Diploma Thesis

Energy-Efficient TDMA Schedules for Data-Gathering in Wireless Sensor Networks

by

Bernd-Christian Renner

June 2008

Supervised by

Prof. Dr. Volker Turau
Institute of Telematics
Hamburg University of Technology, Germany

Prof. Dr. Hermann Rohling
Institute of Telecommunications
Hamburg University of Technology, Germany

Institute of Telematics
Hamburg University of Technology
TUHH

Abstract

EN In recent years, wireless sensor networks have been frequently adopted for data-gathering. The inherent request for prolonging network lifetime gives rise to the demand for energy-efficiency. In multi-hop networks, this end is met by applying dedicated data-collection phases in combination with Time-Division-Multiple Access (TDMA) protocols for scheduled transmission. Despite the existence of a variety of different scheduling schemes, a detailed comparison has not been carried out. In this thesis, an analytical investigation is therefore provided to reveal the strengths and weaknesses of existing schemes. As the latter exhibit severe disadvantages, a new scheduling scheme, named Spatial Path-Based Reuse (SPR), is devised. Its distributed implementation is frugal and highly efficient, as it combines a small memory footprint with low communication overhead. To permit an in-depth comparison under realistic conditions, a simulation framework for the ns-2 simulator is also developed. The results obtained from extensive simulation with more than 400,000 individual runs substantiate the advantages of SPR, particularly in large networks. Moreover, they reinforce the strengths and shortcomings of the different schemes. The thesis concludes with recommendations for the most suitable scheme for a given data-gathering scenario.

DE In den letzten Jahren wurden drahtlose Sensornetze vielfach zur Datenerfassung eingesetzt. Die inhärente Notwendigkeit zur Steigerung der Lebenszeit eines Netzes bedingt eine effiziente Nutzung der vorhandenen Energie. Dieses Ziel wird durch die Verwendung von dedizierten Phasen zur Datensammlung in Kombination mit TDMA (Time-Division-Multiple Access) Protokollen zur zeitlichen Planung von Funkübertragungen erreicht. Ein Sendeplan wird durch ein Schema beschrieben, das die Sendezeitpunkte der einzelnen Sensorknoten festlegt. Obwohl diverse Verfahren zur Erstellung von Schemata existieren, fehlt bislang ein detaillierter Vergleich. Aus diesem Grund wird in der vorliegenden Arbeit eine analytische Untersuchung durchgeführt, um die Stärken und Schwächen existierender Verfahren aufzudecken. Letztere offenbaren hierbei gravierende Nachteile, so dass ein neuer Algorithmus namens Spatial Path-Based Reuse (SPR) entworfen wird. Seine verteilte Implementierung ist durch die Kombination von geringem Speicherverbrauch mit minimiertem Kommunikationsaufwand sowohl einfach als auch effizient. Um einen umfassenden Vergleich unter realistischen Bedingungen durchführen zu können, wird eine Simulationsumgebung für den ns-2 Simulator entwickelt. Die hiermit aus über 400,000 individuellen Simulationen gewonnenen Resultate belegen die Vorteile von SPR insbesondere in großen Netzen. Ferner bestätigen sie die Stärken und Schwächen der anderen Verfahren. Die Arbeit endet mit Empfehlungen für das jeweils am besten geeignete Verfahren in einem gegebenen Szenario.

Table of Contents

List of Symbols

$\mathcal{V} = \{v_0, \ldots, v_{N-1}\}$	Set of N nodes v_i belonging to a network with sink v_0
$d_{i,j}$	Distance between two nodes v_i and v_j
R_{com}	Communication radius
$R_{int} = \gamma R_{com}$	Interference radius and interference factor
$P_i^T, \quad P_{i,j}^R$	Sending power of node v_i and sensed reception power by node v_j during reception from v_i
α	Pathloss exponent used for modeling wireless signal propagation
$\theta_{cs}, \quad \theta_{rx}$	Carrier-sense and receive threshold. θ_{cs} is the sensitivity of the radio transceiver, θ_{rx} is the signal power required to initiate reception by the transceiver
θ_{int}	Threshold to decide if simultaneous packet reception leads to a collision
ϱ	Node density, i.e., the average number of nodes inside a communication circle with radius R_{com}
$(v_i, v_j) \in \mathcal{E}$	Set of bidirectional communication links in the network
$G = (\mathcal{V}, \mathcal{E})$	Graph representation of a wireless sensor network
$\mathcal{N}_i = \{v_j \in \mathcal{V} \mid i \neq j \wedge (v_i, v_j) \in \mathcal{E}\}$	The set of bidirectional communication partners of node v_i
$\mathcal{T} \subseteq \mathcal{E}, \quad \lvert\mathcal{T}\rvert = N-1$	Routing tree rooted in the sink
$\mathcal{T}_i$	Subtree rooted in (and including) node v_i
$\mathcal{C}_i, \quad C_i = \lvert\mathcal{C}_i\rvert$	The set and number of children of node v_i in $\mathcal{T}$
$C = \max_i \lvert\mathcal{C}_i\rvert$	Maximum number of children of a node in $\mathcal{T}$
$\mathcal{F} = \{v_i \in \mathcal{V} \mid \mathcal{C}_i = \varnothing\}, \quad P = \lvert\mathcal{F}\rvert$	Set of leafs in $\mathcal{T}$ and the number of leafs, or equivalently, the number of paths from a leaf to the sink

$\mathcal{F}_i$	Set of leafs in the subtree $\mathcal{T}_i$ of $\mathcal{T}$
$\{p_0, \ldots, p_{P-1}\}$	Set of paths from each leaf to the sink
$h, \quad h^\star, \quad h_i, \quad \ell$	Depth h of a tree $\mathcal{T}$, the minimum depth $h^\star$ of a corresponding minimum-depth tree, and the depth h_i of an individual node v_i. $\ell \in \{0, \ldots, h\}$ refers to a level of $\mathcal{T}$
$\mathcal{S}, \quad \lvert\mathcal{S}\rvert = R$	Set of slots with round length R assigned by a TDMA scheme. Indices I, II, III, and SPR are used with R to indicate a particular TDMA schedule
$\mathcal{S}_i = \{s_i^0, ..., s_i^M\}$	Slots assigned to node v_i
T	Runtime of a data-gathering collection phase. Indices I, II, III, and SPR are used to indicate the runtime of a particular TDMA schedule. $\uparrow$ and $\downarrow$ are used, if slots are assigned in ascending or descending order from leafs to the sink
σ	Maximum throughput at the sink, i.e., the number of equally sized data packets collected per time unit
$L_i, \quad \tilde{L}_i$	Initial and current buffer fill level of a node v_i
$L_i^* = \sum_{v_j \in \mathcal{T}_i} L_j$	Load of a node v_i, i.e., the sum of packets it has to forward
$B, \quad \tilde{B}$	Buffer size of each node in the network and the soft limit of the buffer used for flow control
$\omega_c, \quad \omega_p$	The number of (sending) slots to be skipped by a child, in case its parent encounters a buffer overflow (ω_c); and the number of (listening) slots to be skipped by a parent, if its child experiences a buffer underrun (ω_p).
r	Maximum number of packet retransmissions before the corresponding link is declared interrupted
η	Multiplier for the communication radius R_{com} used to model the area in which a Type I slot assignment does not reassign a slot
λ	Parameter used by Type III to assign slots to a node v_i according to its initial buffer fill level L_i
κ	The (maximum) number of slots alloted to each path p_m by the SPR slot assignment
$\mathbf{d}_i, \quad \mathbf{o}_i$	Displacement and offset vector determining the set of slots $\mathcal{S}_i$ assigned to node v_i by SPR

Chapter 1

Introduction

Monitoring and controlling of processes imposes the need to gather data, which can be performed by sensors. To evaluate the data, two traditional solutions exist. The first one is to store data locally and collect it manually. However, this is time-consuming, labor-expensive, impractical in harsh environments, and renders prompt evaluation impossible. The second one is to wire up sensors, which makes their readings instantly available. Yet, this solution is invasive, intricate and inflexible, costly, and thus generally infeasible. Recently, a new option has arisen. The advance in micro computing has put forth so-called wireless sensor nodes. These are tiny, low power, embedded boards, equipped with sensing devices, a processing unit, and a radio module for wireless transmission of data. Mass production has made them affordable. The list of their features already reveals their suitability and superiority to fulfill the requirements of data-gathering. Their wireless connectivity renders fast access to data and the deployment close to or even inside phenomena possible, even under harsh or dangerous environment conditions. Their size allows for the deployment of a large number of nodes, so that a non-invasive monitoring solution with high spatial resolution is achievable.

Monitoring applications can be subdivided as follows. In the one-to-one communication paradigm, sensor nodes generate events and generally report them to a destination that may differ depending on the event and the node generating it. In the many-to-one paradigm, each sensor node periodically performs measurements, possibly pre-processes and finally reports them to a central node. The many-to-one paradigm is frequently adopted in data-gathering scenarios, such as environment or agricultural monitoring. The latter has recently been employed to observe climate, soil and pasture. A particularly challenging example is that of tideland monitoring,

where sensor nodes are placed off shore in order to sample data and only report their readings during ebb tide.

Even though much research has been dedicated to the many-to-one paradigm, open issues inherent in the nature of wireless sensor networks require further investigation. Because of their small size and relatively low cost, wireless sensor nodes provide low computation power and few resources concerning memory. Furthermore, wireless transmission range and bandwidth are limited. Hence, nodes may not be able to directly report their sensor readings to a central node, so that data has to be forwarded by other nodes using multi-hop routing. Finally, sensor nodes are equipped with low-capacitive batteries only. Batteries are required for autarkic and autonomous operation. However, they must be small in order not to spoil the advantage of a sensor node's small size. As a result, nodes must operate as energy-efficient as possible so as to increase network lifetime.

As the radio transceiver of a sensor node consumes the largest part of the energy, it must be switched off whenever possible and efficiently used when switched on. Time-Division Multiple Access (TDMA) of the channel promises to meet this end by preventing the major sources of inefficiency: idle listening, overhearing, and collisions. This is achieved by setting up a sending and listening schedule for each node, which requires a distributed algorithm. Although many different scheduling schemes have been proposed, comparisons are rather focused on individual solutions than on the different principles of schedules. Hence, an ideal solution for the many-to-one paradigm has not been identified so far.

This thesis encompasses three objectives. Firstly, a theoretical analysis of the scheduling problem and the capability of the existing schemes is to be conducted. Secondly, the outcome of this analysis shall be employed to design a new scheduling scheme. Thirdly, a comparison between this new scheme and the existing ones ought to be performed.

Having identified the weaknesses of the competitors through the theoretical analysis, a new, light-weight TDMA schedule for data-gathering, named Spatial Path-Based Reuse (SPR) scheme, is developed. Its distributed implementation is frugal and highly efficient, as it combines a small memory footprint with low communication overhead. To demonstrate and evaluate the advantages of SPR, an in-depth comparison via simulation is necessary. For this purpose, a simulation framework based on ns-2 is conceived and implemented, so that the analysis can be performed under realistic conditions. This comparison particularly includes energy-efficiency and the overall time required to forward a given amount of data to the central node. The

latter is an important criterion. Firstly, the time available for forwarding data may be restricted as in the mentioned tideland application. Secondly, decreasing the time required for collecting all data allows to increase the sampling rate and data resolution.

The results obtained from more than 400,000 individual simulation runs show that SPR produces schedules that effectively minimize the overall time required to forward all data in large networks. Here, SPR profits from spatially reusing slots and its ability to prevent buffer congestion, which influences the performance of its opponents severely. Furthermore, SPR achieves energy-efficiency close to the possible minimum. In conclusion, SPR outperforms its competitors particularly in large networks and thus advocates itself to be adopted in TDMA-driven data-gathering applications. However, it is, e.g., not suitable in small and in sparse networks, where other approaches show to be favorable. For this reason, the simulation results are also used to identify the strengths and advantages of the different schedules. This knowledge is exploited to devise a schema that encompasses suggestions on which schedule to employ depending on the respective scenario.

Chapter 2

State of the Art

In this chapter a general overview about wireless sensor networks and their characteristics will be given. Based on this, it follows an introduction to monitoring applications using those networks. Finally, a detailed analysis on the adoption of energy-efficiency will be carried out.

2.1 Wireless Sensor Networks

In the recent past the advance in micro computing has put forth so-called *wireless sensor nodes*. They can be described as tiny, low power, embedded boards, equipped with sensing devices, a processing unit, and a radio module for wireless transmission of data. Estrin et. al. provide an in-depth analysis of the opportunities of wireless sensor networks [ECPS02]. In particular, their wireless connectivity allows for deploying them close to or inside phenomena, even in harsh or dangerous environments. Due to their size, a non-invasive monitoring solution is attainable with high resolution. Wireless sensor nodes have therefore become a suitable and convenient solution for new possibilities of sensing; application fields can be divided into ecological [MPS+02], agricultural [WCS+07], environmental [MPR+05, SWC+07, CO05], and structural [KPC+06] monitoring.

Those applications have in common that they are built upon a *many-to-one* communication scheme: All nodes in the network report their sensor readings to a central node, the *sink*. This pattern can be found in most monitoring applications and is often regarded as *data-gathering*. Before a detailed introduction to this topic will be provided, a general understanding of wireless sensor networks is required. Therefore, their general characteristics will be presented briefly.

2.1.1 Characteristics and Challenges

In spite of their general suitability for many applications, wireless sensor networks have their weaknesses. Due to their size and relatively low cost, wireless sensor nodes are restricted to only provide low computational power and few resources concerning memory, wireless transmission range, and bandwidth. Furthermore, nodes have a limited amount of energy, because they are generally battery-powered. This is a necessity in order to realize unattended deployments. However, those batteries are low-capacitive, as to meet the desire for small sized and low priced sensor nodes. Since battery replacement is usually not feasible in large networks or even impossible in harsh environments, conservation of energy is a crucial point in order to maximize network lifetime.

As a result of the restrictions concerning memory and computational power, application and protocol layout must be done carefully. Firstly, saving memory is an important design feature. If, e.g., nodes must store information about remote nodes within their communication range, the data to store must be reduced as much as possible. Secondly, expensive and recurring computations must be avoided. Tradeoffs between conservation of memory, energy, and computing time are commonly required.

To prolong the lifetime of a network, energy-efficiency is of vital importance. As the radio is the heaviest energy consumer [SLR+05], sensor nodes offer sleep modes, in which they switch off the radio and sensing devices. The fraction of time a node actually spends outside a sleep mode is called *duty cycle*. Conserving energy implies to reduce the latter. Hence, sleep modes should be entered whenever possible. If wireless communication is required, efficient usage of the radio is mandatory. This must be realized by the *Medium Access Control (MAC)*, which is responsible for controlling the radio. In the following, the main origins pertaining energy wastage by wireless radio communication will be explored.

The most obvious source of energy dissipation is that of signal *interference*. Here, a signal results from packet transmission. Interference occurs, if a node receives signals from two or more different senders at the same time. If at least two of the packets are destined to that receiver, this is called a *collision*. An attempt to avoid interference is the application of *contention-based*, also called *Carrier-Sense Multiple Access (CSMA)*, protocols that do a carrier sense before starting a transmission. If the sensed signal is below the carrier-sense threshold, the sender assumes a clear channel and commences transmitting. If not, a new carrier sense will be started after a backoff. However, this method is inappropriate to solve what is called the hidden-terminal problem. Two nodes that are outside each other's communication range can start

timely overlapping transmissions. If at least one of the intended receivers is inside the communication range of both senders, there will be interference.

Idle listening on the radio channel already consumes a multiple of energy as compared to the sleep modes. Additionally, listening on the channel leads to *overhearing*. Here, nodes receive (overhear) packets not destined to them, which causes wastage of energy due to useless reception and packet processing [BR04]. Measurements of energy consumption during packet reception for the Scatterweb Platform [SLR+05] can be found in [TW07a]; they reveal that reception and transmission draw approximately the same current. Both problems are attenuated by scheduled MAC protocols; particularly *Time-Division Multiple Access (TDMA)* is a promising approach. Here, time is divided into transmission slots, and slots are assigned to nodes.

In the context of reducing radio usage, packet size is yet another aspect. On the one hand, short packets lead to large overhead caused by packet headers. On the other hand, long packets are more likely to be corrupted by bit errors. To increase reliability in the face of packet loss, retransmissions and forward error correcting codes can be used [JE07]. Discussions on packet length and energy-efficiency can be found in [SAM03, Joe05]. Yet, the choice of optimal parameters in this context is difficult, as it largely depends on the radio chip, protocol, and environment.

The limitations of wireless sensor nodes have to be addressed at different levels. Memory and computational restrictions are application specific and will be considered during application and protocol design. Due to the relevance of interference, models will be presented in the following, preceded by a formal abstraction of wireless sensor networks. Idle listening and overhearing will be addressed during the discussion of data-gathering applications and MAC protocols in Sections 2.2.4 and 2.3. The problem of finding optimal packet sizes and integrating forward error correcting codes will not be explicitly addressed in this thesis. They are regarded as an orthogonal problem.

2.1.2 Formal Abstraction

A *wireless sensor network* can be described as a graph $G = (\mathcal{V}, \mathcal{E})$. $\mathcal{V} = \{v_0, \ldots, v_{N-1}\}$ denotes the set of equally equipped sensor nodes (including the sink v_0), and $\mathcal{E}$ is the set of *links* between nodes. A tuple (v_i, v_j) is a member of $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, if and only if a reliable, bidirectional communication between v_i and v_j is possible.

In a real network, links are often unstable and asymmetric [ZK07]. Hence, a node v_i must keep track of its *neighborhood* $\mathcal{N}_i$. Here, $v_j \in \mathcal{N}_i$ implies that v_j is inside the

communication range of v_i, i.e., the distance $d_{i,j}$ between v_i and v_j is at most the *communication radius* R_{com}. By calculating and exchanging link-qualities with its neighbors, a node can identify bidirectional and reliable links. The Wireless Neighborhood eXchange (WNX) protocol is an example protocol dedicated to this task [TRV+05]. Related to neighborhoods, an important characteristic of a wireless sensor network is its *density* ϱ. It is defined as the average number of neighboring nodes or, alternatively, the average number of nodes inside a circle area with radius R_{com}. Thus, it can be expressed as $\varrho = \frac{1}{N}\sum_i 1 + |\mathcal{N}_i|$.

Throughout this thesis, links determined by the neighborhood protocol are assumed to be stable over time. Antennas are expected to be omni-directional, and signal propagation does not depend on node positioning or orientation. Additionally, all nodes are in line-of-sight, use the same transmitting power and generally have the same physical characteristics. However, this does not imply that packets between two nodes v_i and v_j with a link $(v_i, v_j) \in \mathcal{E}$ can be exchanged without packet loss or interference.

2.1.3 Interference in Wireless Communication

Interference is a serious problem in data-gathering applications: All data is transported to the sink, so that traffic, and thus interference, increases with closeness to the sink. A detailed analysis of interference using field studies is presented in [ZHSA05]. In addition, different models have been developed in order to permit an interference analysis.

