, as shown in Figure 4.15. We now want to invert this function, that is, compute a stream set with a bandwidth demand as close as possible to a given target bandwidth demand u, and with the maximum possible T_b. To this end, we solve the following modified ILP:

$$\begin{aligned} \underset{\{x_1, x_2, \ldots, x_{\overline{P}}\}}{\text{minimize}} \quad & \sum_{i=1}^{\overline{P}} x_i/i \\ \text{subject to} \quad & \sum_{i=1}^{\overline{P}} x_i = N \\ & \sum_{i=1}^{\overline{P}} x_i/i \geq u \\ & \sum_{i=1}^{\overline{P}} x_i \lceil 1/i \rceil > B \end{aligned}$$

[4]Although the non-linear ceiling function appears in the ILP, it does not operate on the variables x_i and T_b is a known input. Thus, the left-hand sides of the inequalities are linear in the variables, and the program is efficiently solved by an ILP solver.

$$\sum_{i=1}^{\overline{P}} x_i \lceil 2/i \rceil > 2B$$

$$\vdots$$

$$\sum_{i=1}^{\overline{P}} x_i \lceil (f(u)-1)/i \rceil > (f(u)-1)B$$

$$\sum_{i=1}^{\overline{P}} x_i \lceil f(u)/i \rceil \leq f(u)B$$

We solve the two ILPs above to determine worst-case stream profiles considering the maximum number of streams $N = 200$ supported by our Blink prototype on the TelosB for a maximum stream period of $\overline{P} = 255$ seconds. We determine different sets of $N = 200$ streams for bandwidth demands between 5 % and 95 %, with $B = 51$ available slots per round. Table 4.2 lists the stream sets we compute and use for the experiments together with their synchronous busy period T_b.

Method: comparison implementation. To assess the effectiveness of our bucket queue implementation of the **LS** scheduler, we also implement the first three steps in Figure 4.12 following the conventional approach that is based on analytical computations like those in (4.3). This includes an implementation of the fastest analytic EDF schedulability test known today [ZB09] for admission control.

Method: execution. We benchmark both implementations in 2.5 hours executions with Blink. During those, requests for each of the 200 streams are submitted one by one in consecutive rounds, and we measure in each round the individual execution times of the four different steps in the **LS** scheduler. Then, we take *for each step individually* the maximum execution time we measured throughout the 2.5 hours. The results we report are *sums* of these *individual* maximum execution times, which are *higher* than the maximum *total* execution time we measured across all rounds.

Method: platforms. We use three diverse MCUs. A 16-bit MSP430F1611 running at 4 MHz, which is available on the TelosB and representative of the class of MCUs currently used to achieve the lowest possible energy consumption in the applications outlined in Section 4.1. We also consider a 32-bit ARM Cortex-M0 clocked at 48 MHz and a 32-bit ARM Cortex-M4 running at 72 MHz. The two ARM cores offer higher processing power but also incur higher energy consumption; nevertheless, yet they might represent a viable option in scenarios where some energy overhead can be traded for better computing capabilities [KKR+12].

Table 4.2: Stream profiles used in the experiment of Section 4.5.3 to approximate the worst-case execution time of the **LS** scheduler. *For all 200 streams $s_i \in \Lambda$, the start time S_i is set to 0 and the deadline D_i is equal to the period P_i, which is shown in the table.*

Bandwidth demand (%)	**Synchronous busy period (s)**	**Periods of the worst-case stream profiles (number of streams with a given period in seconds, written as #streams × period)**
5	5	1×2, 4×3, 195×255
10	5	5×3, 11×4, 184×255
15	5	4×1, 8×3, 1×4, 187×255
20	5	4×1, 15×3, 2×4, 179×255
25	5	3×1, 1×2, 25×3, 1×4, 170×255
30	6	3×1, 1×2, 6×3, 36×4, 1×5, 153×255
35	6	3×1, 39×3, 5×4, 153×255
40	6	7×1, 36×3, 4×5, 153×255
45	7	19×1, 1×2, 4×3, 5×4, 1×5, 170×255
50	7	15×1, 24×3, 6×4, 2×5, 153×255
55	8	11×1, 44×3, 1×4, 8×5, 136×255
60	9	27×1, 6×6, 14×7, 153×255
65	10	31×1, 3×2, 166×255
70	11	31×1, 1×6, 14×7, 18×9, 136×255
75	13	35×1, 1×5, 1×8, 1×9, 10×10, 14×11, 138×255
80	15	36×1, 1×3, 44×11, 119×255
85	19	39×1, 1×3, 1×5, 1×7, 1×12, 40×14, 5×17, 112×255
90	28	42×1, 5×2, 4×13, 4×24, 9×25, 136×255
95	50	46×1, 3×3, 2×8, 3×40, 5×41, 2×42, 5×43, 5×44, 2×45, 5×46, 5×47, 117×255

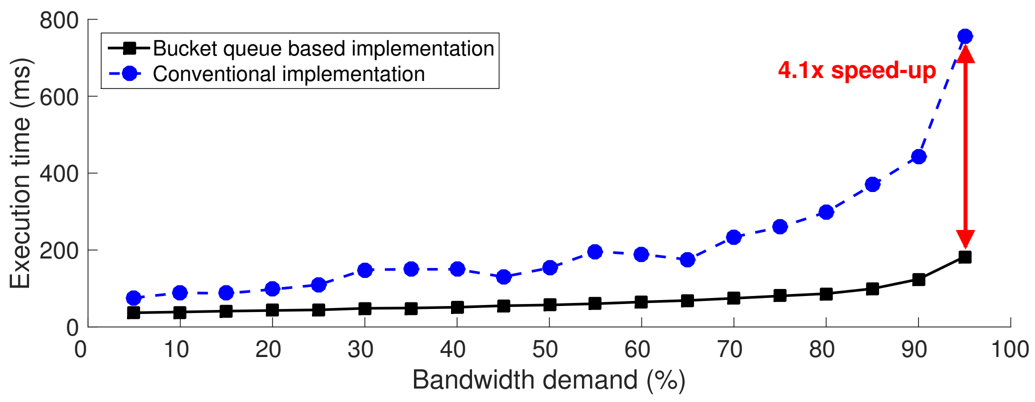


Figure 4.16: Execution time of **LS** scheduler against bandwidth demand on a 16-bit MSP430 running at 4 MHz, for our bucket queue-based implementation and a conventional one. *Our implementation using simple bucket queues consistently outperforms the conventional approach that uses only analytical computations, achieving speed-ups of up to 4.1×.*

We compile the exact same code by using msp430-gcc v4.6.3 for the MSP430 and IAR build tools for the two ARM cores; we always choose the highest possible optimization level that makes the binaries still fit into program memory. We deploy these binaries in the MSPsim time-accurate instruction-level emulator [EOF⁺09] for the MSP430, and on evaluation boards from STMicroelectronics for the two ARM cores. In all cases, we measure execution times in software with microsecond accuracy.