The Fixed Power Protocol Interference Model (FPPrIM) discussed in [RRDM08] makes use of the *interference radius* $R_{int} = \gamma R_{com}$. It is commonly assumed that $1 \leqslant \gamma \leqslant 2$. A transmission from v_i to v_j with $d_{i,j} \leqslant R_{com}$ is interfered, if the receiver v_j is inside the interference radius of a simultaneously transmitting node v_k:

$$d_{k,j} \leqslant R_{int} = \gamma R_{com} \tag{2.1}$$

An example is shown in Figure 2.1. Here, v_5 may cause interference on a transmission from v_{13} to v_{12}.

A more realistic model uses the receiving power of signals in order to detect interferences. If a signal is sent at v_i with a transmitting power P_i^T, it will be received at v_j with power $P_{i,j}^R = P_i^T \cdot (d_0/d_{i,j})^{\alpha}$. Here, α is the *pathloss exponent* usually taken from the range $[2, 4]$, and d_0 is a reference distance. A transmission from v_i to v_j is

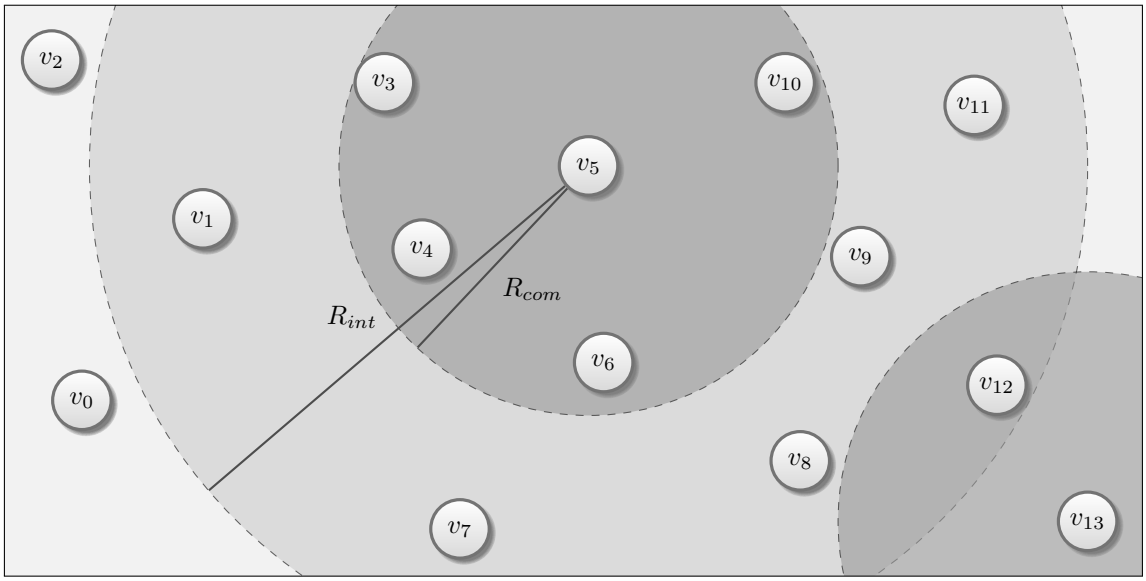


Figure 2.1: FPPrIM-based interference with $R_{int} = 2\,R_{com}$

disturbed by a simultaneous transmission from v_k, if the ratio $P^R_{i,j}/P^R_{k,j}$ is below the *interference threshold* θ_{int}:

$$\frac{P^R_{i,j}}{P^R_{k,j}} < \theta_{int} \tag{2.2}$$

The latter is a parameter of the radio chip and can be understood as its sensitivity to interference. This model is used, e.g., in the implementation of the IEEE 802.11 standard for ns-2 with $\theta_{int} = 10$ [FV08, Liu]. With $\alpha = 2.7$ [SRS90] and equal transmission powers, the situation in Figure 2.1 changes as follows: A transmission from v_{13} to v_{12} cannot be disturbed by v_5. Yet, v_5 may still interfere a transmission from v_{12} to v_8.

In wireless sensor networks, it frequently occurs that more than two nodes are transmitting at the same time. Hence, just comparing one receiving power with another gives only a part of the picture. The *signal-to-noise+interference-ratio (SINR)* takes all signals on the channel into consideration, including the ambient noise N_0. Using this model, a transmission from v_i to v_j is disturbed, if the following equation is satisfied:

$$\text{SINR} = \frac{P^R_{i,j}}{N_0 + \sum_{k,k\neq i,j} P^R_{k,j}} \;<\; \theta_{int} \tag{2.3}$$

An even more elaborate, statistical model [ZK04] is actualized by considering bit error rates (BER) in favor of the static threshold θ_{int}. For this it uses the SINR in combination with an error function that depends on the used modulation, e.g., On-Off-Keying yields

$$\mathrm{BER} = \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{\frac{\mathrm{SINR}}{2}}\right) \quad \text{with} \quad \mathrm{erfc}(z) = \frac{2}{\sqrt{\pi}}\int_z^\infty \mathrm{e}^{-\zeta^2}\mathrm{d}\zeta \tag{2.4}$$

Here, erfc(.) is the result of assuming signals to be normally distributed random variables and performing integral substitution. Besides modulation, this model accounts for packet length, encoding, and forward error control, since the packet reception rate is calculated by means of BER and those three components. For this model an implementation for ns-2 is available [Unt08].

2.2 Data-Gathering in Wireless Sensor Networks

As outlined in Section 2.1, data-gathering is a common task in wireless sensor networks. Nodes are deployed in order to collect and report sensor readings to a single sink, that, e.g., writes all data to a database. In general, the covered area of a many-to-one setup exceeds the communication range of sensor nodes. Besides, reducing the communication range by decreasing transmission power helps saving energy and avoids interference. In this case, a tree rooted in the sink provides a suitable routing environment. In order to achieve reliable communication, an energy-efficient mechanism avoiding packet loss is needed. Additionally, nodes possibly have to store own sensor readings and must forward foreign data towards the sink. Hence, buffer management must be considered.

When setting up a data-gathering application, some major issues have to be considered. Firstly, *reliability* may be required. Here, it is defined as the property of all sensor readings eventually reaching the sink. Another important feature is the *throughput* σ, i.e., the number of equally sized data packets received by the sink per time unit. *Latency* is defined as the time elapsing from generation of a sensor reading by a node and reception by the sink.

2.2.1 Strategies for Collecting Data

There are two different strategies for the collection of data. The first is to forward sensor readings in direction of the sink shortly after they become available. This *on-demand* approach is necessary, if there are time constraints, i.e., low latency is required. The second strategy is to employ a *two-phase system*: During the *sensing phase*, nodes store all sensor readings in a buffer. Periodical *collection phases* are

used to forward bulk data to the sink [TW07b]. Of course, this system is applicable, only if all sensed data is delay-tolerant.

The optimization of energy-efficiency, crucial for the lifetime of a sensor network (cf. Section 2.1.1), demands low duty cycles. Provided delay tolerance, the two-phase strategy offers to meet this end by exploiting a priori knowledge of the traffic pattern. Firstly, it is sufficient to create a new routing tree at the beginning of each collection phase. This disencumbers the application from continuous tree maintenance as required for on-demand routing. Additionally, sending schedules can be set up after tree construction, so that parents know when their children will be transmitting packets. By knowing the mapping between receiving slots and the corresponding children, a node is enabled to detect interrupted links and can thus stop listening in those slots. Finally, during the remainder of the collection phase, a node forwards data until its buffer runs empty and no more packets are expected from its children. It then informs its parent, which in turn is allowed to stop listening to that node. As a result of the whole procedure, idle listening is completely prevented.

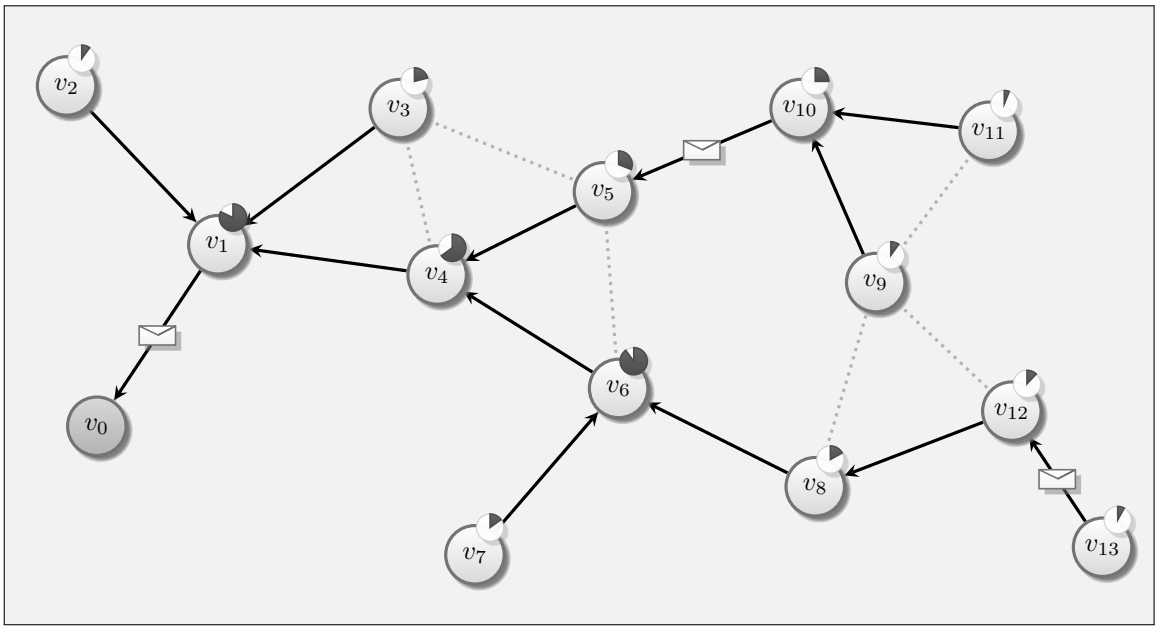


Figure 2.2: Data-gathering setup in a wireless sensor network

An example for a data-gathering application during a collection phase is depicted in Figure 2.2. Dotted edges between nodes show possible communication links, solid ones indicate the routing tree, pointing from a child to its parent. Envelopes symbolize ongoing transmission of a packet on the corresponding link. Here, e.g., the transmission schedule allows concurrent transmission of nodes v_1, v_{10}, and v_{13}. Buffers are displayed by the small circles at the upper right of each node, where the dark part is

the current fill level. The example illustrates the high buffer fill level of inner nodes, which have to forward remote packets in addition to sending their own ones. Note that the sink v_0, which is displayed in a darker shade, does not have a buffer, because connection to a database is presumed.

2.2.2 Data-Gathering Tree

A *data-gathering tree* $\mathcal{T} \subseteq \mathcal{E}$ is a spanning tree of G rooted in the sink. Subtrees of a node v_i, including v_i, are denoted $\mathcal{T}_i$. Each node v_i has one *parent* (except for the sink) and a set of *children* $\mathcal{C}_i$, which is empty for leaf nodes. For convenience, $C_i = |\mathcal{C}_i|$ will be used. $C = \max_i C_i$ is called the maximum number of children or maximum *tree degree*. The set of leaf nodes is given as $\mathcal{F} = \{v_i \in \mathcal{V} \mid \mathcal{C}_i = \varnothing\}$, and $\mathcal{F}_i$ denotes the leafs in the subtree $\mathcal{T}_i$.

Zhou et. al. [ZK03] compare approaches realizing different parent selection strategies. Xue and Fumar [XK04] state that network connectivity is ensured with high probability, only if C is in the magnitude of $\log N$. This is an important finding, since completeness of the tree is an imperative for a data-gathering application.

Tree construction largely depends on the *topology*, which is basically characterized by the number of nodes N, the position of the sink, the density ϱ, and the distribution of nodes. An important, energy-relevant factor of tree construction is the resulting overall *depth* h of $\mathcal{T}$ and the individual node depths h_i. This is because every packet originated in v_i must be sent at total h_i times, until it reaches the sink. A lower bound of h is given by the quotient of the maximum distance from a node to the sink v_0 and the communication radius: $h \geqslant \max_i \left\lceil \frac{d_{0,i}}{R_{com}} \right\rceil$. Usually, $C \ll \varrho$ does not lead to a considerably larger h. The reason of this is the potentially exponential growth C^h of nodes with increasing tree depth, whereas the density ϱ usually stays constant or even decreases with increasing distance from the sink.

Besides depth and overall energy consumption, the number of packets a node has to forward must be accounted for [YW08]. Consequently, nodes close to the sink send more packets and thus consume more energy. Leaf nodes, in contrast, only send their own packets and consume the least energy in the network. As a result, network lifetime is sensitive to the node consuming the most energy. Therefore, balanced trees lead to more equally distributed energy consumption among nodes with an equal depth in $\mathcal{T}$, which increases overall network lifetime. Yet, the construction of maximum-lifetime trees for data-gathering is $\mathcal{NP}$-complete [YW08], but breadth-first search already produces sufficiently balanced trees [DH03].

Link-qualities may vary over a longer period, and in consequence, routing trees can become partitioned. One solution to this problem would be to allow nodes having more than one parent [TTS05]. Yet, this produces idle listening of the potential parents. Additionally, tree maintenance imposes communication overhead and may affect tree depth and balance. This underlines the advantages of the two-phase data-gathering strategy presented in Section 2.2.1.

2.2.3 Reliable Transmission and Buffer Management

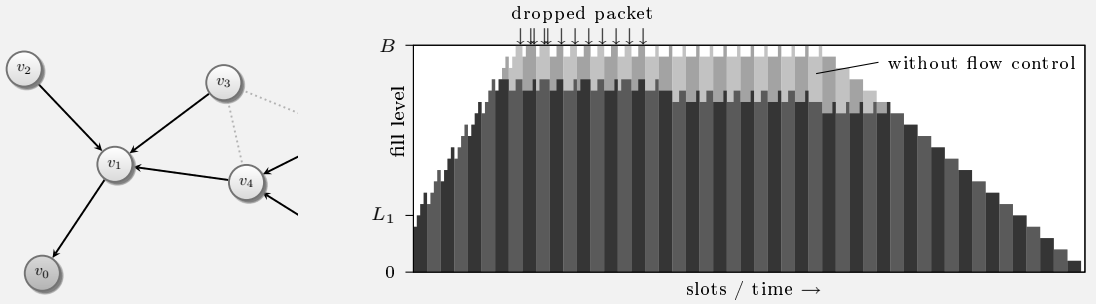
Reliability is an important issue in data-gathering applications, because loss of data may render the whole experiment bootless. In order to achieve reliable end-to-end communication, i.e., all data collected in the network is sent to the sink without loss, in an energy-efficient way, hop-to-hop *acknowledgments (ACK)* are preferable [EB04, SH03]. In addition, a sender gains fast feedback about successful packet reception at the next hop, so that detection of lost packets is simplified.

By using hop-to-hop acknowledgments, each node must store received packets in a local buffer, until they have been successfully forwarded to the next hop. In a routing tree, a node may have several children, whereas it (usually) has precisely one parent. Therefore, the number of incoming packets is potentially higher than that of outgoing ones, if every node has the same amount of time for packet forwarding. This may lead to buffer congestion [EB04]. Amplification of this problem due to packet loss, especially on the link to the parent, is likely. In general, three important characteristics on this topic are the initial *fill level* L_i of a node v_i, its expected *load* $L_i^* = \sum_{v_j \in \mathcal{T}_i} L_j$, and its *buffer size* B.

Flow control is required in this situation to serve two purposes: Firstly, it must timely detect and avoid buffer overflows. Secondly, if a buffer overflow cannot be avoided, it must prevent futile transmissions [EB04]. In general, this kind of flow control can be achieved via backpressure. Including according information into acknowledgments keeps communication overhead low, so that energy-efficiency is preserved. In the presence of flow control, throughput performance may be affected by buffer sizes [BBLV05]. However, given the two-phase collection strategy as presented in Section 2.2.1, flow control must also ensure that enough buffer space will be left for the next sensing phase.

Figure 2.3(a) illustrates a situation, in which node v_1 receives packets from three nodes and must forward all data to node v_0. Assuming equal transmission times for each node, possible buffer usage is depicted in Figure 2.3(b) for active flow control

(dark) and no flow control (light). Shadings are used to embrace sets of four slots consisting of one for each node. Arrows at the top indicate dropped packets due to buffer overflow. Note that flow control prevents buffer overflow, but does not affect the number of slots required to forward all data.



(a) traffic situation (b) resulting buffer fill level of v_1

Figure 2.3: Buffer fill level with and without flow control

Enforcing reliability may lead to packet duplicates, if acknowledgments are lost. Since nodes remove successfully forwarded packets from their buffer, they may not be able to identify duplicates by scanning through their buffer on a new packet reception from a child. Attenuation of this problem can be achieved by using MAC sequence numbers or packet hashs [TW07a].

2.2.4 Existing Approaches

In the early stages of data-gathering applications using wireless sensor networks, approaches have been fairly intuitive [MPS+02]. Sensor readings have been sent to the sink via routing trees, as soon as they became available. This was mainly achieved by purely contention-based MAC protocols. Packets have been forwarded using best-effort strategies, i.e., using MAC layer acknowledgments and transmission retries up to a given maximum. With time passing and experience increasing, applications have become more elaborate. Particularly energy-efficiency has been addressed in order to prolong network lifetime. This process will be documented in this section.

Directed Diffusion [IGE00] is one of the earliest data-gathering protocols for wireless sensor networks. It allows for best-effort data collection without making use of flow control or acknowledgments. Hence, reliability is not offered. A sink periodically injects (floods) an interest for data into the network. Nodes spread interests and set

up gradients on their reception, so that directed paths to the sink are formed. Good gradients, according to a given metric, are reinforced; thus, fast adaptation of high quality paths is achieved. This results in high overhead, as packets are likely to traverse multiple paths. However, the authors address energy-efficiency by particularly suggesting TDMA, although Directed Diffusion has been simulated using CSMA.

RMST [SH03] is the Reliable Multi-Segment Transport protocol for data-gathering based on Directed Diffusion on top of a contention-based MAC. To achieve reliability and error recovery, it employs a selective hop-to-hop negative acknowledgment (NACK) strategy. Only so-called caching nodes are able to detect packet loss and can reinforce loss recovery. Routing of packets is left up to the diffusion layer.

The Event-to-Sink Reliable Transport (ESRT) protocol presented in [SAA03] is an energy-preserving approach for event-detection applications. However, reliability as defined at the beginning of this section is not obtained. ESRT only ensures the collection of a required minimum number of reportings of an event. Energy-efficiency is achieved by running most algorithms on the sink and by applying congestion control. The latter is additionally used to adapt the event reporting frequency. Here, the number of reportings is adjusted to keep the transmission volume low, while conserving the minimum reporting frequency. Routing is done using Dynamic Source Routing (DSR) in combination with a contention-based MAC, resulting in idle listening.