Results. Figure 4.16 plots the total execution time of the two **LS** scheduler implementations on the MSP430 as the bandwidth demand increases from 5 % to 95 %. We see that the total execution time increases slightly in the beginning, but ramps up severely for the conventional implementation as the bandwidth demand exceeds 65 %. This is due to an increase in the times needed for synchronous busy period computation, admission control, and start of round computation, whereas the time needed for slot allocation remains almost constant.

As a consequence, our bucket queue-based implementation consistently outperforms the conventional one, culminating in a 4.1× speed-up at 95 % bandwidth demand. At this high demand, the reduced scheduler execution time (182 ms versus 756 ms) means there is space for 44 instead of only 3 data slots per round. Simulating analytical computations using our efficient bucket queue implementation is thus mandatory for a viable implementation of the **LS** policy on this class of devices, which in turn ensures minimal network-wide energy consumption given the operation of the underlying LWB communication support.

While useful for approximating the worst-case execution time of the **LS** scheduler, the stream sets in Table 4.2 are not often seen in *low-power* wireless applications. Our review in Section 4.1 indicates that the typical demands would rarely exceed 20 % of the maximum bandwidth. In this regime, we measure execution times below 43 ms with the bucket queue implementation: well below the upper bound of 100 ms in our current Blink prototype. Thus, due to a 2.3× speed-up over the conventional implementation in this regime, there is still plenty of room for employing more constrained ultra-low-power platforms, or for considerably scaling up the number of streams with more available memory.

This also holds for the ARM cores. As one would expect, especially the conventional implementation benefits from the more powerful instruction sets, in particular on the Cortex-M4, which features a small set of SIMD instructions and also a hardware divide. This explains why we consistently measure scheduler execution times below 30 ms. Nevertheless, our bucket queue based implementation achieves speed-ups of 1.6–2× on both cores for realistic bandwidth demands of up to 20 %. This is mostly because using the bucket queues, the next time t that the loop in Algorithm 2 should examine is readily available because of the EDF-based ordering of the streams. By contrast, using the conventional approach, the next time t must be explicitly computed, which costs as much as computing the future demand $h_i(t)$ itself via (4.3). In conclusion, a bucket queue based implementation of the **LS** scheduler is beneficial even on less constrained state-of-the-art platforms, allowing to increase the bandwidth or to schedule more streams in the same amount of time.

4.6 Discussion and Limitations

Our current Blink prototype supports streams with relative deadlines between 1 and 255 seconds. Thus, it already satisfies the needs of many CPS applications characterized by time-critical monitoring and control functionality, especially in case of installations with many nodes across large areas [ÅGB11, Oga01, PL10, Whi08]. In specific closed-loop control settings, however, the networks are often smaller containing some tens of nodes, and tighter deadlines of 10–500 ms are commonplace [ÅGB11].

Our prototype can also support these scenarios by reducing the length of a round. This essentially means to reduce the number and size of slots in a round and the time allotted to the scheduler, as shown in Figure 4.1 (B). We detail in Appendix 4.B how the former two are influenced by factors such as packet size and network diameter. As a concrete example, in a 3-hop network, $B = 20$ slots per round, 40 ms for computing the schedule, and 10-byte packets—sufficient for sensor readings and control signals—we can tune our Blink prototype to support deadlines as short as 200 ms.

4.7 Summary

This chapter presented Blink, the first low-power wireless protocol that supports hard real-time traffic in large multi-hop networks at low energy costs. Blink overcomes the fundamental limitations of prior approaches with respect to the scalability against time-varying network state and the ability to smoothly adapt to dynamic changes in the application's real-time requirements. We demonstrated through real-world experiments that our Blink prototype meets nearly 100 % of packet deadlines regardless of changes in the set of streams or the network state, while maintaining the minimum energy consumption within the limits set by LWB, which we leverage as communication support in Blink. Our efficient priority queue data structure enables speed-ups of up to 4.1× over a conventional scheduler implementation based on analytical computations on popular low-power microcontrollers. We thus maintain that our work provides a major stepping stone towards a widespread deployment of reliable and time-critical low-power wireless applications in emerging CPS scenarios.

4.A Synchronous Busy Period Computation

The duration of the synchronous busy period T_b is crucial to admission control and computing the start time of the next round according to the **LS** policy. It denotes the time needed to contiguously serve the maximum demand that a given set of streams creates when all streams release a packet at the exact same time. The real-time literature suggests computing the synchronous busy period T_b through an iterative process [SRS98]

$$\omega^0 = \frac{n}{B} \quad \text{and} \quad \omega^{m+1} = \frac{1}{B}\sum_{i=1}^{n}\left\lceil \frac{\omega^m}{P_i} \right\rceil \tag{4.24}$$

which terminates when $\omega^{m+1} = \omega^m$; then, $T_b = \lceil \omega^m \rceil$.

As discussed in Section 4.4.2, implementing this iterative method on a resource-constrained platform leads to prohibitive running times due to many costly divisions. To overcome this problem, we compute T_b by *simulating* the execution of **CS**, which essentially entails going through the same processing that underlies (4.24) in a step-by-step manner. To this end, we trigger the maximum demand by letting all streams release

Algorithm 4 Compute Synchronous Busy Period

```
Input A bucket queue based copy of the current set of streams, where
    $s.releaseTime$ is initialized to 0 for all streams $s_i$ and streams with smaller
    $releaseTime$ are given higher priority.
Output The synchronous busy period $T_b$ of the set of streams.
  initialize $T_b$ and slot counter $c$ to 0
  set $s$ to highest-priority stream using $s$ = FindMin()
  while ($s.releaseTime = 0$) or ($s.releaseTime < T_b$ and $c = 0$) or ($s.releaseTime \leq T_b$
  and $c > 0$) do
      if current round has only one free slot ($c = B - 1$) then
          set $T_b = T_b + 1$ and $c = 0$ to "start" a new round
      else
          set $c = c + 1$ to "allocate" a slot in the current round
      end if
      $s.releaseTime = s.releaseTime + s.period$
      update priority of $s$ using DecreaseKey($s$, $s.period$)
      set $s$ to highest-priority stream using $s$ = FindMin()
  end while
  if current round has at least one allocated slot ($c > 0$) then
      set $T_b = T_b + 1$ to round up to the next discrete time
  end if
```

a packet at time $t = 0$. Using **CS**, we then serve this demand "as fast as possible" until we find the first idle time where no packet is pending.

Algorithm 4 shows the pseudocode. The algorithm operates on a deep copy of the current set of streams, maintained in a bucket queue in order of increasing release time.[5] It fictitiously allocates slots to packets in the order in which they are released. T_b keeps track of the number of *full* rounds that contain no free slot, and slot counter c keeps track of the number of allocated slots in the current round. The algorithm executes as long as there is a stream s whose initial packet is still to be sent (i.e., $s.releaseTime = 0$), or there is a packet that was already pending before the new round started (i.e., $s.releaseTime < T_b$ **and** $c = 0$), or there is any pending packet while the current round has at least one allocated slot (i.e., $s.releaseTime \leq T_b$ **and** $c > 0$). Otherwise, the algorithm has encountered the first idle time, which marks the end of the synchronous busy period.

[5]This results in high efficiency, because in each iteration the algorithm needs to look at the earliest release time of all streams to check whether it has encountered the first idle time where no packet is pending.

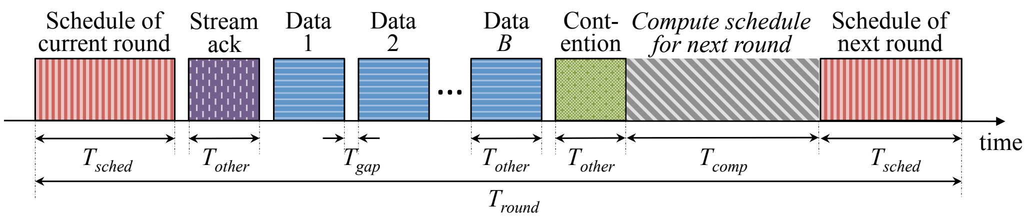


Figure 4.17: Communication slots and processing in a complete LWB round in our Blink prototype.

4.B Supporting Sub-second Deadlines

The fixed length of a round, T, determines the shortest possible period P_i and relative deadline $D_i \leq P_i$ of a stream s_i in Blink. To make our Blink prototype support shorter deadlines than $T = 1$ second, we must reduce the length of a round while carefully considering a number of influencing factors, as discussed in the following.

Length of a round. To reason about (a lower bound on) T, we must consider a *complete* round composed of all possible slots and processing activities. Figure 4.17 provides a zoomed-in view of Figure 4.1 (B), showing the slots and activities in a complete round in our prototype.

Every round starts with a slot in which the host distributes the schedule for the current round. This schedule allows each node to update its synchronization state and specifies the nodes that send in the following slots. Next, there is a slot in which the host sends a possible stream acknowledgment, informing a node whether its requested stream has been admitted or not. Nodes send their data packets in the following B data slots. In the contention slot, nodes compete to transmit their stream requests.[6] Based on received stream requests and all streams admitted thus far, the host computes the schedule for the next round. Finally, the host distributes the new schedule, so nodes know when the next round starts and can thus turn off their radios until then to save energy.[7]

As visible in Figure 4.17, there is a small gap between consecutive slots. This gap gives LWB just enough time to put a received packet into the incoming packet queue (at the intended receives) and to fetch the packet that is to be sent in the next slot from the outgoing packet queue (at the respective senders). To enable this operation, the application

[6]A node uses the contention slot only to submit its *first* stream request. Once a node has an admitted stream, it submits further request to add, remove, or update streams by piggybacking on data packets [FZMT12].

[7]Like in the original LWB [FZMT12], Blink's real-time scheduler allocates slots for a stream acknowledgment and data packets as needed, and schedules the contention slot less often if no stream request recently arrived, without changing the length of a round.

must ensure that all released packets are in LWB's outgoing queue before a round starts. Conversely, LWB ensures that when a round ends, all received packets are in the incoming queue. In fact, as mentioned earlier, it is not until this time that the application gains control of the MCU to process received packets.

Using the notation in Figure 4.17, we add up the durations of the different slots, gaps, and schedule computation to obtain an expression for the (minimum) length of round

$$T = 2T_{sched} + (B + 2)(T_{other} + T_{gap}) + T_{comp}. \tag{4.25}$$

From (4.25) follows that we can make a round shorter by reducing (*i*) the number of data slots in a round B, (*ii*) the time for computing the schedule T_{comp}, or (*iii*) the size of schedule T_{sched} and other slots T_{other}. (*i*) is an application-specific trade-off between a higher energy overhead per round (relative to the B useful data slots) and the possibility to support shorter periods and deadlines. (*ii*) depends on the scheduling policy and the processing demand induced by the application-dependent streams. Finally, (*iii*) is a function of several application, deployment, and platform characteristics, as discussed next.

Length of a slot. Each slot within a round consists of a Glossy flood. To obtain (a lower bound on) the length of a slot, we need to briefly recap the operation of Glossy.

As shown in Figure 4.18, the *initiator* (i.e., the node that is to send in a slot according to the schedule) starts the flood by transmitting its packet, while all other nodes, the *receivers*, have their radios turned on. Nodes within the initiator's radio range, the *1-hop receivers*, receive the packet at the same time and, by ensuring a constant processing time across all nodes, they also relay the packet at the same time. As this operation continues, nodes that are two, three, or more hops away from the initiator also receive and relay the packet. To achieve reliabilities above 99.9 %, each node transmits the packet up to N times during a Glossy flood [FZTS11]. For example, in Figure 4.18, each node transmits $N = 2$ times.

The length of a slot, T_{slot}, should be sufficient to allow also the nodes farthest away from the initiator to *receive* N times. Let H be the network diameter (i.e., the maximum hop distance between any two nodes), and let T_{hop} be the time between consecutive transmissions during a flood (see Figure 4.18). T_{hop} is constant during a flood, since nodes do not alter the packet size. Thus, the length of a slot such that each node in a H-hop network gets the chance to receive the packet N times during a flood is

$$T_{slot} = (H + 2N - 2)T_{hop}. \tag{4.26}$$

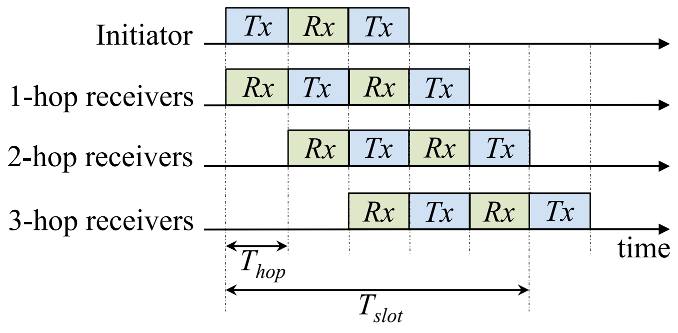


Figure 4.18: Illustration of a Glossy flood in a 3-hop network, where each node transmits $N = 2$ times. *The length of a slot T_{slot} should be sufficient to give all nodes in the network the chance to receive the packet N times.*

We obtain an expression for T_{hop} by adding up the time required for the actual packet transmission (or reception) T_{tx}, the processing delay of the radio at the beginning of a packet reception T_d, and the software delay introduced by Glossy when triggering a packet transmission T_{sw}

$$T_{hop} = T_{tx} + T_d + T_{sw}. \tag{4.27}$$

T_d is a radio-dependent constant and T_{sw} is a constant specific to the Glossy implementation for a given platform; Table 4.3 lists their values for the TelosB and the CC2420 radio. The time needed by the CC2420 to transmit an IEEE 802.15.4-compliant packet is the sum of the time needed to calibrate the radio's internal voltage controlled oscillator T_{cal}, the time for transmitting the 5-byte synchronization header and the 1-byte PHY header T_{header}, and the time for transmitting the MAC protocol data unit $T_{payload}$, which contains the actual payload

$$T_{tx} = T_{cal} + T_{header} + T_{payload}. \tag{4.28}$$

The values of T_{cal} and T_{header} are listed in Table 4.3. The time required to transmit a payload of size $L_{payload}$ (between 1 and 127 bytes in IEEE 802.15.4)

Table 4.3: Constants specific to the CC2420 radio and the Glossy implementation for the TelosB platform we use.