Data-gathering using the Reactive, Opportunistic Protocol for Environment monitoring (ROPE) is discussed in [CO05]. ROPE schedules transmissions in bulks using a path feedback system in order to decrease packet loss. Acknowledgments are sent after every tenth packet received and include a list of sequence numbers of successfully received packets. Reliability is only enforced up to a given percentile. On the one hand, energy consumption is reduced by data compression. On the other hand, idle listening is extensively used in order to achieve 2-hop data delivery. Due to the lack of a routing mechanism—ROPE is originally designed for single-hop applications—, nodes with connectivity to the sink offer data relay to nodes that do no receive an acknowledgment after having transmitted data to the sink. Relay-offering nodes gain their information by overhearing, so that a contention-based MAC is used.

FlexiMAC [LDCO06] is a highly integrated transport protocol based on TDMA. It promises fair channel access and high throughput, but does not offer reliability by itself. Communication failures are overcome by local repair, and buffer usage is minimized. Energy consumption is reduced by adjusting transmission power. FlexiMAC takes care of implicit routing tree construction and maintenance as well as transmission scheduling. Yet, this involves a high degree of complexity and overhead.

	Diffusion [IGE00]	RMST [SH03]	ESRT [SAA03]	ROPE [CO05]	FlexiMAC [LDCO06]	Dozer [BvRW07]	[TW07b]
MAC Type	both	CSMA	CSMA	CSMA	TDMA	TDMA	TDMA
Multi-Hop	✓	✓	✓	(✓)	✓	✓	✓
Tree-Routing	(✓)	(✓)	–	–	✓	✓	✓
ACK Usage	–	✓	–	(✓)	–	✓	✓
Reliability	–	✓	–	–	–	✓	✓
Buffering	–	✓	✓	–	✓	✓	✓
Flow Control	–	–	✓	–	–	✓	✓

Table 2.1: Basic characteristics of different data-gathering approaches

An ultra-low power application for data-gathering, called Dozer, is introduced in [BvRW07]. In order to reduce energy wastage caused by the communication subsystem on individual nodes, Dozer is highly cross-layered. Transmission scheduling is accomplished using a light-weight TDMA protocol. Reliable collection of periodic, delay-tolerant data is done using a routing tree, which is computed once by trading depth off against subtree load. In case of link failures, nodes may chose from a list of secondary parents. If no parent is available, a node has to rescan the neighborhood. In addition, a suspend mode is used to compensate for temporary link failures and resulting tree partitioning. It is entered by a node that cannot find a (new) parent in the tree. After a timeout, a suspended node wakes up and attempts to get reconnected to the tree.

Long-term reliable data-gathering has been addressed in [TW07b]. Periodically, routing tree construction is initiated by the sink using breadth-first search. Hence, link and temporary node failures are coped with by tree rebuilding. In the following collection phase, reliable transportation of sensor readings towards the sink is accomplished by acknowledgments. Energy-efficiency is preserved by deploying a TDMA protocol, eliminating duplicate packets, and implementing congestion detection. Finally, nodes enter a sleeping mode after they have forwarded all data of their subtree.

The characteristics of the given examples, summarized in Table 2.1, disclose the transition from contention-based MAC protocols to their TDMA counterparts in energy-efficient data-gathering applications. Delay tolerance is assumed in all cases, but solely later approaches make use of buffering. Closely related to this, recent works show the importance of congestion control in terms of preservation of energy and make use of hop-to-hop acknowledgments to assure reliability. Cross-layering is yet another development in minimizing the wastage of energy and thus prolonging overall network lifetime. Furthermore, routing trees have been found to be most

suitable for the data-gathering, many-to-one scheme, although no clear trend can be observed towards either periodic tree reconstruction or continuous maintenance. However, the former appears more suitable in a two-phase data-gathering system, since link-qualities may considerably change between two collection phases. Modern protocols introduce a suspend mode, in case a node becomes disconnected from the tree. Depending on the data-gathering strategy, nodes wake up in the next collection phase or after a reconnect timeout, respectively. Thus, temporary node failures and interrupted links can be compensated with low overhead.

2.3 MAC Protocols for Wireless Sensor Networks

Section 2.2.4 has briefly sketched that different MAC protocols have been used in data-gathering applications. However, TDMA has recently become the first choice, but different flavors have been developed and deployed. Therefore, this section will provide a general comparison of data-gathering MAC protocols, followed by a detailed look at TDMA scheduling variants and slot assignment.

2.3.1 Comparison of MAC Protocols

Research in the field of wireless sensor networks has brought up a vast variety of MAC protocols [DEA06]. In general, two different classes can be identified: CSMA and TDMA. The former basically relies on carrier sensing—to assess the channel—before starting a transmission. Reception is done whenever a signal is detected. As this implies extensive idle listening, advanced schemes use synchronized schedules: During sleep periods, nodes switch off their transceiver. Only in between those periods, nodes (simultaneously) assess the channel and may start transmission or reception. TDMA protocols divide time into rounds that consist of slots of usually same length. Slots are assigned to nodes (or links), and a node may only start transmission in its own slot(s). Appropriate scheduling completely prevents collisions; exchanging schedules between neighbors reduces overhearing and idle listening. However, TDMA requires tight node synchronization and increases latency in case of low traffic. Recently, hybrid protocols have been proposed. They enable nodes to employ TDMA or CSMA depending on local traffic. In the following, representatives of the different types of MAC protocols will be portrayed and analyzed.

Initially, contention-based (CSMA) protocols were considered mainly [WC01]. Early adoptions for wireless sensor networks are S-MAC [YHE02, YH04] and B-

MAC [PHC04]. S-MAC is a port of 802.11, whereas B-MAC has been tailored to the needs of wireless sensor networks. Among its design goals are low-power operation, effective collision avoidance, efficient channel utilization, and adaptivity to changes in radio connectivity. E.g., it makes use of low-power listening and scheduled sleeping for energy conservation; collision avoidance is addressed by carrier sensing and back offing. However, being designed as a general-purpose MAC protocol, B-MAC reveals weaknesses in a many-to-one paradigm with a single destination. Firstly, it severely suffers from collisions [RR08, AHM+06]. Secondly, it generates high latency, because sleep schedules make no use of the tree structure used in a data-gathering application. Additionally, it is susceptible to overhearing due to the same reason. D-MAC explicitly solves those issues by means of advanced schedules [LKR04]. Those are set up by taking the depth of each node in the routing tree into consideration. Since D-MAC is contention-based, it conserves the ease of adding new nodes to the network, but is still prone to collisions. Overhearing and idle listening are likewise problematic.

Since contention-based MAC protocols are likely to suffer from collisions, this may lead to considerable forfeits in throughput under heavy traffic [AHM+06, WR05]. The hybrid Z-MAC [RWAM05] protocol tries to solve this shortcoming by falling back to a TDMA schedule, if congestion is discovered. Another hybrid approach, the Funneling-MAC, has been shown to outperform both B-MAC and Z-MAC in a real-world data-gathering experiment with 45 nodes [AHM+06]. This is achieved by accounting for the funneling effect, which can be explained as the increase of network traffic with closeness to the sink. To overcome funneling, TDMA is used on nodes close to the sink and CSMA in the outer, less dense regions of the network. For this, the sink sends out beacons with transmission power adapted to the needs of the network. Nodes receiving these beacons consider themselves close to the sink and switch to TDMA. Since both TDMA and CSMA are used in the same network, a complicated scheduling mechanism is used. Its most important task is to avoid interference, which imposes a considerable problem on nodes on the edge of the TDMA region, since they must operate both MACs. In conclusion, Funneling-MAC outperforms B-MAC and Z-MAC at the expense of a considerable amount of complexity. Additionally, none of the inherent problems of CSMA—collisions, overhearing and idle listening—have been completely eliminated.

TDMA protocols have recently drawn increasing attention, as they are generally capable of solving those problems. NAMA [BGLA01] produces unique slot assignments in a distance of two hops. However, it is neither energy-aware nor free of overhearing. Built on its foundation, TRAMA [ROGLA03] has been designed to overcome those

deficiencies. Slots are scheduled traffic-adaptively, and energy-efficiency is achieved by allowing nodes to go into sleep modes, if they are not sending or receiving any data. This is accomplished by exchanging transmission schedules between nodes in random access periods (comparable to pure CSMA). For successful exchange, all nodes have to be listening during those phases. In contrast to this, FLAMA [ROGLA05], an improvement over TRAMA, relies on traffic information provided by the underlying application. As simulations have revealed [ROGLA05], FLAMA outperforms TRAMA and S-MAC in terms of energy-efficiency. Yet, there are two notable drawbacks. Firstly, FLAMA (like NAMA and TRAMA) is likely to produce collision-afflicted schedules [ZHSA05]. Secondly, FLAMA uses random access periods, which unnecessarily compromise energy-efficiency.

Another TDMA protocol designed for data-gathering is TDMA-EC [RXMC05]. Slots intended for sending and receiving are staggered with respect to node depths, so that continuous flows of packets from source to sink are maintained, the intention being a massive reduction in latency as compared to D-MAC. Although TDMA-EC has been shown to achieve energy-efficiency comparable to D-MAC [RXMC05], it still produces overhead due to random slot selection and broadcast messages during actual data-gathering.

The application described in [TW07a] (cf. Section 2.2.4) makes use of a static TDMA scheme with one slot for each node. Assignment is done according to node identifiers, so that each node can directly determine its own slot and the ones of its children in the routing tree. As a result, collisions, overhearing, and idle listening are completely avoided. However, latency, throughput, and buffer usage are particularly affected by growing network size. An improvement has been reported by the same authors in [TW07b]: Nodes can give up their slots, so that other nodes along the path to the sink can reuse them. Different strategies of where to reuse slots have been discussed and analyzed by simulation. Since notifying nodes about additional slots implies additional information exchange, this approach is a tradeoff between throughput and energy-efficiency. Note that this modification is possible solely in a two-phase data-gathering system.

Table 2.2 gives an aggregated overview of the strengths and weaknesses of different MAC protocol types. In general, TDMA protocols have been shown to outperform contention-based MAC protocols in data-gathering applications regarding energy consumption [SYL06, ROGLA05, WR05]. However, the presented TDMA protocols reveal significant differences, particularly in terms of scheduling. Many protocols offer dynamic slot reassignment, which is required in case of changes in the routing tree.

	Pure CSMA	Scheduled CSMA	Hybrid	TDMA
Overhearing	⊖⊖	⊖	○	⊕ / ⊕⊕
Idle Listening	⊖⊖	⊖	○	⊕ / ⊕⊕
Collisions	⊖	⊖	○	⊕
Protocol Overhead	⊕ / ⊕⊕	○	⊖	⊖
Traffic Adaptivity	⊕	○	⊕	○
Node Adding	⊕⊕	⊕	⊖	⊖
Energy-Efficiency	⊖⊖	⊖	○	⊕

Table 2.2: Comparison of MAC protocols for wireless sensor networks

Static schedules promise higher energy-efficiency, since overhearing and idle listening can be completely prevented. However, they pose the restriction of static routing trees as constructed at the beginning of each collection phase in a two-phase data-gathering strategy.

2.3.2 TDMA Slot Assignment for Data-Gathering

TDMA scheduling, i.e., slot assignment, in data-gathering applications widely varies, as can be obtained from the examples in Section 2.3.1. In order to compare the performance of different approaches in a data-gathering context, a general classification is desirable. Only static and semi-static slot assignments will be considered, because the traffic pattern is a priori known in a two-phase data-gathering application with periodically rebuilt trees. Thus, efficient slot assignment does not require dynamic components. Note that a scheme with static slot assignment but dynamic access is regarded as dynamic (cf. [RXMC05]).

Inspired by [TW07c], it is useful to categorize slot assignments by the size of the overall set of different slots $\mathcal{S}$ required. Here, $R = |\mathcal{S}|$ is called the *round length*. The categories are as follow: A Type I slot assignment seeks to minimize R and thus has less slots than there are links in the routing tree. A Type II slot assignment has exactly as many slots as there are links in the tree; and finally, a Type III assignment has more slots than there are links in the tree. Using the size of the routing tree $\mathcal{T}$ as a reference has two reasons: Firstly, communication in the routing tree is done only between parents and their children. Hence, communication is directed and it is natural to speak of link assignments (cf. [Ram97]). Secondly, it is more convenient and less confusing, because the sink does not send any data and therefore needs no dedicated slot. Particularly, note here that $|\mathcal{T}| = N - 1$.

	Type I		Type II		Type III	
	k-hop	interference	standard	enhanced	plain	load aware
Assignment Strategy	static	static	static	semi-static	static	static
Assignment Overhead	⊖	⊖	⊕ / ⊕⊕	⊕ / ⊕⊕	⊕	⊕
Slot Storage	⊕⊕	⊕⊕	⊕⊕	⊖	⊕⊕	⊕⊕
Reusing Slots	✓	✓	–	(✓)	–	–
Collisions Possible	✓	–	–	–	–	–
Load Aware	–	–	–	(✓)	–	✓

Table 2.3: Comparison of static slot assignment schemes

In the remainder of this section, the named three categories of slot assignments will be examined in detail. Table 2.3 summarizes their main characteristics.

Type I Minimizing R aims at keeping latency low and throughput high, whereas individual node loads are not considered. Since every node or link is assigned one slot, storage is achieved with a small memory footprint. However, minimizing R is $\mathcal{NP}$-complete [EV05], so that different heuristics have been developed to approximate the optimum Type I slot assignment. They can be divided into two classes. Firstly, k-hop neighborhood information can be used to uniquely assign slots within k hops. This is equivalent to the corresponding coloring-problem of the graph G. Secondly, an interference calculation algorithm that, e.g., takes actual receiving powers of signals from remote nodes into consideration could be used. Note that using the k-hop neighborhood information can be thought of as a simplification of its interference-considering counterpart.

Ramanathan presents a unified framework for centralized slot assignment, which particularly includes link coloring [Ram97]. Three different algorithms are presented, where the most relevant is RAND, because a distributed version, called DRAND, is available [RWMX06]. DRAND is based on a randomized approach in order to pick the next link to color. Although it is a node coloring algorithm, link coloring could be also achieved. Time complexity and message overhead of DRAND are in the magnitude of the largest 2-hop neighborhood of a network.

Bryan et. al. propose the Color Constraint Heuristic (CCH) for slot assignment [BRD+07]. Unlike DRAND, they use a metric to determine the next node to color. Each node calculates the weighted sum of already colored nodes in its 2-hop neighborhood. The node having the highest CCH value is considered the most constrained node and thus proceeds in picking a new color. The distributed version of CCH (DSA-CCH) does not perform global node ordering for assigning colors, because

this would lead to exhaustive message exchange and energy consumption. Instead, nodes calculate a coloring score and compare it with a threshold; from this a node figures how many neighbors must be colored before it may pick a color itself. Simulation results reveal that DSA-CCH consumes less slots and energy than DRAND, but needs more time for the slot assignment to finish. Yet, DSA-CCH has been shown to fail in some combinations of thresholds and topologies. A variant of DSA-CCH, DSA-AGGR, uses the routing tree to assign colors so that parents receive higher colors than their children. This extension is meant to minimize latency [CMGL05].

An example of a Type I slot assignment using CCH is presented in Figure 2.4. Here, the choice of $k = 3$ leads to a lower probability of collisions [Grö04]. Slot 0 is assigned to links (v_0, v_1) and (v_9, v_{10}). It could not be assigned to (v_5, v_{10}), since v_{10} is a 3-hop neighbor of v_1. Note that the link (v_0, v_1) is the only connection between the sink and the rest of the network. Hence, throughput at the sink is $\sigma = \frac{1}{9}$.

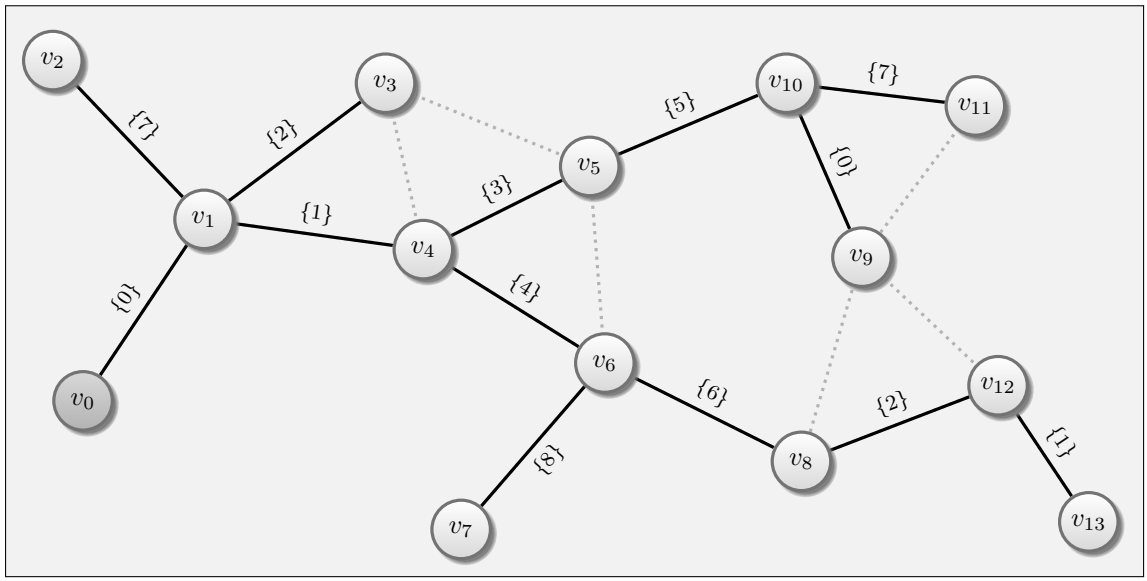


Figure 2.4: Type I slot assignment, $N = 14$, $R = 9$

Further strategies on finding minimum R exist, e.g. [GDP05, BGLA01], but all have in common that they do not produce collision-free schedules [Grö04]. Grönkvist shows by simulation results that there is no k for which collisions can be completely avoided. A remedy is to employ slot assignments taking the SINR (cf. Section 2.1.3) into consideration. Two algorithms developed to achieve this goal are STDMA [DG07] and the radio interference detection (RID) protocol [ZHSA05]. Although a lightweight derivative of RID exists, those protocols have considerable drawbacks. They are more complicated concerning implementation and work correctly, only if nodes are able

to accurately measure signal strengths produced by all remote nodes. Variations in signal strengths or imprecise measurements will consequently lead to collision-afflicted schedules. Besides, measured signal strengths have to be exchanged

Type II The main driver in assigning one fixed slot to each link is to minimize assignment overhead [TW07a]. In the easiest case, every node uses its unique identifier for slot assignment. Here, no overhead is produced. In a more advanced scheme with little overhead, a depth-first search on $\mathcal{T}$ can be performed to assign slots in ascending or descending order from leafs to the sink. The former reduces latency, the latter buffer usage. None addresses individual node loads. Using this method keeps assignment overhead low. However, Type II exhibits a severe disadvantage in large or sparse networks. In this situation, throughput is dramatically reduced [DML03] due to the high number of slots and the sink's low number of children. The immediate children of the sink have to forward all data inside their subtrees, but only have one slot, while R grows with the number of nodes N. Hence, throughput decays with $\frac{1}{R}$.

An enhanced version of a Type II slot assignment is described in [TW07b]. As before, there is one distinct slot for each link in the routing tree; but slots may be reused by other nodes on the same path from the original slot owner to the sink, when the former finishes sending. This increases throughput, but imposes a higher energy consumption, because information about reusable slots has to be sent via radio. Separate packets and piggybacking on data packets are possible methods. Two discussed strategies are as follows. Firstly, a node could keep every second slot it received from its children and forward all other slots into the direction of the sink. Secondly, a node v_i at depth h_i could keep every h_i'th slot. In both cases, children of the sink keep every received slot. The second solution has the advantage of slots being more frequently reused close to the sink, aiming at the reduction of latency and increase of throughput.

Figure 2.5 illustrates an example Type II slot assignment with ascendingly ordered slots. During depth-first search, v_{13} is the first visited leaf and link (v_{12}, v_{13}) is thus assigned slot 0. Slots 1 and 2 go to the other links on the same path until branching at v_6. Slot 3 is assigned to the link of the next leaf v_7. Compared to the earlier Type I assignment, the bottleneck at link (v_0, v_1) has become more severe: Throughput at the sink is reduced to $\sigma = \frac{1}{13}$. Enhanced Type II considerably attenuates this problem, as slots could be reused preferably on the link(s) to the sink; finally resulting in $\sigma = 1$.

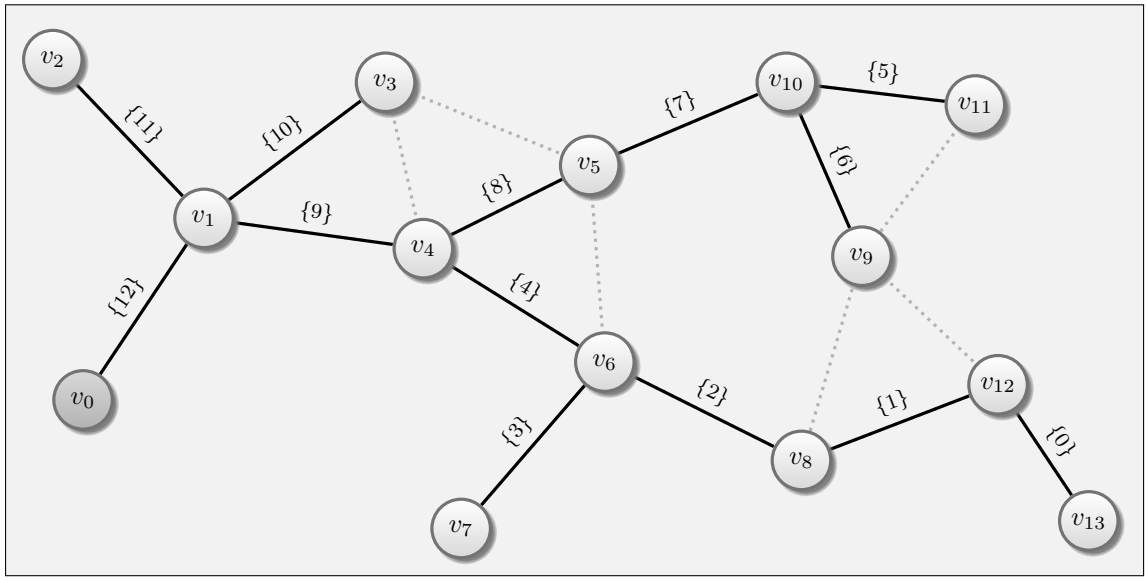
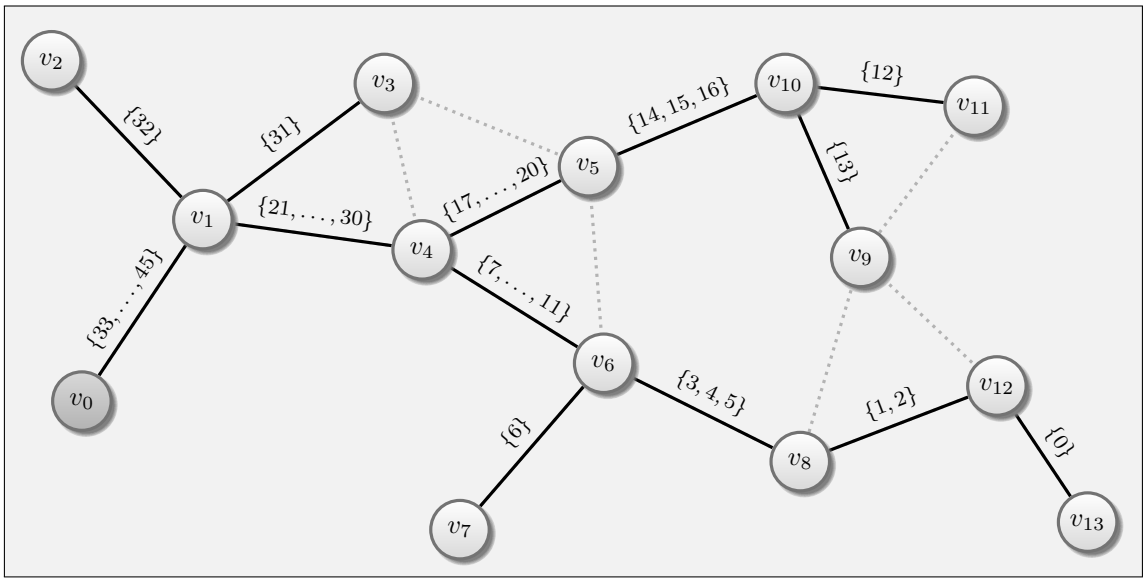


Figure 2.5: Type II slot assignment, $N = 14$, $R = 13$

Type III The design goal of a plain Type III slot schedule is to assign slots with respect to node loads, i.e., each node is assigned exactly as many slots for sending as it has for receiving plus one slot for its own data [TW07c]. In particular, a leaf node or its link to the parent, respectively, is assigned one slot. Any other node v_i is assigned one slot for each node within its subtree. A recursive expression of this is given by $|\mathcal{S}_i| = 1 + \sum_{v_j \in \mathcal{C}_i} |\mathcal{S}_j|$. To prevent interference, all sets have to be pairwise disjoint. As a result, each node v_i in the network will contribute h_i slots, so that the number of slots totally consumed will quickly increase with growing N. However, by assigning sets of slots in consecutive sequences during a depth-first search, each node solely has to save its first slot and the size of the set. As a result, constant storage space is achieved, but slot assignment produces some overhead.

An advantage of the plain Type III scheme is that during each round, the throughput at the sink is $\frac{N-1}{R}$, which does not depend on the sink's number of children. This is because each child of the sink has exactly as many sending slots as there are nodes within its subtree (including itself). Again, slots may be assigned in ascending or descending order from leafs to the sink. Note that in an descending order, the children of the sink will be the first sending nodes in each round. Hence, they must have at least as many packets as they have slots, or slots will be wasted, i.e., they cannot be used. An ascending order, in contrast, requires large buffers, as each node collects data from its subtree, before it forwards data itself. Figure 2.6 depicts a plain Type III slot assignment with ascending slot order based on the same depth-first

search as in the Type II example in Figure 2.5. It produces considerably more slots, but is capable of alleviating the bottleneck at link (v_0, v_1). The throughput at the sink is raised to $\sigma = \frac{13}{46}$.



■ **Figure 2.6:** Type III slot assignment, $N = 14$, $R = 46$

Data-gathering is considered to be a two-phase system with periodical collection phases. If a node v_i senses and stores different amounts of data than other nodes, or if v_i experienced a link failure during the last collection phase, its initial buffer fill level L_i may differ from other nodes' fill levels. To account for this, the authors of [TW07c] propose a load-aware approach that makes use of L_i. By introducing the constant λ, the advanced Type III assigns $|\mathcal{S}_i| = \lceil \frac{L_i}{\lambda} \rceil + \sum_{v_j \in \mathcal{C}_i} |\mathcal{S}_j|$ slots to a node v_i. The authors show that this approach can effectively reduce runtime in case of variable loads.

Chapter 3

Efficient TDMA Schedules for Data-Gathering

In this chapter, the objectives of this thesis will be outlined. Furthermore, an analytical view on the TDMA schedules presented in Chapter 2 is provided. A new scheduling scheme is introduced and compared with those existing solutions. It follows a discussion of metrics and parameters for evaluating the suitability of different TDMA schedules in a data-gathering application.

3.1 Objectives

Data-gathering is a task frequently performed by wireless sensor networks. As carried out in Chapter 2, energy-efficiency is of vital importance to prolong the lifetime of a network. Closely related to this is the choice of the data-collection strategy. If the gathered data is delay-tolerant, a two-phase scheme as presented in Section 2.2.1 is favorable. Between two sensing phases, a collection phase is employed: A tree is built and then used to forward all data to the sink.

This strategy particularly disencumbers from tree maintenance, and tree construction has been widely discussed. Here, breadth-first search has been found to produce good results (cf. Section 2.2.2) and can thus be used for tree construction without further examination. In contrast, data collection using different TDMA schedulings, as outlined in Section 2.3.2, should be closer investigated. In Section 2.3.1, fields of research on different MAC protocols have been presented. However, none of those works covers the comparison of different types of TDMA scheduling approaches in a data-gathering application. TDMA scheduling generally promises to prevent idle

listening, overhearing, and collisions. Yet, the described types of slot assignments exhibit striking differences. Hence, analyzing and comparing them in a two-phase data-gathering application is eligible. For this purpose, simulation is the method of choice to obtain results under realistic conditions. In this thesis, a suitable simulation framework is designed and implemented in order to finally conduct a detailed comparison and examination.

However, before this task can be performed, an analytical investigation of the different types of TDMA schedules is useful in order to identify parameters and metrics. For this reason, the three existing types of slot assignments will be discussed in the following. Inspired by the outcome of this, a new slot assignment will be presented. It is tailored to overcome the disclosed deficiencies of the existing approaches. Finally, parameters and metrics for comparing the described scheduling types will be pinpointed.

3.2 Analytical View on Existing TDMA Schedules

TDMA scheduling approaches can be divided into three categories, as explained in Section 2.3.2. Their distinct characteristics lead, in the first place, to a different number of slots produced. This affects the number of slots assigned to individual nodes and the overall round length. Both are relevant for the *runtime* T, which is defined as the total time required for collecting all data stored in the network. Additionally, slot assignment may have an impact on buffering issues and the consequences of packet loss. Hence, an analytical view on those topics promises a better understanding of the differences and their consequences. It will also help to identify necessary metrics and parameters.

3.2.1 Prerequisites

In order to estimate runtime T and the total number of slots R consumed by a slot assignment approach, a couple of notations are advantageous. They will be introduced and explained in the following.

At first, a tree can be characterized by its shape, particularly its depth h and its maximum degree C. In a *minimum-depth tree* with maximum degree C, each node has minimum possible depth $h^\star$. This implies that there are C^ℓ nodes in each level $\ell = 0, \ldots, h-1$ of the tree. Figure 3.1(a) shows an example minimum-depth tree with $N = 9$ and $C = 3$. Equation 3.2 yields $h^\star = 2$, which complies with the shown

tree. A minimum-depth tree is called a *complete tree*, if there is at most one node at level $h-1$ having neither 0 nor C children. The difference between a minimum-depth tree and the corresponding complete tree is depicted in Figure 3.1(b). If the number of nodes at level $\ell = h$ is C^ℓ, that tree is called a *full tree*. Figure 3.1(c) displays a full tree with same depth and C as before. The additionally needed nodes to fill the tree are light-colored.

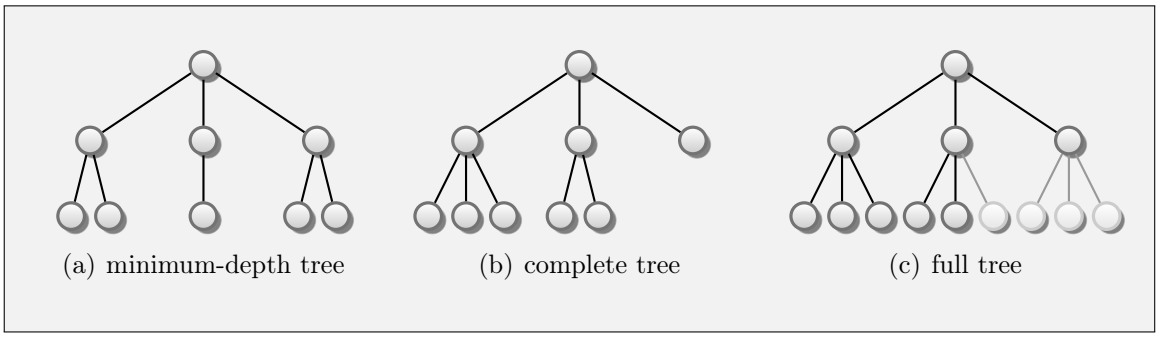


Figure 3.1: Different tree shapes with minimum depth

The minimum depth $h^\star$ of a tree having N nodes is that of the corresponding minimum-depth tree. Making use of the fact that a minimum-depth tree has at most as many nodes as the corresponding full tree, gives

$$N \leqslant \sum_{\ell=0}^{h^\star} C^\ell = \frac{C^{h^\star+1}-1}{C-1} \tag{3.1}$$

Transformation yields

$$h^\star = \lceil \log_C \left(N\left(C-1\right)+1\right) \rceil - 1 \tag{3.2}$$

3.2.2 Number of Slots

The number of slots a node can use for sending and receiving has two impacts: Firstly, storage of slots must be considered, and, secondly, their ratio influences buffer utilization and management. The round length determines the per-node throughput and thus particularly the throughput at the sink. Hence, overall runtime for data collection can be retrieved from the number of slots. As a result, looking at the number of slots has to be the first step.

Type I Minimum slot assignment, independent of whether using k-hop or interference information, is generally based on spatial slot reuse at some minimum distance. A simplified view is to require unique slot assignment inside a circle of radius ηR_{com}

with $\eta > 1$. The number of nodes inside this circle determines the minimum number of slots required in a Type I slot assignment. By means of the node density, the average number of nodes inside a communication circle is given as ϱ (cf. Section 2.1.2). It follows that the round length R_{I} of Type I is proportional to $\eta^2\varrho$. However, this estimation is not very precise, because the number of neighbors in a network may considerably vary. Hence, safe lower and upper bounds are given by

$$\varrho \leqslant \max_i |\mathcal{N}_i| + 1 \leqslant R_{\mathrm{I}} \leqslant N - 1 \tag{3.3}$$

Type II If every link in the tree $\mathcal{T}$ is assigned exactly one slot, the number of slots is always given by

$$R_{\mathrm{II}} = |\mathcal{T}| = N - 1 \tag{3.4}$$

Type III Given the plain slot assignment strategy as presented in Section 2.3.2 on page 24, the number of slots consumed is closely related to the tree structure. Every node produces one slot for each hop to the sink, so that we find

$$R_{\mathrm{III}} = \sum_{v_i \in \mathcal{V}} h_i \tag{3.5}$$

The sum takes its maximum value, if the tree is degenerate. It follows that

$$R_{\mathrm{III}} \leqslant \sum_{\ell=1}^{N-1} \ell \leqslant \frac{N^2 - N}{2} \tag{3.6}$$

The minimum is achieved in a minimum-depth tree:

$$R_{\mathrm{III}} \geqslant \sum_{\ell=1}^{h^\star-1} \ell\, C^\ell + h^\star \left(N - \sum_{\ell=0}^{h^\star-1} C^\ell \right) \geqslant N \left(h^\star - \tfrac{C}{C-1} \right) \tag{3.7}$$

This theoretical minimum can usually be achieved for small N only, as creating a minimum-depth tree is not possible, if ϱ does not increase with distance to the sink (cf. Section 2.2.2).

For the load-aware variant of this slot assignment, Equation 3.5 has to be multiplied with $\left\lceil \frac{L}{\lambda} \right\rceil$ in the case of the same initial buffer fill level L for each node. If initial buffer fill levels vary among nodes, there is no easy way to give a good estimation without an accurate knowledge about the L_i of each v_i. However, a rough upper bound can be derived by multiplying Equation 3.6 with $\max_i \left\lceil \frac{L_i}{\lambda} \right\rceil$ (cf. Section 2.3.2 on page 24). This is true, because each node accounts for $\left\lceil \frac{L_i}{\lambda} \right\rceil \cdot h_i$ slots. Taking the maximum and

inserting it into Equation 3.6 gives the stated upper bound. Accordingly, a lower bound is obtained by multiplying Equation 3.7 with the minimum $\min_{i \neq 0} \left\lceil \frac{L_i}{\lambda} \right\rceil$.

3.2.3 Memory Usage

Each node in the network must store its own slots and the ones used by its children to prevent idle listening (cf. Section 2.2.1). By combining the findings of Sections 3.2.2 and 2.3.2, a brief discussion of required memory for slot storage can be conducted.

Type I Since every node (except the sink) has exactly one slot for sending and at most C for receiving, it must store at most $C + 1$ slots. Thus, memory usage is linear in terms of C.

Type II As long as slots are not reused, it suffices to store one slot for sending and up to C ones for a node's children. Yet, as soon as slots are reused by other nodes, a node v_i may have to store up to $|\mathcal{T}_i|$ slots. In general, sets of slots will be fragmented, so that there is no easy way to store them. As no pattern inherent in those sets can be foreseen, the benefit from compression will be small. Besides storing, this problem also affects the forwarding of slots in the routing tree. In order to keep packet size low, it may only be possible to forward small numbers of slots at a time.

Type III Although a Type III slot assignment strategy produces considerably more slots than the other types, slot storage is linear in terms of C. Each node must store one set of slots for sending and up to C sets for receiving. Since those sets consist of consecutive slots, it suffices to store the first slot and the size of each set. Hence, storage consumes at most twice the amount of memory as compared to Type I.

3.2.4 Runtime Analysis

The runtime T of a data-collection phase is a crucial characteristic, because it determines the maximum sampling or sensing frequency of a node and thus the network. Knowing an estimation of the minimum possible runtime allows for a first performance evaluation of the different types of slot assignments. Furthermore, it is useful when analyzing results of a real deployment or a simulation. In the following, minimum runtime of the three different scheduling types will be roughly estimated. For this, a tree of depth $h \geqslant 2$ will be assumed, as slot assignments will not differ otherwise. Packet loss, buffer overflows and underruns will not be considered during calculation,

but their possible influence will be discussed separately in the following section. Runtime T will be expressed as the number of slots required for data collection, as it is assumed that slots have equal length and allow the transmission of exactly one data packet.