Name	**Value**
T_d	3 μs
T_{sw}	23.5 μs
T_{cal}	192 μs
T_{header}	192 μs
R_{bit}	250 kbps

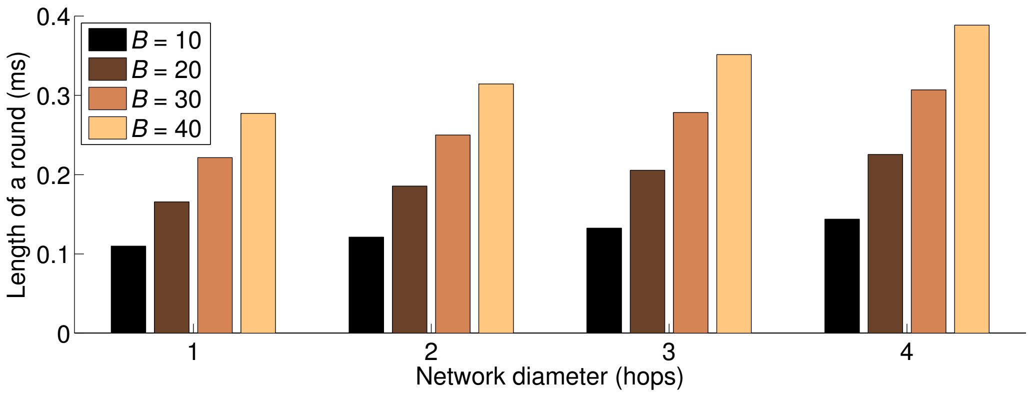


Figure 4.19: Length of a round in Blink depending on network diameter and number of data slots in a round B.

using a radio that has a transmit bit rate of R_{bit} (see Table 4.3) is

$$T_{payload} = 8L_{payload}/R_{bit}. \tag{4.29}$$

Discussion. We use the above expressions to obtain an estimate of the length of a round T in our current Blink prototype, targeting the TelosB platform and the CC2420 radio. T determines the shortest possible period P_i and deadline D_i of a stream in Blink. Since short periods are particularly important in specific closed-loop control scenarios [ÅGB11], we consider typical characteristics of those.

The networks are relatively small, containing up to 50 nodes [ÅGB11]. Assuming the number of streams in the system is on the same order, our results from Section 4.5.3 with 200 streams suggest that the **LS** scheduler completes for sure within T_{comp} = 40 ms for a wide range of realistic bandwidth demands. Discussions with control experts indicate that a payload of $L^{other}_{payload}$ = 10 bytes is often enough for actuation signals, sensor readings, stream requests, and stream acknowledgments. A schedule packet consists of a 7-byte header and a sequence of node/stream IDs, so the payload of schedule packets is $L^{sched}_{payload} = 7+2(B+2)$ bytes. We set T_{gap} = 3 ms and want every node in the network to receive the packet at least N = 2 times—at this setting, Glossy provides a packet reliability above 99.9 % in real-world experiments with more than 100 nodes [FZTS11].

Given these settings, Figure 4.19 plots the length of a round T depending on network diameter H and number of data slots in a round B. For example, in a 3-hop network and B = 20 slots per round, we can tune our Blink prototype to support periods and deadlines as short as 200 ms. Thus, under given assumptions, Blink satisfies the needs of specific closed-loop control scenarios in terms of high refresh rates and hard end-to-end packet deadlines [ÅGB11].

5 Conclusions and Outlook

Collections of small, embedded devices with low-power wireless network interfaces enable applications that are expected to have a profound impact on the world. Data collection applications employ sensing devices that are deeply embedded into the environment for monitoring physical processes at unprecedented spatio-temporal resolutions, from habitats to manmade structures. Cyber-physical systems (CPS) applications, on the other hand, use sensing and actuating devices to control physical processes, usually via feedback loops in scenarios such as building and factory automation.

To successfully deploy these applications, designers must ensure that the network's global, end-to-end performance matches given application-specific performance goals. In particular, requirements in terms of reliable and timely yet energy-efficient packet delivery have to be met in the face of severe resource constraints of the employed devices and unpredictable and non-deterministic changes in the environment. Unfortuntely, existing low-power wireless communication protocols and systems typically focus on meeting a single application-defined performance goal (e.g., minimum energy) or consider only local metrics (e.g., per-hop latency), and provide no hard guarantees on end-to-end packet deadlines which are necessary to use multi-hop low-power wireless networks in critical CPS applications.

5.1 Contributions

To fill these gaps, we have made three main contributions in this thesis.

pTunes. We designed pTunes, a framework that adapts the operational parameters of a given low-power MAC protocol at runtime in response to dynamic changes in the network state and the traffic load. Targeting data collection applications employing tree-based routing, pTunes thus meets multiple soft application requirements specified in terms of global, end-to-end performance metrics. pTunes achieves this with a novel flooding-based solution for efficiently and reliably collecting consistent network state at a central controller and distributing new MAC parameters back to the nodes independent of the network state, and determining optimized MAC parameters using a network-wide performance model.

Simple yet accurate modeling of ST-based protocols. Given the need for accurate performance models and the inherent difficulties to obtain them for protocols using link-based transmissions (LT), we examined whether this situation improves for the rapidly growing class of protocols utilizing synchronous transmissions (ST). Indeed, we empirically showed that the Bernoulli assumption, which can simplify protocol modeling to a great extent, is largely valid for ST in Glossy. We could thus devise a Markovian model to estimate LWB's energy costs with an unparalleled accuracy, and sufficient conditions to provide probabilistic guarantees on LWB's end-to-end reliability. As a recent example that demonstrates the simplicity and practicality of our models, Filieri et al. have successfully used them within a runtime efficient probabilistic model checking framework that executes right on resource-constrained low-power wireless devices [FTG15].

Blink. Finally, we extended LWB's best-effort operation to built Blink, which is, to the best of our knowledge, the first protocol that provides hard guarantees on end-to-end packet deadlines in multi-hop low-power wireless networks. LWB's globally time-triggered operation allowed us to abstract the entire network as a single resource that runs on a single clock. We could thus map the scheduling problem in Blink to the well-known problem of scheduling tasks on a uniprocessor. We devised scheduling policies that Blink leverages to determine online a schedule that provably meets all deadlines of admitted packet streams at minimum energy cost, while tolerating changes in both the network state and the set of streams. Supported by efficient data structures and algorithms we designed, Blink thus provides timing-predictable wireless communication across multiple hops, which is crucial for the correctness of critical CPS applications.

5.2 Possible Future Directions

End-to-end communication performance is what really matters for many real-world low-power wireless applications, from data collection to CPS scenarios. We maintain that this thesis contributes key stepping stones to support these applications. A key lesson from our work on pTunes is that a centralized approach is both necessary to satisfy end-to-end application requirements and can indeed be implemented in a highly efficient manner. The surprisingly low energy overhead of pTunes's runtime support based on Glossy floods compared to the energy footprint of state-of-the-art MAC protocols was enlightening, and partially triggered and motivated us to really pursue the idea of a wireless bus, which is now embodied by LWB. Conceptually, the feedback control approach we adopted in pTunes can also be found in LWB: The host computes and distributes schedules based on the traffic demands and application requirements, such as minimizing energy consumption as in the original LWB scheduler or meeting packet deadlines at minimum energy costs as with Blink's real-time scheduler. However, things become way simpler in LWB, because there is no time-varying network state to collect or consider in the scheduling decisions, despite nodes with streams joining or leaving the network. Additionally, the modeling becomes easier and also more accurate, owing to the validity of the Bernoulli assumption in combination with the time-triggered and highly deterministic behavior of a LWB node being largely independent of the volatile network state. Like in pTunes, our models could be used at runtime to, for example, detect violations of application requirements or guide the scheduling decisions in LWB. In addition, we discuss here three directions that we believe deserve further investigation.

Deeper understanding of ST. We exploited and empirically showed in this thesis how a physical-layer innovation, namely ST in Glossy, impacts protocol design and modeling. To further enhance the performance and reliability of ST and take full advantage of their salient characteristics in the design and implementation of higher-level mechanisms, we believe it is worthwile to conduct further systematic studies of ST, both through controlled experiments, for example, using a wireless channel emulator or in an anechoic chamber, and using analytical models and simulations. This could, for example, provide insights into the development of "ST-friendly" physical layers (modulations scheme, transceiver design, etc.) or the applicability of (analog) network coding schemes to ST.

Distributed scheduling. Although LWB includes mechanisms to resume its operation after a host failure, a distributed scheduling approach where every node computes the communication schedule locally could make LWB even more resilient, more reactive, and more bandwidth efficient. A key challenge to realizing this idea is to ensure a consistent input to the distributed scheduling logic across all nodes. This entails the delivery of stream requests to and the detection of failed nodes by all nodes at the same time, among other things. Using Chaos [LFZ13a] for these all-to-all interactions and to keep nodes time-synchronized could be a promising starting point. Also, based on our experimental results with Chaos, we believe integrating it side-by-side with Glossy into LWB could improve the efficiency of some distributed interactions within LWB and provide support for even shorter deadlines than possible with our Blink prototype.

Unify delivery and real-time guarantees. Many CPS rely on real-time communication for stable closed-loop control, as enabled by Blink. Being resilient against failures of the controller, in turn, requires replicating the controller across different physical devices and reliable ordered delivery of sensor readings to all replicas, as enabled by Virtus [FZMT13]. It appears beneficial but challenging to unify these separate solutions. For example, Blink currently uses no packet retransmissions at all, whereas Virtus relies on possibly infinite retransmissions per packet to provably provide delivery guarantees. Nevertheless, we believe it is possible to integrate the two within an extended LWB, for instance, by providing real-time guarantees for some streams and delivery guarantees for others, or by resorting to probabilistic versions of these guarantees based on information about the minimum transmission reliability in the network.

Bibliography

[ABD$^+$11] Y. Agarwal, B. Balaji, S. Dutta, R. K. Gupta, and T. Weng. Duty-cycling buildings aggressively: The next frontier in HVAC control. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[ÅGB11] J. Åkerberg, M. Gidlund, and M. Björkman. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *Proceedings of the 9th IEEE International Conference on Industrial Informatics (INDIN)*, 2011.

[ASSC02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8), 2002.

[AW07] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, 2007.

[AY05] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Elsevier Ad Hoc Networks*, 3(3), 2005.

[AYC$^+$07] A. M. Ali, K. Yao, T. C. Collier, C. E. Taylor, D. T. Blumstein, and L. Girod. An empirical study of collaborative acoustic source localization. In *Proceedings of the 6th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[BD91] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer, 1991.

[BGH$^+$09] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. PermaDAQ: A scientific instrument for precision sensing and data recovery under extreme conditions. In *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.

[BJ87] K. P. Birman and T. A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles (SOSP)*, 1987.

[BKM$^+$12] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Transactions on Sensor Networks*, 8(4), 2012.

[Bro88] R. Brown. Calendar queues: A fast $o(1)$ priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10), 1988.

[Bro13] G. S. Brodal. A survey on priority queues. In A. Brodnik, A. López-Ortiz, V. Raman, and A. Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms*. Springer, 2013.

[BSB$^+$12] D. Bruneo, M. Scarpa, A. Bobbio, D. Cerotti, and M. Gribaudo. Markovian agent modeling swarm intelligence algorithms in wireless sensor networks. *Performance Evaluation*, 69(3–4), 2012.

[BSBT14] B. Buchli, F. Sutton, J. Beutel, and L. Thiele. Dynamic power management for long-term energy neutral operation of solar energy harvesting systems. In *Proceedings of the 12th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2014.

[But05] G. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1), 2005.

[But11] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011.

[BVJ$^+$10] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester. The w-iLab.t testbed. In *Proceedings of the 6th ICST International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, 2010.

[BvRW07] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *Proceedings of the 6th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[BVT$^+$10] C. A. Boano, T. Voigt, N. Tsiftes, L. Mottola, K. Römer, and M. Zuniga. Making sensornet MAC protocols robust against interference. In *Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN)*, 2010.

[BYAH06] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[CAN] ISO 11898—Controller Area Network (CAN). `http://www.iso.org/iso/catalogue_detail.htm?csnumber=59165`.

[CCD+11] M. Ceriotti, M. Corrà, L. D'Orazio, R. Doriguzzi, D. Facchin, Ş. Gună, G. P. Jesi, R. Lo Cigno, L. Mottola, A. L. Murphy, M. Pescalli, G. P. Picco, D. Pregnolato, and C. Torghele. Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[CCT+13] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali. Forwarder selection in multi-transmitter networks. In *Proceedings of the 9th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2013.

[CFP+06] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10(2), 2006.

[CKJL09] J. I. Choi, M. A. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[CLBR10] O. Chipara, C. Lu, T. C. Bailey, and G.-C. Roman. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[CMP+09] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, Ş. Gună, M. Corrà, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.

[CT11] D. Carlson and A. Terzis. Flip-MAC: A density-adaptive contention-reduction protocol for efficient any-to-one communication. In *Proceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2011.

[CWLG11] O. Chipara, C. Wu, C. Lu, and W. Griswold. Interference-aware real-time flow scheduling for wireless sensor networks. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS)*, 2011.

[CWPE05] A. Cerpa, J. Wong, M. Potkonjak, and D. Estrin. Temporal properties of low power wireless links: Modeling and implications on multi-hop routing. In *Proceedings of the 4th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.

[CWW10] G. W. Challen, J. Waterman, and M. Welsh. IDEA: Integrated distributed energy awareness for wireless sensor networks. In *Proceedings of the 8th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[DBK+07] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. Deployment support network: A toolkit for the development of WSNs. In *Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN)*, 2007.

[DCA11] M. Doddavenkatappa, M. Chan, and A. Ananda. Indriya: A low-cost, 3D wireless sensor network testbed. In *Proceedings of the 7th ICST International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, 2011.

[DCABM03] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2003.

[DCL13] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[DDHC+10] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[DEM+10] V. Dyo, S. A. Ellwood, D. W. Macdonald, A. Markham, C. Mascolo, B. Pásztor, S. Scellato, N. Trigoni, R. Wohlers, and K. Yousef. Evolution and sustainability of a wildlife monitoring sensor network. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[Der74] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings of IFIP Congress*, 1974.