Type I One slot is assigned to each link of the tree, and slots are spatially reused. All data collected by the sink has to pass one of its children. Hence, overall runtime depends on the largest subtree, because the corresponding child of the sink will need the most time to forward all data. However, assuming a tree in which the C_0 children of the sink have equally sized subtrees, runtime takes its minimum:

$$T_{\mathrm{I}} \geqslant \frac{R_{\mathrm{I}}\,(N-1)\,L}{C_0} > \frac{\varrho\,(N-1)\,L}{C} \tag{3.8}$$

This equation points out that the number of immediate children of the sink influences runtime considerably. In contrast to the influence of the sizes of those subtrees, their individual shape, e.g., depth and balance, does not affect overall runtime.

Type II The runtime of a standard Type II slot assignment is above that of a Type I assignment, because slots are not reused. Thus, it is only of interest in very small or dense networks, where Type I produces approximately the same amount of slots as Type II. According to these facts, discussion is restricted to the enhanced version.

Estimation of the minimum runtime of an enhanced Type II slot assignment can be done by simply counting the number of required transmissions. This is equivalent to summing up the product of each node's initial buffer fill level and its depth in a minimum-depth tree. The fundamental idea behind this procedure is to assume that slots can be immediately reused on a new link (on the same path), if no more packets have to be sent via the old link. In this case, slot utilization reaches its optimum, and, hence, minimum runtime is achieved. By assuming a common initial buffer fill level L, estimation becomes basically the same as in Equation 3.7.

$$T_{\mathrm{II}} \geqslant \sum_{v_i \in \mathcal{V}} L h_i \overset{(3.7)}{\geqslant} NL\left(h^\star - \tfrac{C}{C-1}\right) \tag{3.9}$$

The main difference to the runtime of Type I is that $h^\star$ has superseded ϱ, since slots are not spatially reused by Type II. Hence, the actual depth has a decisive impact on runtime. In case a minimum-depth tree cannot be assumed, it may considerably increase. Additionally note that explicit knowledge about the slot reuse strategy is

not required for the estimation. It is not obvious, where slots should be actually reused in order to achieve minimum runtime.

Type III In a plain Type III assignment, a link is assigned as many slots as there are nodes in the corresponding subtree. If slots are assigned ascendingly from leafs to the sink (indicated by ↑), one packet from each node is collected per round. Hence, L rounds are required for collecting all data. Again, assuming a minimum-depth tree leads to the lowest possible runtime, since it generates the minimum amount of slots.

$$T_{\mathrm{III}\uparrow} \geqslant R_{\mathrm{III}} L \overset{(3.7)}{\geqslant} NL\left(h^\star - \tfrac{C}{C-1}\right) \tag{3.10}$$

Note that this equation gives the same result as Equation 3.9. This is not a surprise, since both calculate the minimum number of slots required to collect all data without spatially reusing slots. Assuming that each slot can be actually used to send a packet, the results must be the same.

If slots are assigned in descending order (indicated by ↓), runtime increases. L rounds are required for the last packet of a leaf being sent to the parent. The latter will have its next sending slots in the following round. Therefore, another $h^\star - 2$ rounds plus a fraction of an additional round are required, until the last packet from a leaf arrives at the sink. At the end, no complete round is needed, since only the children of the sink have to send packets and their slots are the first in a round. In order to simplify estimation, this fraction is left out.

$$T_{\mathrm{III}\downarrow} \geqslant R_{\mathrm{III}}\left(L + h^\star - 2\right) \geqslant N\left(L + h^\star - 2\right)\left(h^\star - \tfrac{C}{C-1}\right) \tag{3.11}$$

Estimating runtime for load-aware Type III is not performed, because varying initial buffer fill levels make things complicated. However, if initial buffer fill levels do not vary among nodes, behavior is basically the same as if using the plain version: Round length and the number of slots at each node's disposal are basically increased by the factor $\frac{L}{\lambda}$. Hence, the number of rounds required decreases by approximately $\frac{\lambda}{L}$, so that both effects cancel out.

3.2.5 Buffering Issues

Buffering issues have not been addressed so far, although underruns and overflows may have a severe impact on actual runtime and energy-efficiency. A closer look at this will be presented in the following.

Type I Non-leaf nodes have one slot for sending and at least one slot for receiving. In the absence of packet loss, buffer underrun cannot occur. Hence, a node can always use its sending slot, as long as there is data left in its subtree. In contrast, buffer overflow is likely, depending on the load of a node. During each round, a node may have to buffer up to $C - 1$ new packets. If a node experiences a buffer overflow, this implies that it will be restricted to accept no more than one packet total from all of its children in future rounds. As a result, flow control is required to avoid futile transmissions and thus wastage of energy (cf. Section 2.2.3). However, flow control must prevent buffer underruns in order to preserve low runtime.

Type II As long as slots are not reused, buffer utilization is according to Type I. Reusing slots may completely change the picture. Depending on the shape of the tree and the reuse strategy, buffer underrun and overflow may occur. Given a network with a tree as depicted in Figure 2.5 on page 24, v_1 may reuse slots 10 and 11 for sending, as soon as v_2 and v_3 have forwarded all their data. While v_1 has three slots for sending, it may have just one slot for receiving (from v_4). This may consequently lead to a buffer underrun at v_1 and increased runtime, since two slots per round cannot be used. On the other hand, buffer overflow would be possible, if v_4 reused all slots of its subtree to forward data to v_1.

Type III Nodes use consecutive slot sets for sending and receiving packets. If slots are in ascending order from leafs to the sink, a node will receive packets from its children before it can forward data. In each round, and particularly in the first, of a data-collection phase, the buffer of a node must be large enough to store all incoming packets in addition to the already present ones. If this requirement is violated, the corresponding node v_i is forced to drop received packets. This implies that its children have to resend the corresponding data, and flow control is required. In this situation, subtrees of v_i may forward their data at different paces. If all packets stored in one subtree have been forwarded, while a different one has not finished sending its packets, all slots used in the former cannot be used any longer. In addition, v_i still has sending slots for that already finished subtree. At some time, when the number of incoming packets and those already stored in its buffer falls below the number of sending slots available, v_i can only use a fraction of its sending slots. This leads to a chain reaction, because its parent will receive less packets from v_i than there are slots assigned to that link. Therefore, that parent has less packets to send, and this problem propagates to

the sink. Besides the severe impact on runtime, idle listening will occur, if parents are not informed by their children about unused slots.

By ordering slots descendingly, buffer utilization is alleviated. During the first round of a data collection phase, each node forwards more packets than can be received during the same round. Hence, it is sufficient if all nodes may buffer as many packets as they can receive per round. However, if a node has a lower initial buffer fill level than sending slots, some of them will be wasted during the first round. This will cause a chain reaction as discussed above.

3.2.6 Packet Loss and Link Failure

Due to the lossy nature of wireless sensor networks, packet loss and link failures must be considered. However, there is no easy way to model those issues analytically and quantify their impact, but a theoretical discussion can be used in order to identify possible sensitivities.

Type I The short round length keeps delay comparably low in the case of packet loss; this is true, even if only one link suffers from losses. If all links in the network are equally exposed to losses, runtime will presumably scale with the inverse packet reception rate, and buffering will not be considerably affected. Short disturbances on a link may be implicitly compensated by the time between two sending slots of the same node. Link failures will not severely compromise the overall runtime, because there is just one sending slot dedicated to each link and slots are spatially reused. Thus, even if a large subtree becomes disconnected from the tree, it can be expected that only few slots will be wasted.

Type II Due to the increased round length as compared to Type I, packet loss has a higher impact on runtime. Depending on the slot reuse strategy, packet loss and particularly link failures may lead to a higher probability of buffer underruns. This situation occurs, if slots are reused in a way that provides a node with more sending slots than its children have in sum. Short link disturbances can be overcome by the fair round length. Permanent link failures imply the wastage of slots, because slots are unique within a subtree. Here, graveness depends on the size of the disconnected subtree.

Type III The huge round length makes Type III very sensitive to packet loss and link failures. First of all, one lost packet at a leaf node may prolong runtime by a

complete round. In addition, a lost packet will render all remaining slots dedicated to the same path from the initial sender bootless. This is true, because a node v_i at depth h_i accounts for h_i slots, precisely one slot for each link on the path to the sink. Hence, packet loss and link failure will eventually result in a massive wastage of slots, even if only a single leaf is affected. Besides, temporary losses on one link may affect many slots assigned to the same link, as those slots are consecutive. As a result, a node may have to wait for the next round in order to make up for those unused slots, leading to a long delay.

3.2.7 Summary and Comparison

The previous analysis, in combination with the descriptions in Section 2.3.2, shows that all scheduling types have their strengths and weaknesses.

Type I leads to a small number of slots that only depends on the network density. However, slot assignment requires communication overhead and collisions are possible. Runtime is affected by both the number of nodes and network density. If the latter stays constant, runtime increases linearly with network size, which enables Type I to perform well regardless of it. However, the number of children of the sink and the sizes of their subtrees influence runtime. Hence, Type I may perform poorly, if the sink has few children, or if there is a particularly large subtree. Depending on node loads and buffer sizes, Type I may suffer from buffer overflows, which require additional overhead in order to avoid futile (re)transmissions. Packet loss and interrupted links do not tend to severely influence runtime.

Enhanced Type II offers easy slot assignment, but imposes communication overhead needed for reusing slots. In addition, slot storage requires much space, so that Type II cannot be used in large networks. As the round length exceeds that of Type I, packet loss has a higher impact on runtime. Since the size of each subtree corresponds to the number of slots used within, reusing slots leads to good runtime expectations. In particular, this aspect may alleviate the influence of unbalanced trees. However, runtime is directly affected by the depth of the tree. Type II is prone to both buffer overflows and underruns, which may lead to communication overhead and higher runtime. Due to the complicated runtime behavior concerning slot reuse, no precise forecast on the influence of load and buffer fill levels can be given. Furthermore, it is not clear how to set up an optimal reuse strategy.

Round length is critical in a Type III slot assignment. With increasing network size, its margin grows extensively, as it is affected by the shape of the tree. The conse-

quences of the round length are manifold. Firstly, additional rounds may be required in the case of packet loss. Secondly, interrupted links and disconnected subtrees will lead to many unused slots. Both issues increase runtime, possibly beyond an acceptable limit. A high round length implies that each node has many slots at its disposal for sending and receiving. Their number grows rapidly with network size and tree depth. In conclusion, buffer sizes must likewise increase, or else some slots cannot be used. If slots are assigned in ascending order, this problem is particularly grave, as nodes receive foreign packets before they send packets for the first time in a collection phase. Ordering slots descendingly inhibits this problem, but, in contrast, leads to a higher minimum runtime. Even if all those issues can be neglected, estimations have shown that there exists a critical network size, from which on Type I will lead to a lower runtime.

3.3 A New, Light-Weight TDMA Schedule for Data-Gathering

Due to the results of Section 3.2, we developed a new, light-weight TDMA schedule. In general, it is a variant of Type III with a reduced number of slots and spatial reuse. It equips each node with an amount of slots according to an estimation of its expected load by means of the number of leafs in its subtree. In contrast to Type III, sets of slots alloted to the same link are not consecutive. Thus, buffer usage and the likelihood of overflows is minimized. Still, slot assignment can be achieved with low overhead and a small memory footprint, comparable to Type III.

In the remainder of this section, a detailed view on this new assignment strategy is provided. It is enriched by an analytical study for comparison with the results from Section 3.2.

3.3.1 Spatial Path-Based Reuse Slot Assignment

The idea behind the spatial path-based reuse (SPR) scheme is to regard the routing tree as an overlay of the paths from each leaf to the sink. Each of those paths is handled separately during the slot assignment procedure, and the assigned sets of slots are pairwise disjoint. The final slot assignment consists of the union of those sets. Thus, if a node v_i is on P_i paths, it will get assigned P_i slots. Here, P_i additionally equals the number of leafs in the subtree of v_i. This approach restricts the spatial reuse of slots to nodes that are on a common path.

In the following the basic principle will be introduced; details of the final algorithm will be presented later. In its basic form, fixed-size sets of κ slots are alloted to each path from a leaf to the sink. Since those sets do not overlap, inter-path collisions are completely prevented. Slot assignment starts at the sink, and slots are reused after every κ hops on a path, so that the total number of slots is reduced. Yet, reusing slots may cause intra-path collisions, which can be avoided by choosing κ appropriately in context of the network density and depth of the routing tree.

If κ is equal to the length of the path, then trivially no intra-path collisions can occur. Thus, for the example depicted in Figure 3.2, κ need not be larger than 6. However, with v_0 being the sink, $\kappa = 3$ will most likely lead to collisions at v_0, as v_1 and v_7 would share the same slot and v_0 is within the interference radius of v_7. In this situation, the path bends around a void. Now imagine another node placed in a way, that it was inside the communication range of its parent v_7, but slightly outside the communication range of v_0. In this situation, $\kappa = 4$ would still lead to collisions. This example illustrates that voids may be of almost arbitrary size. Albeit this phenomenon cannot be completely prevented, simulation has shown that it is likely in sparse topologies only.

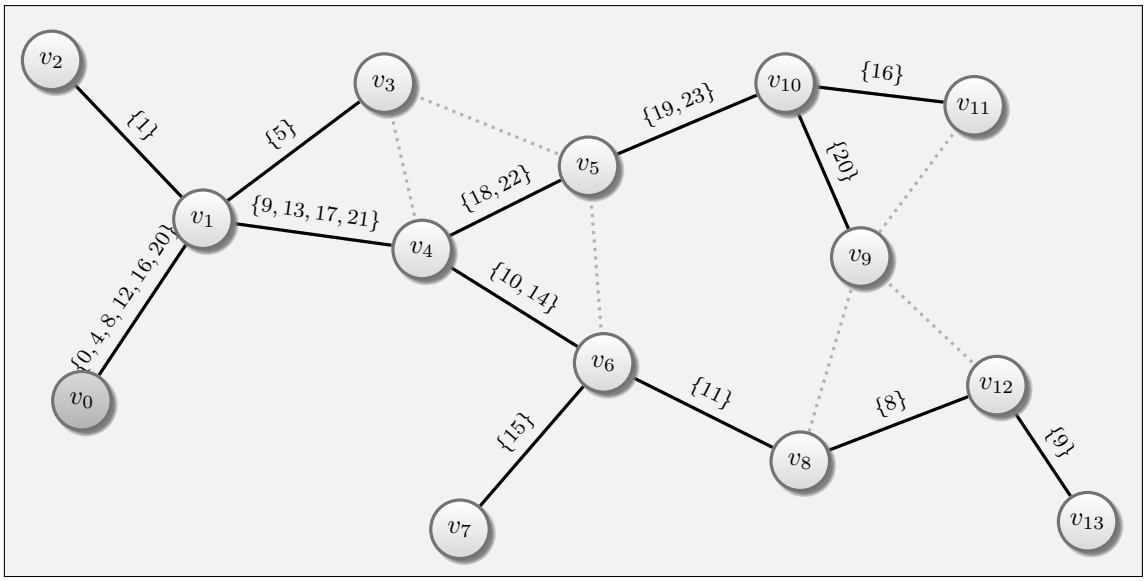
SPR can be efficiently implemented using the information gathered during tree construction. With low effort, this method equips every non-leaf node in the network with exactly as many sending slots as it has receiving slots. This has an outstanding advantage: Buffer usage is considerably reduced, as between every pair of sending slots there is just a single receiving slot. In the absence of packet loss, this implies that buffer usage will never exceed the initial fill level by more than one. Even if there is packet loss, buffer fill levels will not increase rapidly, as the balance between actually incoming and outgoing packets can be expected to be affected slightly only.

Although a node gets assigned as many slots as there are leafs among its children, there is no need to explicitly store them. The following slot assignment procedure guarantees, that each node v_i may use P_i slots with distance κ beginning with slot s_i^0.

- Let $\{p_0, \ldots, p_{P-1}\}$ be the set of all paths from the leafs to the sink, ordered according to a depth-first search.
- Assign the slots $m\kappa, m\kappa + 1, \ldots, m\kappa + \kappa - 1, m\kappa, m\kappa + 1, \ldots$ to the links on path p_m, starting at the sink.

Thus, a node v_i can calculate the sequence of its slots from the smallest slot s_i^0, P_i, and κ. However, in order to identify interrupted links as described in Section 2.2.1, v_i has to map its receiving slots to its children. This can be achieved by v_i storing

the number of paths P_j its child v_j is on; ordered according to the p_m. Finally, v_i can calculate the slots of its children. Because P_i is the sum of the P_j, it does not have to be stored explicitly.



■ **Figure 3.2:** Basic SPR slot assignment with $\kappa = 4$, $N = 14$, $R = 24$

This distribution scheme exhibits a disadvantage. When there are paths having less than $\kappa + 1$ nodes, some slots are not used; e.g., the path from the sink v_0 to v_2 in Figure 3.2 has length 2, while $\kappa = 4$. Although this will not lead to an increase in energy consumption, it will prolong the total runtime.

3.3.2 Effective Implementation

A simple remedy of the shortcoming identified in the previous section would be the following. If a path p_m has less than $\kappa + 1$ nodes, unused slots can be used for the next path p_{m+1}. Consequently, the sets of slots assigned to a path no longer have the form $\{m\kappa, m\kappa + 1, \ldots, m\kappa + \kappa - 1\}$, and the slots of a single node no longer have the constant displacement κ.

In order to assign precisely k slots to paths with length $k < \kappa$, every node v_i calculates its displacement vector $\mathbf{d}_i[1], \ldots, \mathbf{d}_i[\kappa]$ as follows. $\mathbf{d}_i[k]$ is the number of leafs in its subtree with depth $k < \kappa$ in $\mathcal{T}$. For $k = \kappa$ the vector denotes the number of leafs in the subtree with depth κ or higher in $\mathcal{T}$.

$$\mathbf{d}_i[k] = \begin{cases} |\{v_j \in \mathcal{F}_i : h_j = k\}| & \text{if } 1 \leqslant k < \kappa \\ |\{v_j \in \mathcal{F}_i : h_j \geqslant k\}| & \text{if } k = \kappa \end{cases} \tag{3.12}$$

Each node v_i needs to store the displacement vectors $\mathbf{d}_j$ of its children $v_j \in \mathcal{C}_i$ along with a slot offset for each entry in $\mathbf{d}_j$. Those offsets are required, because paths are firstly ordered by length and secondly according to a depth-first equivalent as before. As shown in the following, this information together with the length of the round and the depth of the nodes will be sufficient to compute the slots assigned to a node. The space required for this information solely depends on κ and the number of children a node has.