[Det13] M. Dettling. Applied time series analysis, 2013. Online at `http://stat.ethz.ch/education/semesters/ss2013/atsa/ATSA-Scriptum-SS2013_130324.pdf`.

[DGV04] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - A lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 1st IEEE Workshop on Embedded Networked Sensors (EmNetS-I)*, 2004.

[Dia69] R. B. Dial. Algorithm 360: Shortest path forest with topological ordering. *Communications of the ACM*, 12(11), 1969.

[DL03] T. Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[DOTH07] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th IEEE Workshop on Embedded Networked Sensors (EmNets)*, 2007.

[DPR00] S. R. Das, C. E. Perkins, and E. M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2000.

[EHD04] A. El-Hoiydi and J.-D. Decotignie. WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks. In *Proceedings of the 1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, 2004.

[EOF$^+$09] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón. COOJA/MSPSim: Interoperability testing for wireless sensor networks. In *Proceedings of the 2nd ICST International Conference on Simulation Tools and Techniques (SIMUTools)*, 2009.

[FaSdAL04] A. X. Falcão, J. Stolfi, and R. de Alencar Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1), 2004.

[Fle] ISO 17458—FlexRay Communications System. `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=59804`.

[FTG15] A. Filieri, G. Tamburrelli, and C. Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 2015. To appear.

[FZMT12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2012.

[FZMT13] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Virtual synchrony guarantees for cyber-physical systems. In *Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, 2013.

[FZTS11] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[GB12] M. C. Guenther and J. T. Bradley. PCTMC models of wireless sensor network protocols. In *Proceedings of the 28th UK Performance Engineering Workshop (UKPEW)*, 2012.

[GCB08] M. Gribaudo, D. Cerotti, and A. Bobbio. Analysis of on-off policies in sensor networks using interacting Markovian agents. In *Proceedings of the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2008.

[Gel07] E. Gelenbe. A diffusion model for packet travel time in a random multihop medium. *ACM Transactions on Sensor. Networks*, 3(2), 2007.

[GFJ$^+$09] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[GGB$^+$10] E. Gaura, L. Girod, J. Brusey, M. Allen, and G. Challen, editors. *Wireless Sensor Networks: Deployments and Design Frameworks*. Springer, 2010.

[GGL10] O. Gnawali, L. Guibas, and P. Levis. A case for evaluating sensor network protocols concurrently. In *Proceedings of the 5th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2010.

[GHLX09] Y. Gu, T. He, M. Lin, and J. Xu. Spatiotemporal delay control for low-duty-cycle sensor networks. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, 2009.

[GPM89] C. E. Garciá, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3), 1989.

[har] HART communication foundation. `http://en.hartcomm.org/main_article/wirelesshart.html`.

[HB10] P. Hurni and T. Braun. MaxMAC: A maximally traffic-adaptive MAC protocol for wireless sensor networks. In *Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN)*, 2010.

[HHSH06] C. Hartung, R. Han, C. Seielstad, and S. Holbrook. FireWxNet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2006.

[HLR03] Q. Huang, C. Lu, and G.-C. Roman. Mobicast: Just-in-time multicast for sensor networks under spatiotemporal constraints. In *Proceedings of the 2nd IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2003.

[HLW71] Y. Y. Haimes, L. S. Lasdon, and D. A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(3), 1971.

[hon] Honeywell Inc: Honeywell Process Solutions—White Paper. `https://www.honeywellprocess.com/library/support/Public/Documents/WirelessWhitePaper_Nov2006.pdf`.

[HRV+13] F. Hermans, O. Rensfelt, T. Voigt, E. Ngai, L.-Å. Nordén, and P. Gunningberg. SoNIC: Classifying interference in 802.15.4 sensor networks. In *Proceedings of the 12th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2013.

[HSE] Health and safety legislation—laws in the workplace. `http://www.hse.gov.uk/legislation/`.

[HSLA05] T. He, J. A. Stankovic, C. Lu, and T. F. Abdelzaher. A spatiotemporal communication protocol for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 16(10), 2005.

[isa] ISA-100 Wireless Compliance Institute. `http://www.isa100wci.org/`.

[JBL07] R. Jurdak, P. Baldi, and C. V. Lopes. Adaptive low power listening for wireless sensor networks. *IEEE Transactions on Mobile Computing*, 6(8), 2007.

[KG93] H. Kopetz and G. Grünsteidl. TTP - A time-triggered protocol for fault-tolerant real-time systems. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, 1993.

[KGN09] R. Kuntz, A. Gallais, and T. Noel. Medium access control facing the reality of WSN deployments. *ACM SIGCOMM Computer Communication Review*, 39(3), 2009.

[KKR+12] J. Ko, K. Klues, C. Richter, W. Hofer, B. Kusy, M. Bruening, T. Schmid, Q. Wang, P. Dutta, and A. Terzis. Low power or high performance? a tradeoff whose time has come (and nearly gone). In *Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN)*, 2012.

[KLS+01] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. Distributed multi-hop scheduling and medium access with delay and throughput constraints. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2001.

[KRH+06] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: Practical wireless network coding. In *Proceedings of the ACM SIGCOMM Conference*, 2006.

[Lan08] K. Langendoen. Medium access control in wireless sensor networks. In *Medium Access Control in Wireless Networks*. Nova Science Publishers, 2008.

[LBA+02] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He. RAP: A real-time communication architecture for large-scale wireless sensor networks. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002.

[LBL+13] Y. Lee, S. Bang, I. Lee, Y. Kim, G. Kim, M. H. Ghaed, P. Pannuto, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1 mm^3 die-stacked sensing platform with low power I^{2}C inter-die communication and multi-modal energy harvesting. *IEEE Journal of Solid-State Circuits*, 48(1), 2013.

[Lee08] E. A. Lee. Cyber physical systems: Design challenges. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[LF76] K. Leentvaar and J. H. Flint. The capture effect in FM receivers. *IEEE Transactions on Communications*, 24(5), 1976.

[LFZ13a] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2013.

[LFZ+13b] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proceedings of the 12th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2013.

[lin] The Linux Completely Fair Scheduler (CFS). `https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt`.

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[LLL+09] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. RACNet: A high-fidelity data center sensing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[LM10] K. Langendoen and A. Meier. Analyzing MAC protocols for low data-rate applications. *ACM Transactions on Sensor Networks*, 7(2), 2010.

[LPCS04] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[LPLT10] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis. Surviving Wi-Fi interference in low power ZigBee networks. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[LSW09] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal clock synchronization in networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[LW09] J. Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, 2009.

[MCP+02] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[MELT08] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2008.

[METS+06] R. Musaloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life under your feet: A wireless soil ecology sensor network. In *Proceedings of the 3rd IEEE Workshop on Embedded Networked Sensors (EmNets)*, 2006.

[ML06] R. Madan and S. Lall. Distributed algorithms for maximum lifetime routing in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 5(8), 2006.

[MMWG14] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi. Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2014.

[MT06] R. Makowitz and C. Temple. Flexray - A communication network for automotive control systems. In *Proceedings of the 7th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2006.

[MTBB10] C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive power management for environmentally powered systems. *IEEE Transactions on Computers*, 59(4), 2010.

[MWZT10] A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele. ZeroCal: Automatic MAC protocol calibration. In *Proceedings of the 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2010.

[NAE] NAE grand challenges for engineering. `http://engineeringchallenges.org/cms/challenges.aspx`.

[OBB+13] T. O'Donovan, J. Brown, F. Büsching, A. Cardoso, J. Cecilio, J. Do Ó, P. Furtado, P. Gil, A. Jugel, W.-B. Pöttner, U. Roedig, J. Sá Silva, R. Silva, C. J. Sreenan, V. Vassiliou, T. Voigt, L. Wolf, and Z. Zinonos. The GINSENG system for wireless monitoring and control: Design and deployment experiences. *ACM Transactions on Sensor Networks*, 10(1), 2013.

[Oga01] K. Ogata. *Modern Control Engineering*. Prentice Hall PTR, 4th edition, 2001.

[Per] Personal communication with Tomas Lennval of ABB Corporate Research.

[Pfa] B. Pfaff. An introduction to binary search trees and balanced trees. Online at `http://adtinfo.org/libavl.html/`.

[PFJ10] P. Park, C. Fischione, and K. Johansson. Adaptive IEEE 802.15.4 protocol for energy efficient, reliable and timely communications. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2010.

[PG07] J. Paek and R. Govindan. RCRT: Rate-controlled reliable transport for wireless sensor networks. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

[PH10] D. Puccinelli and M. Haenggi. Reliable data delivery in large-scale low-power sensor networks. *ACM Transactions on Sensor Networks*, 6(4), 2010.

[PHC04] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[PL10] M. Paavola and K. Leiviska. *Wireless Sensor Networks in Industrial Automation*. Springer, 2010.

[PSC05] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005.

[RC08] B. Raman and K. Chebrolu. Censor networks: A critique of "sensor networks" from a systems perspective. *ACM SIGCOMM Computer Communication Review*, 38(3), 2008.

[RC10] J. M. Reason and R. Crepaldi. Ambient intelligence for freight railroads. *IBM Journal of Research and Development*, 53(3), 2010.

[RCBG10] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale. PIP: A connection-oriented, multi-hop, multi-channel TDMA-based MAC for high throughput bulk transfer. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[RCCD09] J. M. Reason, H. Chen, R. Crepaldi, and S. Duri. Intelligent telemetry for freight trains. In *Proceedings of the 1st ICST International Conference on Mobile Computing, Applications, and Services (MobiCASE)*, 2009.

[RGGP06] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. In *Proceedings of the ACM SIGCOMM Conference*, 2006.

[RHK10] H. Rahul, H. Hassanieh, and D. Katabi. SourceSync: A distributed wireless architecture for exploiting sender diversity. In *Proceedings of the ACM SIGCOMM Conference*, 2010.

[RLSS10] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC)*, 2010.

[SAÅ$^+$04] L. Sha, T. Abdelzaher, K.-E. Årzén, T. Cervin, Anton adn Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2–3), 2004.

[SDTL10] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks*, 6(2), 2010.

[SJC+10] K. Srinivasan, M. Jain, J. I. Choi, T. Azim, E. S. Kim, P. Levis, and B. Krishnamachari. The κ factor: Inferring protocol performance using inter-link reception correlation. In *Proceedings of the 16th ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2010.

[SKAL08] K. Srinivasan, A. Kazandjieva, S. Agarwal, and P. Levis. The β-factor: measuring wireless link burstiness. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[SKH06] D. Son, B. Krishnamachari, and J. Heidemann. Experimental study of concurrent transmission in wireless sensor networks. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[SLMR05] J. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *IEEE Computer*, 38(11), 2005.

[SMP+04] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[SMR+12] R. Sen, A. Maurya, B. Raman, R. Mehta, R. Kalyanaraman, N. Vankadhara, S. Roy, and P. Sharma. Kyun queue: A sensor network system to monitor road traffic queues. In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2012.

[SNSW10] S. Shahriar Nirjon, J. Stankovic, and K. Whitehouse. IAA: Interference-aware anticipatory algorithm for scheduling and routing periodic real-time streams in wireless sensor networks. In *Proceedings of the 7th IEEE International Conference on Networked Sensing Systems (INSS)*, 2010.

[Spu96] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, INRIA, 1996.

[SRS98] J. A. Stankovic, K. Ramamritham, and M. Spuri. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[SSF+04] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. I. Jordan, and S. S. Sastry. Kalman filtering with intermittent observations. *IEEE Transactions on Automatic Control*, 49(9), 2004.

[Sta08] J. A. Stankovic. When sensor and actuator networks cover the world. *ETRI Journal*, 30(5), 2008.

[Sve] S. Svensson. ABB Corporate Research: Challenges of wireless communication in industrial systems. Keynote talk at ETFA 2011, online at `http://www.iestcfa.org/presentations/etfa2011/keynote_Svensson.pdf`.

[SXLC10] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Real-time scheduling for WirelessHART networks. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS)*, 2010.

[TPS+05] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

[tsc] IEEE 802.15.4e Wireless Standard - Amandment1: MAC sublayer. `http://standards.ieee.org/findstds/standard/802.15.4e-2012.html`.

[Vil12] J. Villasenor. Observations from above: Unmanned aircraft systems. *Harvard Journal of Law and Public Policy*, 2012.

[WALJ+06] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[WC01] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the 7th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2001.

[Whi08] T. Whittaker. What do we expect from wireless in the factory? In *ETSI Wireless Factory*, 2008.

[WHM+12] Y. Wang, Y. He, X. Mao, Y. Liu, Z. Huang, and X. Y. Li. Exploiting constructive interference for scalable flooding in wireless networks. In *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)*, 2012.

[WHR+13] H. Wennerström, F. Hermans, O. Rensfelt, C. Rohner, and L.-Å. Nordèn. A long-term study of correlations between meteorological conditions and 802.15.4 link performance. In *Proceedings of the 9th IEEE International Conference on Sensing, Communication and Networking (SECON)*, 2013.

[WLS14] M. Wilhelm, V. Lenders, and J. B. Schmitt. On the reception of concurrent transmissions in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 13(12), 2014.

[XTLS08] F. Xia, Y.-C. Tian, Y. Li, and Y. Sun. Wireless sensor/actuator network design for mobile control applications. *IEEE Sensors*, 7(10), 2008.

[YHE02] W. Ye, J. Heidemann, and D. 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)*, 2002.

[YMKT99] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1999.

[YZDPHg11] W. Yi-Zhi, Q. Dong-Ping, and H. Han-guang. Pareto optimal collection tree protocol for industrial monitoring WSNs. In *Proceedings of IEEE GLOBECOM Workshops*, 2011.

[ZB09] F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 58(9), 2009.

[ZDR08] M. Zimmerling, W. Dargie, and J. M. Reason. Localized power-aware routing in linear wireless sensor networks. In *Proceedings of the 2nd ACM International Workshop on Context-Awareness for Self-Managing Systems (CASEMANS)*, 2008.