Since SPR relies on the routing tree, a three-step procedure can be used to assign slots. Firstly, the sink starts generating the tree $\mathcal{T}$ via a breadth-first search. Secondly, the leafs of the just constructed tree $\mathcal{T}$ reflect the construction wave up to the sink. During this contraction phase, every non-leaf node v_i in the network stores the displacement vector $\mathbf{d}_j$ for each of its C_i children. Leaf nodes have no children and their own $\mathbf{d}_i$ is uniquely determined by their own depth in $\mathcal{T}$. Finally, the sink acquires the displacement vectors of its children and thus its own. As a result, it uses its own displacement vector $\mathbf{d}_0$ to calculate $R = \sum_{k=1}^{\kappa} k\,\mathbf{d}_0[k]$. Thirdly, the slots are actually assigned. Given its displacement vector $\mathbf{d}_0$, the sink calculates an offset vector $\mathbf{o}_0$, where $\mathbf{o}_0[k]$ represents the smallest of all slots belonging to a path with displacement k.

$$\begin{aligned} \mathbf{o}_0[1] &= 0 \\ \mathbf{o}_0[k+1] &= \mathbf{o}_0[k] + k\,\mathbf{d}_0[k] \quad (1 \leqslant k \leqslant \kappa - 1) \end{aligned} \tag{3.13}$$

Next, the sink calculates the offset vectors $\mathbf{o}_j$ using the $\mathbf{d}_j$ and $\mathbf{o}_i = \mathbf{o}_0$:

$$\mathbf{o}_j[k] = \mathbf{o}_i[k] + k \sum_{\substack{v_{\tilde{j}} \in \mathcal{C}_i \\ v_{\tilde{j}} \prec v_j}} \mathbf{d}_{\tilde{j}}[k] \quad (1 \leqslant k \leqslant \kappa, v_j \in \mathcal{C}_i) \tag{3.14}$$

Here, the values $\mathbf{o}_j[k]$ represent the smallest of all slots belonging to a path with displacement k within the subtree of child v_j. The operator $\prec$ refers to the required ordering of children. Also note that the value of $\mathbf{o}_i[k]$ is not used during actual slot calculation, if $\mathbf{d}_i[k] = 0$.

The sink passes each $\mathbf{o}_j$ to the corresponding child. In addition, each child receives R. Every node v_i that receives an offset vector from its parent proceeds accordingly. First, it adopts the received offset vector as its own $\mathbf{o}_i$ and generates for each of its children a new offset vector $\mathbf{o}_j$ using the same method as the sink, i.e., applying Equation 3.14. Then, R and the calculated $\mathbf{o}_j$ are sent to the children v_j of v_i. It suffices that v_i additionally stores $\mathbf{o}_i$, because it can always reconstruct the $\mathbf{o}_j$

from $\mathbf{o}_i$ and the $\mathbf{d}_j$, when needed. Finally, v_i can calculate its own slots by means of the $\mathbf{o}_j$ and $\mathbf{d}_j$. The algorithm terminates, if all leafs have received their offset vector.

3.3.3 Explicit Slot Calculation

Each node v_i has explicit knowledge about its depth h_i in $\mathcal{T}$, its own offset vector $\mathbf{o}_i$, and its children's displacement vectors $\mathbf{d}_j$. The vectors $\mathbf{o}_j$ and $\mathbf{d}_i$ can be calculated from those. While $\mathbf{d}_i$ is directly available on leaf nodes, inner nodes obtain $\mathbf{d}_i$ by

$$\mathbf{d}_i[k] = \sum_{v_j \in \mathcal{C}_i} \mathbf{d}_j[k] \qquad (1 \leqslant k \leqslant \kappa) \tag{3.15}$$

Putting these pieces of information together, a node can determine its set of slots

$$\mathcal{S}_i = \left\{ s \mid 1 \leqslant k \leqslant \kappa,\ 0 \leqslant d < \mathbf{d}_i[k] : s = \mathbf{o}_i[k] + k\,d + (h_i - 1) \bmod k \right\} \tag{3.16}$$

and the ones of its children $v_j \in \mathcal{C}_i$

$$\mathcal{S}_j = \left\{ s \mid 1 \leqslant k \leqslant \kappa,\ 0 \leqslant d < \mathbf{d}_j[k] : s = \mathbf{o}_j[k] + k\,d + h_i \bmod k \right\} \tag{3.17}$$

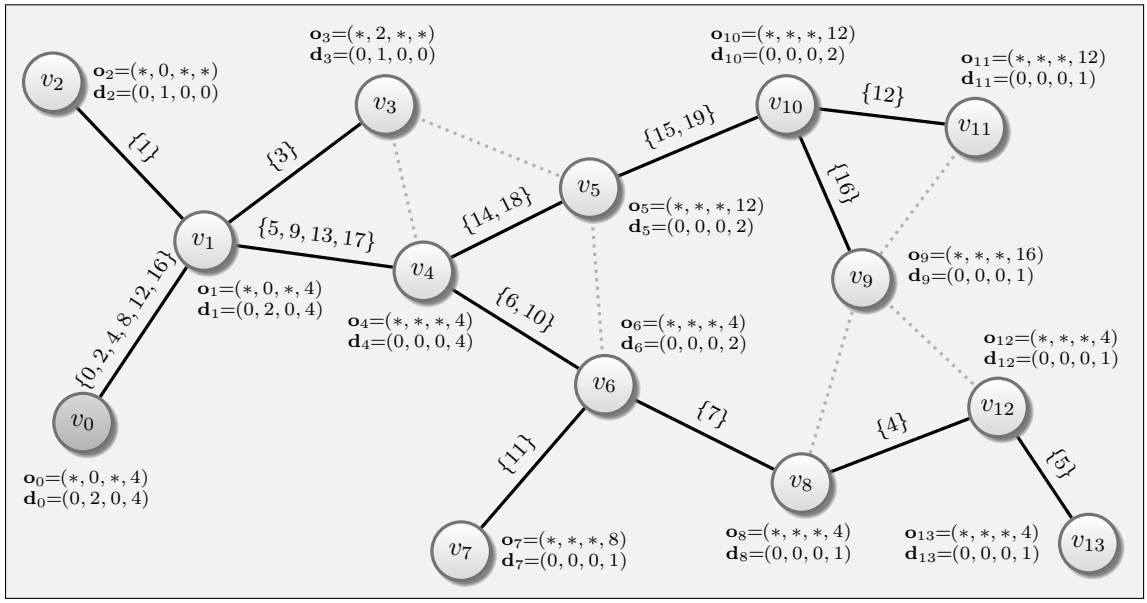
Note that the modulo operator is effective and thus required only for $k = \kappa$, because, by construction, paths with length $k < \kappa$ are assigned k slots. In addition, the sink must not compute $\mathcal{S}_0$, because there are no slots assigned to it.

3.3.4 Example

In order to illustrate the explicit slot calculation as outlayed in Section 3.3.3, Figure 3.3 shows an example slot assignment for $\kappa = 4$. Displacement and offset vectors have been calculated as described in Section 3.3.2, unused values of the latter are marked with asterisk for clarity.

If v_{10} wants to calculate the set of slots $\mathcal{S}_9 = \{s_9^0\}$ its child v_9 will use for sending, this can be done by using Equation 3.17. With $h_{10} = 4$, $\mathbf{o}_9 = (*, *, *, 16)$, and $\mathbf{d}_9 = (0, 0, 0, 1)$ it follows

$$s_9^0 = 16 + 4 \cdot 0 + 4 \bmod 4 = 16$$



■ **Figure 3.3:** Advanced SPR slot assignment with $\kappa = 4$, $N = 14$, $R = 20$

Likewise, v_{10} can calculate its set of sending slots $\mathcal{S}_{10} = \{s_{10}^0, s_{10}^1\}$ with Equation 3.16. From $h_{10} = 4$, $\mathbf{o}_{10} = (*, *, *, 12)$, and $\mathbf{d}_{10} = (0, 0, 0, 2)$ it follows

$$s_{10}^0 = 12 + 4 \cdot 0 + 3 \bmod 4 = 15$$
$$s_{10}^1 = 12 + 4 \cdot 1 + 3 \bmod 4 = 19$$

Note that in this example, the impact of the modulo operator has taken effect for slot reuse.

3.3.5 Analytical Evaluation and Comparison

In the following the same estimations as found in Section 3.2 will be derived for the SPR slot assignment. They will be used to compare SPR with the other types of slot assignments.

The number of slots consumed by SPR depends on the number of paths from the sink to the leafs in $\mathcal{T}$ and their lengths. It can be calculated by using the set of leafs:

$$R_{\text{SPR}} = \sum_{v_i \in \mathcal{F}} \min(h_i, \kappa) \tag{3.18}$$

Hence, for $N > \kappa$, κ is the minimum number of slots. An upper bound for the round length can be calculated by means of κ and a complete tree, as it has a maximum

number of leafs. The number of leafs in a complete tree is the sum of nodes at level $h^\star$ plus all nodes at level $h^\star - 1$ that have no children. Here, the number of nodes at level $h^\star$ is implicitly given by the difference between N and the number of nodes below level $h^\star$ (cf. Equation 3.1). It follows that

$$R_{\text{SPR}} \leqslant \kappa \left|\mathcal{F}\right| \leqslant \kappa \left(N - \tfrac{C^{h^\star}-1}{C-1} + C^{h^\star-1} - \left\lceil \tfrac{1}{C}\left(N - \tfrac{C^{h^\star}-1}{C-1}\right)\right\rceil\right) \leqslant \frac{C\kappa N}{C+1} \tag{3.19}$$

Runtime estimation for SPR can be generally based on that for Type II by neglecting the spatial reuse of slots to begin with, i.e., assuming $h = \kappa$. A closer analysis of the influence of reusing slots reveals that it leads to simultaneous transmissions of packets on the same path. Hence, counting the number of required transmissions must be done using the minimum of a node's depth in $\mathcal{T}$ and κ. This is true, because for any transmission at a depth greater than SPR, a slot is reused, and thus there already is another transmission using the same slot. As a result, that slot must only be counted once. Again, a common buffer fill level L is assumed, and using Equation 3.7 immediately yields

$$T_{\text{SPR}} \geqslant \sum_{v_i \in \mathcal{V}} L \min\left(h_i, \kappa\right) \overset{(3.7)}{\geqslant} NL\left(\min\left(h^\star, \kappa\right) - \tfrac{C}{C-1}\right) \tag{3.20}$$

Compared to Equation 3.9, $h^\star$ is replaced by κ, if $\kappa \leqslant h^\star$. Therefore, runtime of SPR becomes independent of the tree depth and thus grows linearly with the network size N only. This implies that SPR is capable of outperforming Types II and III particularly in large networks. In addition, a small κ leads to the conclusion that the runtime of SPR can stay below that of Type I.

Although Equation 3.20 discloses the minimum possible runtime, it leaves generally open, where leafs must be placed and how many leafs are needed to achieve minimum runtime. For $\kappa \leqslant h^\star$, a minimum-depth tree leads to lowest runtime. Yet, the number of leafs in a minimum-depth tree and their position may vary (cf. Figure 3.1), but the influence is not clear. In addition, $\kappa < h^\star$ leads to the same minimum runtime for a wide range of trees, because only κ hops of each node's depth are taken into consideration. Here, the influence of balance and depth of a tree are not obvious and require further investigation.

As explained in Section 3.3.1, SPR assigns slots in a way that between each two receiving slots, there is always one sending slot. Hence, buffer overflow is not likely even in the presence of packet loss. However, buffer underrun is possible. Referring to Figure 3.3, v_6 exposes a critical situation. If v_7 has finished sending all packets,

	Type I k-hop	Type II enhanced	Type III plain	SPR
Lower Bound for R	ϱ	$N-1$	$N\left(h^\star - \frac{C}{C-1}\right)$	κ
Upper Bound for R	$N-1$	$N-1$	$\frac{1}{2}\left(N^2-N\right)$	$\frac{C}{C+1}\kappa N$
Slot Storage	$\mathcal{O}(C)$	$\mathcal{O}(CN)$	$\mathcal{O}(C)$	$\mathcal{O}(C\kappa)$
Lower Bound for T	$\frac{\varrho}{C}(N-1)L$	$NL\left(h^\star - \frac{C}{C-1}\right)$	$NL\left(h^\star - \frac{C}{C-1}\right)$	$NL\left(\min(h^\star,\kappa) - \frac{C}{C-1}\right)$
Buffer Overflow	✓	✓	✓	–
Buffer Underrun	–	✓	✓	✓

Table 3.1: Analytical comparison of static slot assignment schemes

v_6 will be left with two sending slots but only one active receiving slot. Up to that point, v_6 has forwarded as many packets as it has received, so that its buffer is at its initial fill level. Hence, that slot can be effectively used. However, the balance between incoming and outgoing packets is disturbed. If the buffer of v_6 runs empty before the second subtree has finished sending, one slot per round will be wasted. Compared to Type III, propagation of this problem is slower and its impact less severe, if the slot is reused somewhere else. Due to the own initial fill level, each node will suffer from the disturbed balance with a certain delay. Nevertheless, in case of buffer underrun, an information mechanism is required to prevent parents from idle listening.

Table 3.1 summarizes the main characteristics of the four slot assignment strategies. As shown in Section 3.3.2, slot assignment using SPR can be done with low overhead. It is comparable to that of enhanced Type II and Type III. In addition, the explicit slot calculation described in Section 3.3.3 allows for a small memory footprint. However, SPR schedules may be collision-afflicted, so that the choice of κ is an critical issue. Comparing runtime estimations gives evidence, that SPR can achieve low runtime even in large or dense networks. However, performance depends on the number of leafs and the shape of the tree. Unbalanced trees may provoke buffer underruns, so that some slots will be wasted. The actual effect of buffer underrun cannot be grasped by analytical investigation. Therefore, it is impossible to decide at this point, if and under which conditions SPR may outperform other scheduling types.

3.4 Considerations for a Simulative Comparison

The analytical investigation performed in Sections 3.2 and 3.3.5 helps to generally understand the behavior, strengths and weaknesses of the different types of slot assignment. However, estimations are rough and based on assumptions, such as equal buffer fill levels, unlimited buffers, and the absence of packet loss. In a real sensor network, they can be hardly justified, as particularly discussed in Section 2.1.1. Some issues, such as energy-efficiency, cannot be forecasted, since they are, e.g., closely related to packet loss and unpredictable tree structure. Hence, simulation is the method of choice in order to gain more realistic performance results. Additionally, it is required to verify and weight the influence of the findings in this chapter. Yet, before developing a simulation framework as suggested in Section 3.1, metrics have to be assessed and parameters must be identified.

3.4.1 Metrics

In a data-gathering application, energy-efficiency is one of the most critical characteristics, since it determines the possible lifetime of a network. Although energy-efficiency is affected by the overall energy consumption, this would imply to consider tree construction, synchronization (for TDMA to work), slot assignment, and data collection. While the first two issues are common to all different slot assignments, they do not have to be considered when comparing slot assignments. The amount of energy required for assigning slots using Type I is beyond the amount needed by the other types. This is true, because Type I relies on precise knowledge of communication or interference neighborhoods in order to create schedules with few or no collisions (cf. Section 2.3.2). Yet, slots are assigned only once at the beginning of each collection phase. Assuming that traffic during actual data collection is in a higher magnitude than slot assignment overhead, the latter can be justifiably neglected.

Still, energy-efficiency can be looked at from two different perspectives. Firstly, the overall amount of energy consumed by all nodes in the network must be considered, because it focuses on global efficiency, i.e., the amount of energy required for receiving a portion of data. In contrast, individual energy consumption is relevant. Nodes close to the sink are exposed to the highest load in the network, so that they will be the first ones to run out of energy. Since they form the only connection points between all other nodes and the sink, data collection will become impossible as soon as these nodes have completely de-energized.

As discussed in Section 2.1.1, communication in wireless sensor networks is suspect to interference. Even in the absence of packet loss due to communication errors, reliability cannot be guaranteed. Since Type I and the new SPR slot assignment strategy may both suffer from collision-afflicted schedules, the *yield* (or success rate) of a collection phase must be analyzed. It is defined as the average ratio of data actually received by the sink as compared to the amount of data generated during a sensing phase. In addition, the number of collisions and *dead links* is of significance in this context. The latter can be described as a disturbed communication between parent and child, which causes both nodes to eventually give up communication, although the child has not completed forwarding all data stored in its subtree.

Besides yield, low runtime is eligible. In consequence to decreasing runtime, a higher sensing or sampling frequency can be achieved or the accuracy of measurements can be boosted. This may be particularly relevant in monitoring applications that profit from a higher resolution of observations. Additionally, runtime may even be limited; e.g., consider a tideland monitoring application, in which data collection is possible exclusively during ebb tide.

Finally, communication overhead during data collection must be looked at. Its sources are manifold. Enhanced Type II, e.g., directly produces overhead by sending slots upwards in the tree, so that some other node on the same path can reuse them. Collisions lead to overhead, because data packets and acknowledgments have to be sent multiple times. In addition, buffer management and flow control, as described in Section 2.2.3, impose additional traffic, even if their influence can be reduced by piggybacking corresponding information.

3.4.2 Parameters

A thoughtful identification of relevant network parameters is indispensable for obtaining a meaningful and broad comparison of the presented slot assignments. The analytical investigations in Sections 3.2 and 3.3.5 have already put forth a variety of parameters that have to be considered.

In the first place, the number of nodes and network density are key characteristic of a wireless sensor network. As shown in Section 3.2.2, they influence the overall number of slots assigned. Both additionally affect the shape of the built routing tree. The tree depth will increase with a growing number of nodes and decreasing density. Increasing the density will additionally result in a higher tree degree. In this context, the impact of limiting the maximum number of children per node should be studied.

It may affect tree depth and thus runtime, and may be necessary in dense networks due to the restricted amount of memory available on wireless sensor nodes.

All estimations in this chapter have assumed, for simplicity reasons, equally distributed initial buffer fill levels. In a real network, this may not be justifiable. Firstly, some nodes may be required to produce more data during a sensing phase than others. Secondly, if links have been interrupted in a prior collection phase, fluctuating buffer fill levels at the beginning of the following collection phase(s) will result. Hence, the influence of different buffer fill levels should be investigated.

In addition, particularly Type III is sensitive to the initial buffer fill levels, individual node load, and the maximum buffer size. This observation proposes two different approaches on this matter: varying initial fill levels and also comparing limited with unlimited buffers. Here, the combination of network size, density, and the resulting tree structure may lead to additional conclusions.

Chapter 4

Simulation Framework

The analytical investigation in Chapter 3 has lead to a better understanding of existing TDMA schedules and the newly introduced SPR. However, further investigations are required in order to validate those findings and render them more precisely. In addition, open issues have to be inspected, e.g., the influence of network density, packet loss, buffer size, and fill level. Such a task can be performed via simulation. The ns-2 network simulator is a well-established tool for evaluating and analyzing wireless sensor network applications. This chapter starts out with a detailed look at the relevant parts of ns-2 (version 2.31) for building a data-gathering simulation. An appropriate simulation framework is defined next, including the specification of a TDMA-based protocol for data-collection. Finally, the implementation of this protocol will be discussed.