[ZFM$^+$12] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. pTunes: Runtime parameter adaptation for low-power MAC protocols. In *Proceedings of the 11th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2012.

[ZG03] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[ZSJ09] H. Zhang, P. Soldati, and M. Johansson. Optimal link scheduling and channel assignment for convergecast in linear WirelessHART networks. In *Proceedings of the 7th IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, 2009.

List of Publications

The following list includes publications that form the basis of this thesis. The corresponding chapters are indicated in parentheses.

M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. **pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols.** In *Proceedings of the 11th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN/IP Track).* Beijing, China, April 2012. ***Best paper runner-up.*** (Chapter 2)

M. Zimmerling, F. Ferrari, L. Mottola, and L. Thiele. **On Modeling Low-power Wireless Protocols Based on Synchronous Packet Transmissions.** In *Proceedings of the 21st IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS).* San Francisco, CA, USA, August 2013. (Chapter 3)

M. Zimmerling, F. Ferrari, L. Mottola, and L. Thiele. **Poster Abstract: Synchronous Packet Transmissions Enable Simple Yet Accurate Protocol Modeling.** In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Rome, Italy, November 2013. (Chapter 3)

M. Zimmerling, P. Kumar, L. Mottola, F. Ferrari, and L. Thiele. **Energy-efficient Real-time Communication in Multi-hop Low-power Wireless Networks.** *Under Submission at the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys)* Seoul, South Korea, November 2015. (Chapter 4)

The following list includes publications that are not part of this thesis.

M. Zimmerling, F. Ferrari, M. Woehrle, and L. Thiele. **Poster Abstract: Exploiting Protocol Models for Generating Feasible Communication Stack Configurations.** In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).* Stockholm, Sweden, April 2010.

D. Hasenfratz, A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele. **Poster Abstract: If You Have Time, Save Energy with Pull.** In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Zurich, Switzerland, November 2010.

A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele. **ZeroCal: Automatic MAC Protocol Calibration.** In *Proceedings of the 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS).* Santa Barbara, CA, USA, June 2010.

J. Beutel, B. Buchli, F. Ferrari, M. Keller, L. Thiele, and M. Zimmerling. **X-Sense: Sensing in Extreme Environments.** In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE).* Grenoble, France, March 2011. Invited paper.

F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. **Efficient Network Flooding and Time Synchronization with Glossy.** In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN/IP Track).* Chicago, IL, USA, April 2011. ***Best paper award.***

F. Ferrari, M. Zimmerling, L. Thiele, and L. Mottola. **The Bus Goes Wireless: Routing-free Data Collection with QoS Guarantees in Sensor Networks.** In *Proceedings of the 4th International Workshop on Information Quality and Quality of Service for Pervasive Computing (IQ2S), in conjunction with IEEE PerCom.* Lugano, Switzerland, March 2012.

F. Ferrari, M. Zimmerling, L. Thiele, and L. Mottola. **Poster Abstract: The Low-power Wireless Bus: Simplicity is (Again) the Soul of Efficiency.** In *Proceedings of the 11th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).* Beijing, China, April 2012.

F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. **Low-power Wireless Bus.** In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Toronto, Canada, November 2012.

O. Landsiedel, F. Ferrari, and M. Zimmerling. **Poster Abstract: Capture Effect-based Communication Primitives.** In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Toronto, Canada, November 2012. ***Best poster award.***

R. Lim, C. Walser, F. Ferrari, M. Zimmerling, and J. Beutel. **Demo Abstract: Distributed and Synchronized Measurements with FlockLab.** In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Toronto, Canada, November 2012.

R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. **FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems.** In *Proceedings of the 12th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN/SPOTS Track).* Philadelphia, PA, USA, April 2013.

F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. **Virtual Synchrony Guarantees for Cyber-physical Systems.** In *Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems (SRDS).* Braga, Portugal, October 2013.

O. Landsiedel, F. Ferrari, and M. Zimmerling. **Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale.** In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Rome, Italy, November 2013. ***Best paper award.***

M. Zimmerling, F. Ferrari, R. Lim, O. Saukh, F. Sutton, R. Da Forno, R. S. Schmidt, and M. A. Wyss. **Poster Abstract: A Reliable Wireless Nurse Call System: Overview and Pilot Results from a Summer Camp for Teenagers with Duchenne Muscular Dystrophy.** In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Rome, Italy, November 2013.

F. Sutton, R. Da Forno, R. Lim, M. Zimmerling, and L. Thiele. **Poster Abstract: Automatic Speech Recognition for Resource-constrained Embedded Systems.** In *Proceedings of the 13th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).* Berlin, Germany, April 2014.

F. J. Oppermann, C. A. Boano, M. Zimmerling, and K. Römer. **Poster Abstract: Automatic Configuration of Controlled Interference Experiments in Sensornet Testbeds.** In *Proceedings of the 12th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Memphis, TN, USA, November 2014.

F. Sutton, R. Da Forno, M. Zimmerling, R. Lim, T. Gsell, F. Ferrari, J. Beutel, and L. Thiele. **Poster Abstract: Predictable Wireless Embedded Platforms.** In *Proceedings of the 14th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).* Seattle, WA, USA, April 2015.

R. Lim, M. Zimmerling, and L. Thiele. **Passive, Privacy-preserving Real-time Counting of Unmodified Smartphones via ZigBee Interference.** In *Proceedings of the 11th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS).* Fortaleza, Brazil, June 2015.

Curriculum Vitæ

Personal Data

Name	Marco Zimmerling
Date of Birth	August 14, 1982
Citizenship	German

Education

11/2009 – present	ETH Zurich, Switzerland Computer Engineering and Networks Laboratory Ph.D. under the supervision of Prof. Dr. Lothar Thiele
10/2002 – 08/2009	Dresden University of Technology, Germany Diploma in Computer Science (equivalent to M.Sc.) Minor in Mathematics

Professional Experience

11/2009 – present	ETH Zurich, Switzerland Computer Engineering and Networks Laboratory Graduate research and teaching assistant
01/2009 – 08/2009	SICS Swedish ICT and Uppsala University, Sweden Networked Embedded Systems and Communications Research Group Visting student
06/2006 – 11/2006	IBM T.J. Watson Research Center, Hawthorne, NY, USA Sensors and Actuators Department Intern
10/2005 – 05/2006	IBM Research and Development, Boeblingen, Germany Sensors and Actuators Solutions Department Intern

11/2004 – 03/2005	Dresden University of Technology, Germany Software Engineering Group Undergraduate research assistant
08/2003 – 01/2004	Infineon Technologies, Munich and Dresden, Germany Backend Engineering & Unit Process Development and Memory Products Production Backend Department Intern

Honors and Awards

- Best Paper Award, ACM SenSys, 2013
- Best Poster Award, ACM SenSys, 2012
- Best Paper Runner-up, ACM/IEEE IPSN, 2012
- Best Paper Award, ACM/IEEE IPSN, 2011
- Best M.Sc. Thesis Award, SensorNets School, 2009
- Scholarship, German Academic Exchange Service (DAAD), 2009
- Best Paper Award, German Computer Science Society Informatiktage, 2008