4.1 Introduction to ns-2

This section provides a brief look at the ns-2 stack. As the characteristics of the MAC layer and its influence on interference have major impact on data-gathering applications, a detailed description of their realization in ns-2 will be provided.

4.1.1 Protocol Stack

ns-2 employs a strictly layered stack, in which communication is generally possible only between adjacent layers. They are implemented in C++ in order to achieve high performance. Connecting these layers and configuring a simulation run is done using an OTcl script for flexibility. Figure 4.1 illustrates a simplified diagram of those layers with their interaction. Solid arrows indicate packet flow, dashed ones refer to access

during packet processing. In the following, the general packet flow will be explained. Here, the terms incoming and outgoing will be used to distinguish between packets received and sent by a node, respectively. Details about the physical and MAC layer are found in the following sections.

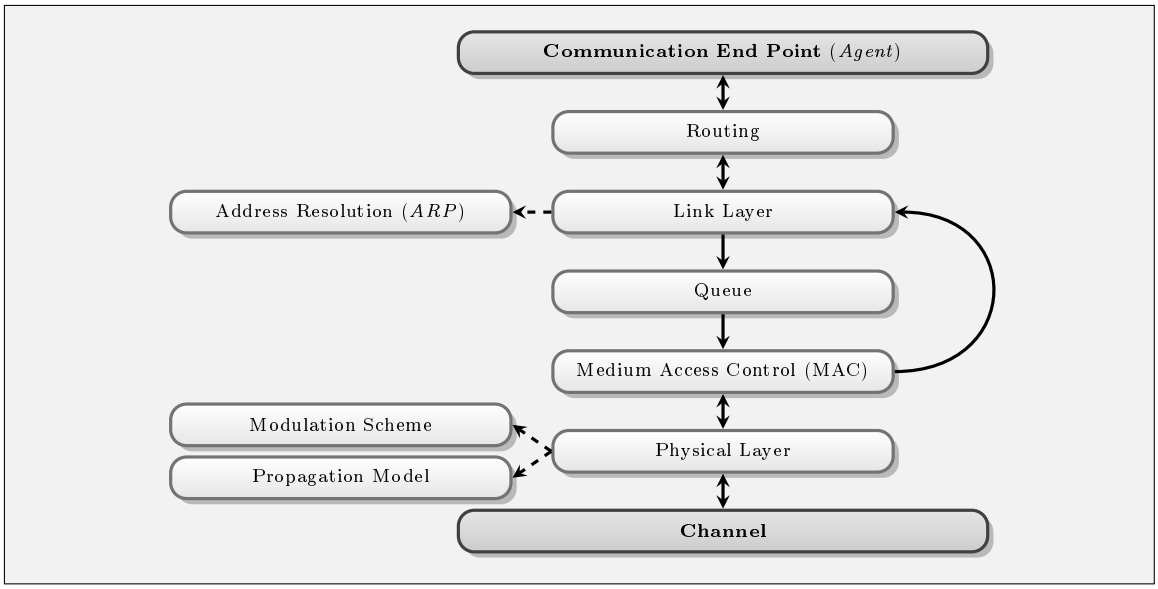


Figure 4.1: ns-2 layer model

An application packet is generated by the communication end points, called agents in ns-2. It is passed down the stack to the routing layer, which is mainly responsible for addressing, e.g., determining the next hop in multi-hop communication. Logical-to-physical address resolution is done at the link layer, before outgoing packets are sent to the queue. By default, the latter uses a blocking mechanism. Being initially unblocked, the queue changes to a blocked state, after having sent a packet to the MAC. In this state, further packets are buffered, until the MAC eventually unblocks the queue on completed packet delivery to the physical layer. At the physical layer, the packet is stamped with information, such as sending power, and sent to the channel, which is common to all nodes. On packet reception, the channel distributes a copy of the packet to each potential receiver. Packets are delivered with individual radio propagation delays, which are calculated by taking the ratio of distance to the sender and the speed of light.

On reception of an incoming packet from the channel, the physical layer determines, if the packet could actually be received by the radio hardware. This is required, because the channel does not consider the receiving power during packet distribution. If a modulation scheme is configured, the physical layer simulates bit errors for received

packets, which are finally sent up through the MAC. Incoming packets do not pass the queue, they are directly delivered to the link layer. The routing layer has to check whether the packet has to be forwarded (routed). If this is the case, it looks up the next (intermediate) receiver in its routing table and writes this information into the packet, before sending it down the stack again. If the packet was intended for the local node, it is sent up to the appropriate agent(s).

4.1.2 Wireless Physical Layer and Channel

On reception of an incoming packet from the channel, the physical layer obtains its signal strength using the specified propagation model. Three models are currently available in ns-2: free space, two-ray ground, and shadowing. In general, they behave like the pathloss model described in Section 2.1.3, but additionally make use of the radio frequency and antenna characteristics. An easy way to include their influence into the presented model is to adjust the sending power P_i^T of the sender v_i correspondingly. The free space model is based on $\alpha = 2$; the two-ray ground model uses $\alpha = 4$, but behaves like the free space model at low distances. The shadowing model allows for customization of α. In addition, it adds to the signal power a random variation that is drawn from a normal distribution with zero mean and configurable standard deviation. Concrete formulas for all models can be found in the ns-2 manual [FV08].

Having determined the receiving power, the physical layer compares it with two thresholds. The first one is the *carrier-sense threshold* θ_{cs}, which is the sensitivity of the radio transceiver. Packets with a signal below that threshold cannot be received by a real radio transceiver and are thus discarded by the physical layer of ns-2. This procedure is necessary, because the ns-2 wireless channel distributes packets to all nodes within a given radius of the sender. This delivery radius is the distance at which the signal power has decayed to θ_{cs} plus a safety margin of 5 m. The wireless channel calculates it once on first packet reception, using the transmission power of the sending node. Hence, using individual transmission powers may lead to undesirable results. Additionally note that if the shadowing model is used, packets are distributed to all nodes in the network, because radius calculation is not possible due to the signal randomization.

The second threshold is the *receive threshold* θ_{rx}. A radio transceiver uses it to decide when to start a reception. Its meaning is to keep the bit error rate below a desired limit (cf. Equations 2.3 and 2.4). In ns-2, packets with signal power of at least θ_{rx} are sent to the MAC. The physical layer can be configured to apply a statistical

bit error model to those packets. Calculation of bit error probability is achieved via configuring a modulation scheme. The physical layer uses this probability in order to decide whether marking a packet as erroneous or not. A packet with signal power equal to or above θ_{cs} and below θ_{rx} is always marked erroneous, but sent up to the MAC. This is required, since the MAC, e.g., may have to perform collision avoidance. If the physical layer has sent a packet to the MAC, it gives up control about it. This is a design limitation of ns-2 and implies that collision detection cannot be done by the physical layer and must be also accounted for by the MAC.

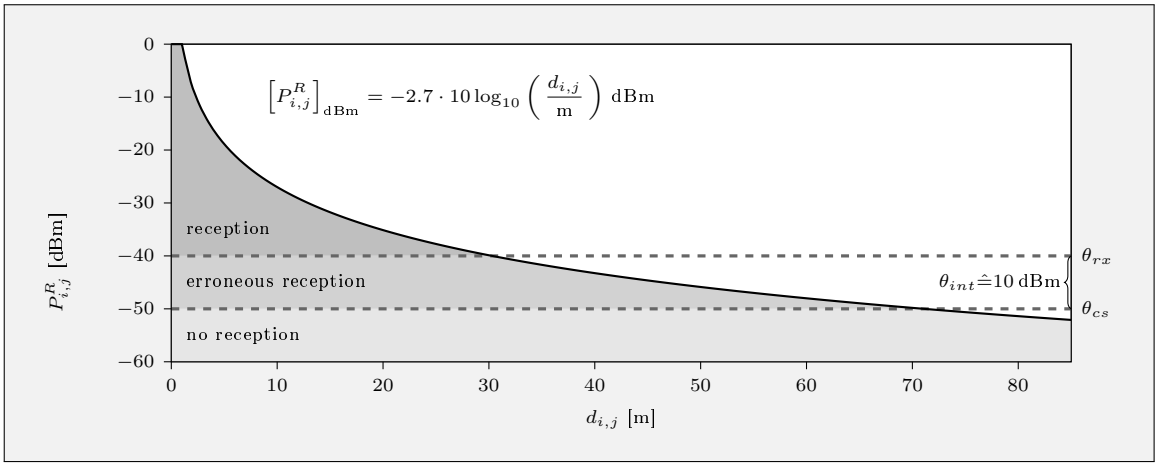
As stated before, the carrier-sense and receive threshold are characteristics of the radio chip. Changing the communication range is done by adjusting the transmission power. In ns-2, however, it is required to set a fixed transmission power and change θ_{cs} and θ_{rx}, since the channel assumes a common transmission power when setting up the packet delivery radius. Configuring the thresholds must be done appropriately. In general, θ_{rx} is simply set to guarantee the desired communication range. Bit error probability, if desired, is separately configured via the modulation scheme. The setup of θ_{cs} is more difficult. On the one hand, it must be chosen low enough to ensure that all packets that may cause interference are sent up to the MAC layer. This depends on the chosen interference model. On the other hand, it should be high enough to prevent the channel from futile packet distribution, since this increases running time of a simulation.

4.1.3 MAC Layer

In the first place, the MAC layer is required to implement protocol specific functionality, e.g., CSMA or TDMA behavior. Besides, the MAC must perform basic tasks, such as simulating packet transmission time and packet collision. A common approach on this matter is as follows. Before the MAC sends a packet to the physical layer, it calculates the transmission time by means of the packet length and the channel bandwidth, which is a MAC parameter in ns-2. It then stamps the packet with that information. If the MAC receives an incoming packet from the physical interface, it delays delivering the packet to the link layer by the attached transmission time. In case the MAC receives a new packet from the channel during it is already receiving another packet, it must check for a collision. If the later packet does not cause a collision, this is called *capturing*. In ns-2, a packet is simply dropped in this situation. If it causes interference, the MAC keeps the packet that will be longer on the channel and drops the other one. However, the kept packet is marked as erroneous. In conse-

quence, only the first detected packet of an overlap can be received successfully. This model is used, e.g., in the IEEE 802.11 W-LAN implementation for ns-2 [Liu] and is explained in [WWJ+05]. A valuable outcome is that it suffices to chose $\theta_{cs} = \theta_{rx}/\theta_{int}$ for collision detection at the MAC layer. In ns-2, the interference threshold θ_{int} is called capture threshold.

Of course, this model is not as precise as the SINR from Equation 2.3, because it does not accumulate radio signal powers. Yet, using the SINR would require the MAC of each node to keep track of the transmission times and receiving powers of all packets on the channel. Even with moderate simplifications, such as neglecting packets, if their receiving power is below θ_{cs}, this would have two implications. Firstly, implementation of the MAC protocol would become complexer, particularly due to mixing MAC functionality with the simulation of physical effects. Secondly, simulation runtime would be increased. In addition, it is questionable, if implementing this complex structure would lead to a higher simulation accuracy, as propagation models are already imprecise. Andel and Yasinac show that improving the preciseness of a single component of a simulator does not necessarily increase simulation accuracy [AY06].



■ **Figure 4.2:** Example receiving power and radio thresholds

Figure 4.2 depicts the receiving power in relation to node distance using the pathloss model from Section 2.1.3 with $\alpha = 2.7$, $d_0 = 1\,\text{m}$, and $P_i^T = 1\,\text{mW} \hat{=} 0\,\text{dBm}$. In order to guarantee a communication range of 30 m, it follows that $\theta_{rx} \hat{\approx} -40\,\text{dBm}$ is required. Here, the MAC implements the threshold-based interference model with $\theta_{int} \hat{=} 10\,\text{dBm}$ (cf. Equation 2.2). This leads to $\theta_{cs} \hat{\approx} -50\,\text{dBm}$, which results in a delivery radius of $R_{int} \approx 71 + 5\,\text{m}$.

4.2 Simulating Data-Gathering in ns-2

As outlined in Section 3.4, the influence of different TDMA schedules on a two-phase data-gathering application shall be compared via simulation. For this purpose, an analysis is performed in order to derive and specify a simulation framework.

4.2.1 Analysis

As stated in Section 3.1, the main focus of this thesis is on the influence of the scheduling schemes on data-gathering. Therefore, it is sufficient to restrict each simulation run to a single collection phase. However, it is not required to simulate all parts of such a phase (cf. Section 2.2) using ns-2. Tree construction and the actual slot-assignment algorithms are not to be investigated. Sourcing them out has considerable advantages. Settings have to be generated only once and can be used for different simulation runs. This saves simulation time and leads to comparability of the results, because it allows to vary one single parameter at a time. Additionally, the implementation for ns-2 is simplified, as only the collection protocol using TDMA has to be considered here.

Splitting the collection phase as proposed above implies that a simulation framework should be founded on three pillars: the simulation settings, the data-collection protocol running on each node, and a simulation script connecting them. Figure 4.3 depicts the general setup of such a simulation framework. Before discussing these components in detail, a general analysis is performed.

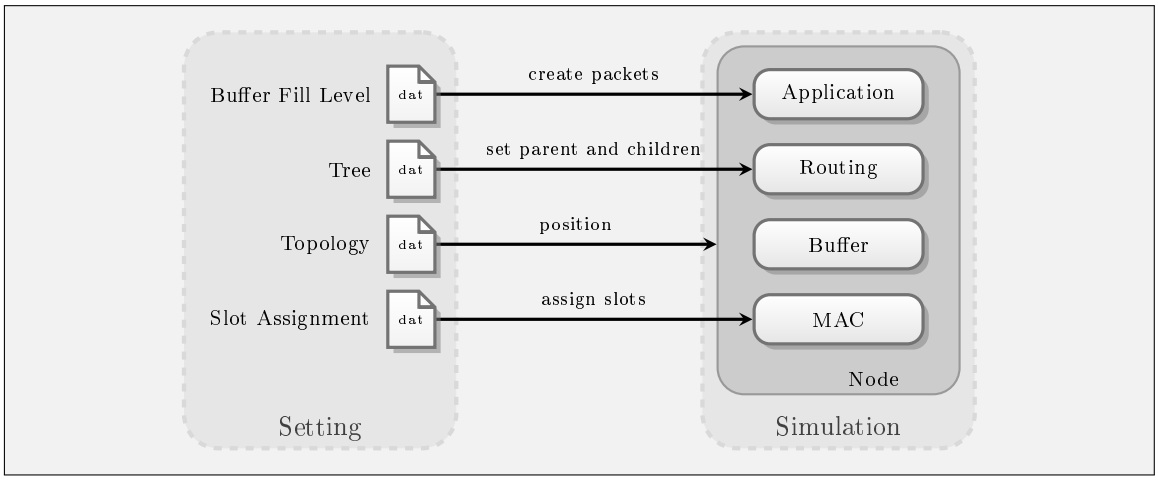


Figure 4.3: Simulation architecture

Simulation settings describe the network and the configuration of each node required to perform data-gathering using TDMA as introduced in Section 2.2.1. Hence, a setting can be divided into four distinct parts: the topology, giving information about node positions, the routing tree, the initial buffer fill levels, and the slot assignment.

As settings are precalculated and thus usable in various simulation runs, a suitable simulation script is needed to load a specified setting to setup a single run. Nodes are created and positioned as defined by the topology. The application of each node is prompted to generate the appointed initial buffer fill level. The routing component is set up according to the tree, and slots are assigned as created by the slot assignment algorithm. This configuration process is also illustrated in Figure 4.3. Besides, it includes setting up diverse ns-2 parameters and the layered stack with the modules presented in this chapter. In addition, the ns-2 link layer must be exchanged. It provides an address resolution protocol (ARP), which is unnecessary in a wireless sensor network. Nodes have unique identifiers that are used as their address, so that the ARP protocol creates undesired packets. Hence, it is sufficient to replace the default link layer with a simple implementation that only passes received packets up and down the stack, respectively. If the simulation is set up, the script initiates the actual simulation via ns-2. After completion, the script generates and stores detailed simulation results for subsequent evaluation. Here, it is useful to generate as much data as possible and leave evaluation to a dedicated tool, because this guarantees flexibility. Specifically, metrics can be easily adjusted or added.

The actual data-collection protocol that runs during a simulation employs a TDMA-driven MAC, as this reduces energy-consumption according to the results of Sections 2.2.4 and 2.3.1. Furthermore, it has to fulfill the requirements discussed in Section 2.2.1. Each node forwards data into the direction of the sink; this includes data generated by the node's application layer and incoming data from remote nodes. If a node has forwarded all data, it sleeps for the remainder of the collection phase. As recommended in Section 2.2.2, a data-gathering tree ought to be used for routing. To achieve reliability, each node must store incoming packets from its children in its buffer and send an acknowledgment, signaling if the packet could be stored. In addition, a node must not remove a packet from its buffer, before the corresponding acknowledgment is received. Figure 4.4(a) illustrates the flow of a data packet on an example two-hop way from the source to the sink. In general, sending a data packet and the corresponding acknowledgment in the same slot increases efficiency. This has several reasons. Firstly, the role of each node is a-priori known: A child sends data in

its slots and a parent listens in the slots of its children. Secondly, data packets and acknowledgments are of different size, so that sending them in different slots leaves a part of a slot unused for acknowledgments, which increases overall runtime. Thirdly, a child gains fast feedback about packet reception. As explained in Section 2.2.3, buffer overflows have to be prevented for maintaining energy-efficiency. For this purpose, an inornate but effective solution is to supply acknowledgments with a counter ω_c that demands the corresponding child to skip ω_c slots. Its value depends on the current buffer fill level $\tilde{L}$.

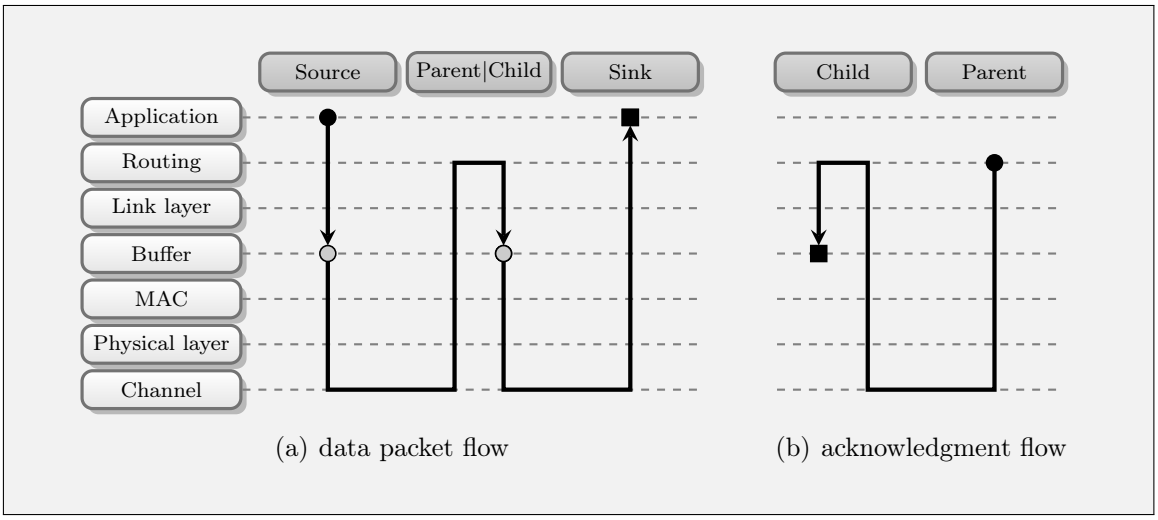


Figure 4.4: Flow of data packets and acknowledgments

Since different types of schedules will be investigated, the MAC layer has to offer a general slot storage mechanism. Energy-efficiency requires that a node knows for each slot, if the latter can be used for sending, is assigned to a child and thus reserved for receiving from that child, or if sleeping is possible (cf. Section 2.2.1). As a result, the slot storage mechanism must provide every node with its sending slots and the receiving slots of its children. From this information, a node can derive the slots, in which it may sleep. In addition, slot storage is required to include the mapping between receiving slots of a node and the corresponding child. This mapping enables a node to identify link failures, which is mandatory to prevent idle listening in the corresponding slots, as described in Section 2.2.1.

A reasonable strategy for detecting interrupted links is to give up receiving, if no packet is received in a certain number of consecutive slots belonging to the same link. This mechanism exhibits a problem in the case of a buffer underrun. If there is no packet to send, while at least one child still has to forward data, the node's parent must be detained from mistakenly considering the link as interrupted. Sending a

keepalive packet each time this situation occurs is a poor decision, as it is not energy-efficient. This is particularly severe, if the affected node has further sending slots at its disposal, before expecting the reception of a new packet. Note that keepalive packets have to be acknowledged, since the detection of interrupted links is a two-way mechanism. To reduce the number of keepalive packets, a node must predict the number ω_p of potentially unusable sending slots once a buffer underrun occurs. ω_p is attached to a packet sent to the parent and advises the latter to skip ω_p slots. Note that this procedure works similar, but in opposite direction, to the flow control explained above.

In general, the four components of the protocol, as depicted in Figure 4.3, can be directly mapped to the ns-2 stack (cf. Section 4.1.1). However, the just described functionalities have to be assigned to these layers under consideration of the ns-2 stack. Firstly, the routing layer has to send acknowledgments, because the MAC cannot access the buffer. Hence, it does neither know, if an incoming packet can be stored, nor is it capable of calculating ω_c. Secondly, these acknowledgments must not be stored in the buffer, but are required to be immediately sent down to the MAC. This is needed in order to allow the sending of a data packet and the corresponding acknowledgment in the same slot. Thirdly, on the reception of an acknowledgment, it must travel all the way from the MAC up to the routing layer and down to the buffer, because a packet cannot be removed from the buffer except by the buffer—the ns-2 queue—itself. Figure 4.4(b) shows the corresponding flow of a single acknowledgment from the parent to a child. Fourthly, only the MAC layer has knowledge about a node's number of slots and the ones of its children. Hence, calculating ω_p is to be done at this layer. It is also responsible for sending keepalive packets due to the same reason.

4.2.2 Simulation Settings

A simulation setting consists of a topology, a routing tree, initial buffer fill levels, and the actual slot assignment. For each topology, there may be several trees and different initial fill levels. Slot assignment is performed as the last step. E.g., load aware Type III requires a tree and the fill levels for generating an assignment. A set of tools, developed in Perl, exists to create the required simulation settings. Their main characteristics will be introduced in the following. A detailed overview about all available tools is provided in Appendix B, and the implementational effort can be obtained from Table 4.2.

As discussed in Section 3.4.2, the number of nodes N and the density ϱ are the two key parameters of a topology. Hence, a topology generator is designed to comply with these needs. It creates a random topology from a specified number of nodes and a given density. At first, nodes are aligned in a scaled grid meeting the specified density and are then randomly distracted. The random distribution used for this is configurable. As the estimations made in Chapter 3 are based on minimum-depth trees, an additional topology generator exists. It is intended for creating appropriate topologies to validate these estimations.

Tree construction is performed by a breadth-first search in conformance with Section 2.2.2. Among all nodes v_i already added to the tree, the one with lowest depth h_i and less than C children is selected that has the closest neighbor v_j not yet connected to the tree. v_j is then added as a child of v_i. Note that C is a parameter here. Of course, this approach cannot be implemented easily in a real wireless sensor network, because distances are usually not available. Yet, application of this strategy is suitable for simulation, because receiving power is a frequent metric for link-quality and is calculated from the distance between sender and receiver in ns-2 (cf. Section 2.1.3). Note that the underlying algorithm permits the construction of ideal routing trees for $C = \infty$. Therefore, comparison under perfect conditions is possible, so that any side-effects that may influence a particular scheduling scheme are prevented. Furthermore, the sensibility of the schemes to C can be analyzed.

Another outcome of Section 3.4.2 is to investigate the influence of different initial buffer fill levels. Hence, the appropriate script supports the generation of both evenly and randomly distributed fill levels among nodes.

Slot assignment has been implemented as suggested in Sections 2.3.2 and 3.3.1. For Type I, the CCH algorithm is used, as it has been shown to produce few slots (cf. Section 2.3.2 on page 21). However, two variants of CCH are considered, as simulation results have revealed that the simple 3-hop approach generates schedules suffering heavily from collisions. Hence, an alternative has been conceived that takes the interference factor γ as a parameter. It derives 2-hop interference neighborhoods from the topology and is thus capable of producing collision-free schedules. To distinguish between these two variants, the interference-based approach is called CCH γ.

The organization of the settings reflects the dependencies of the different parts. A topology is required to build a routing tree. Slot assignments depend on all other three parts. The logical order is thus given as: topology, tree, buffer fill levels, slot assignment. Hence, it is beneficial to store settings in a subfolder structure containing the parameters used during construction, particularly the number

of the nodes N, the maximum number of children C, the buffer size B, minimum and maximum initial fill levels, and the type of slot assignment. The density is not included in the path, but is implicitly encoded by the simulated area size (length and width). An additional counter is used to allow the generation of multiple topologies with the same parameters. An example for such a path would be *./area_1000x1000/nodes_0100/count_007/tree_8/buffer_200_25_75/-slots_1_cch*. The individual setting files are placed inside the corresponding subfolder, e.g., a tree file is stored in *./area_1000x1000/nodes_0100/count_007/tree_8*. This structure permits a simple grouping and individual access strategy that is needed to support many simulation runs.

4.2.3 Simulation Environment

The logical structure of the simulation environment is shown in Figure 4.5. Simulation settings are placed in the folder *settings* using subfolders as explained in Section 4.2.2. Default OTcl configuration files for ns-2 are placed in the *config* folder. Specific simulation setups are located below *experiments*. Here, different simulation runs can be organized in a tree structure, where each subfolder may contain a more specific OTcl configuration file. A list of parameters is provided in Appendix A.

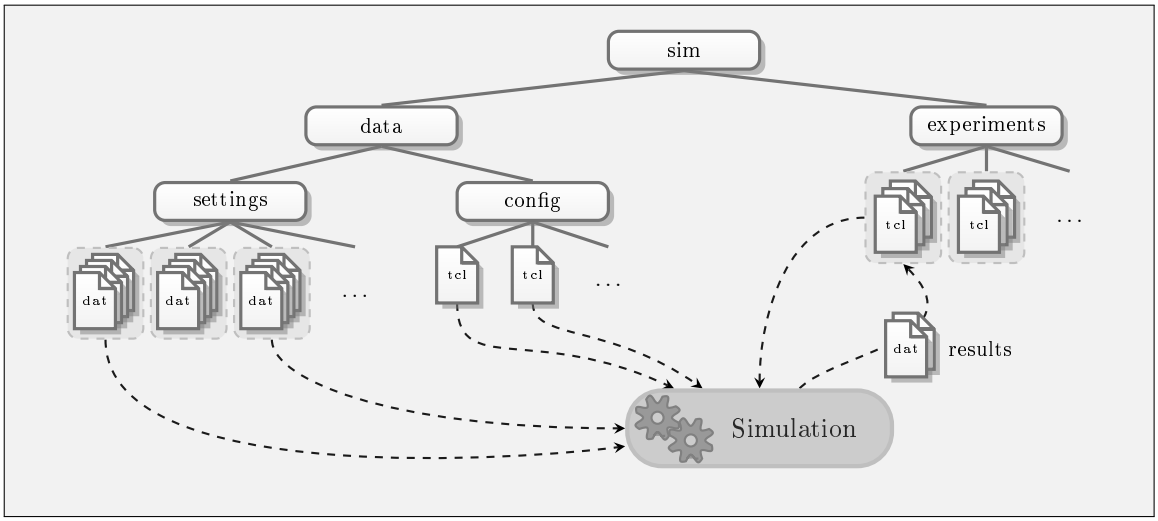


Figure 4.5: Structure of the simulation environment

A simulation run is started by navigating to the corresponding folder below *experiments* and starting the simulation script *Simulation.tcl* via the ns-2 interpreter. It is configured with environment variables that are explained in Appendix A.1.1. At first, the script invokes a dedicated OTcl-library that is responsible for loading the

desired simulation setting from the *settings* folder. Next, a default configuration file and one for the used radio chip are read from the *config* folder. This is followed by reading the individual configuration files from the *experiments* folder down to the folder from which the script has been invoked, assuring that more specific parameters have higher precedence. Configuration options are listed in Appendix A.1.2. The ns-2 simulation is then prepared by setting up the stack with the appropriate modules. Besides initializing the radio and physical interface, the settings have to be applied as described in Section 4.2.1 on page 55.

When the setup is complete, the simulation is started by the script. The latter checks periodically, if all nodes have entered the final sleeping state. For this, the MAC is required to provide an OTcl command. If all nodes but the sink have entered this state, the script stores the simulation results in the current folder. The naming and layout of those evaluation files is described in Appendix A. Finally, the Perl script *getValues.pl* has to be manually invoked to extract data according to the metrics described in Section 3.4.1. Table 4.2 gives an overview about implementational effort of the individual modules used by the simulation script.

A configurable startup script *startSimulation.pl* has been conceived to easily run many simulations. Provided with the parameters encoded in the settings path as a filter, the startup script parses the subtree of *settings* and starts a simulation for each matching path by calling the simulation script with appropriate environment variables (cf. Appendix A.1.1). E.g., it is possible to run a particular simulation for all different network sizes and trees with a given density (encoded by the area), buffer fill level, and scheduling scheme. A corresponding filter could, e.g., be *./area_1000x1000/-nodes_∗/count_∗/tree_∗/buffer_200_25_75/slots_1_cch*. This is schematically illustrated in Figure 4.5. For clarity, the subtree structures of *experiments* and *settings* are displayed as gray shapes with dashed borders. The flow of data is indicated by dashed arrows, and the individual simulation runs are symbolized by the gray shape with the gears attached. Two simulations are initiated by the startup script run from a subfolder of *experiments*. For each matching simulation setting, the default configuration files and local ones are read. Then, the simulation is performed and results are stored in the folder from which the startup scrip is invoked.

4.2.4 Data-Collection Protocol

The analysis at the beginning of this section underlines the advantages of sending a data packet and the corresponding acknowledgment in one slot. The layout of such a

slot is depicted in Figure 4.6. At the beginning of a child's sending slot, the child and its parent switch on their transceiver. However, the child does not immediately start transmission. In a real network, a guard interval is used, because clocks cannot be assumed to be precisely synchronized. Between sending and receiving, propagation delay is observed. As the acknowledgment is sent in the same slot, child and parent must turn their transceiver from the current mode to reception or transmission mode, respectively. Finally, both nodes sleep for the rest of the slot to save as much energy as possible. Another method applied to preserve energy is as follows. A node aborts listening in a slot, if packet reception does not commence in a given time.

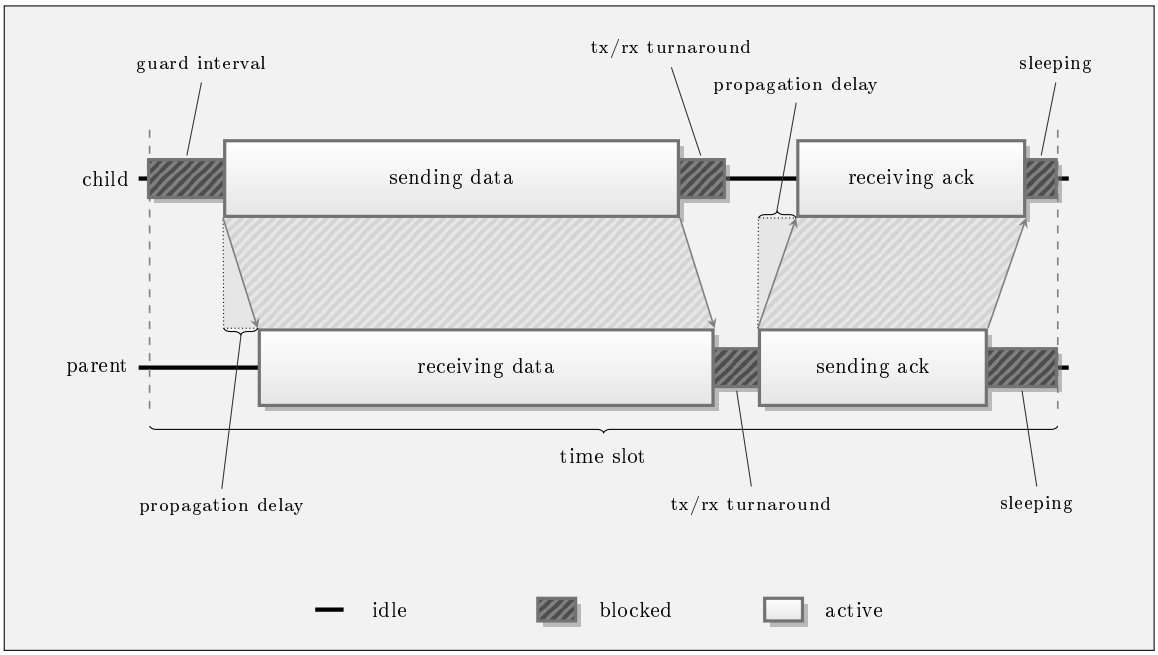


Figure 4.6: TDMA slot layout

To detect interrupted links, a data packet is resent at most r times, if the pertinent acknowledgment is not heard. After having missed $r+1$ consecutive acknowledgments, the node considers the link to its parent to be interrupted and gives up sending. Likewise, the MAC assumes a link to a child to be interrupted, if no packet has been received for $r+1$ of its slots. The choice of r has to satisfy two requirements. On the one hand, it must be chosen large enough to assure a high packet delivery rate, avoid a partitioned tree, and therefore minimize latency. On the other hand, it must be chosen small enough to allow timely detection of interrupted links and thus preserve energy-efficiency.

The packets sent during a slot carry important protocol information. Acknowledgments must include a success flag that indicates whether the packet could be stored

in the buffer. The last-packet flag is used to indicate that a node has sent the last packet, i.e., the current packet is the last one in the buffer and no more incoming packets are expected. In case the MAC receives the last packet from one of its children, it shuts down that link with a delay of r slots. This is required to keep that child from resending its last packet up to r times, if the first acknowledgment is lost. On reception of a packet from a child, the corresponding acknowledgment is equipped with the flow control counter ω_c. The value of ω_c is calculated as the difference between the current buffer fill level and the soft limit $\tilde{B}$. In case ω_c is negative, it is adjusted to zero. In extension to this, an abort flag is set, if $\omega_c > 0$ and if the link to the father is interrupted. This abort flag instructs the corresponding child to stop forwarding packets, since the parent's buffer is full. Packets are stored in the buffer, until their successful transmission to the parent has been acknowledged.

This way of flow control interferes with the detection of interrupted links, as a node may mistakenly declare a link interrupted. The remedy is to store for each child the value of ω_c attached to the last acknowledgment. However, this does not imply that a node does not have to listen in the following ω_c slots of that child. The reason for this is that the acknowledgment could actually be lost, so that the child would not know about ω_c. It would keep sending to its parent, but would not receive an acknowledgment. Hence, it might accidentally consider the link to be interrupted. As discussed during the analysis on page 56, buffer underruns require special handling due to the same reason. A parent must be informed about this situation using ω_p, so that the node suffering from underrun can skip slots. For this purpose, the MAC layer calculates ω_p as the difference between the number of the node's sending slots and receiving slots of its active children. Here, the term active implies that the link to the child is not interrupt, that the child has not sent its last packet, and that it is currently not skipping any slots. If ω_p is positive, it is included into a packet to the parent. Due to the same reasons as in the preceding paragraph, a node is not allowed to skip the just calculated number of sending slots, unless its parent has agreed. The latter can simply do so by attaching a received ω_p to the corresponding acknowledgment. To prevent conflicts between ω_c and ω_p, the larger one is used.

4.2.5 Extensions for Dynamic Slot Reuse

The just presented protocol has to be extended in order to meet the requirements for reusing slots as needed by the enhanced Type II TDMA schedule. In particular, the MAC layer must provide the reception and forwarding of slots that can be reused.

Furthermore, a reuse strategy is required. It is designed based on the description in Section 2.3.2 on page 23. If a parent node has received the last packet from a child, or if the corresponding link is interrupted, the parent may claim the slots of that child. However, it may not reuse all of these claimed slots, but must forward a specified fraction of them. The same strategy is also applied, if a node receives slots from one of its children. In case a node wants to use new slots for sending, it must inform its parent, so that the latter will listen in those slots. Therefore, a node must not actually use new sending slots, before its parent has acknowledged. As a result, two distinct sets of slots can be attached to a packet: the set that a node wants to use itself and the one it forwards. Both of those sets can be sent piggybacked or in standalone packets, e.g., if no more data packets have to be sent or as a replacement for a keepalive packet.

Slots must not be used by more than one node at the same time, as this will potentially lead to collisions. This problem could be solved, if every node kept track of each slot it has received in a collection phase. Due to the restricted amount of memory available on a sensor node, this is not feasible. Hence, a different approach has to be chosen. First of all, packet loss must be coped with. If a node has received a set of slots from a child, but the acknowledgment is lost, the sender will resend the packet along with the attached slots. On the newly reception of the same set of slots, the receiver may come to a completely different decision about which of those slots to use itself and which of them to forward. In particular, this will occur, if the same node has received another set of slots meanwhile. The solution is to identify duplicates via the MAC sequence numbers. This implies that the sender may not alter the set of slots attached to a resent packet, because this could lead to the loss of slots. If a parent node has received the last packet of one of its children, the corresponding slots must not be used before another r slots (of that child) have elapsed. Immediately reusing slots may lead to collisions, as the child will resend its last packet, if the acknowledgment was lost.

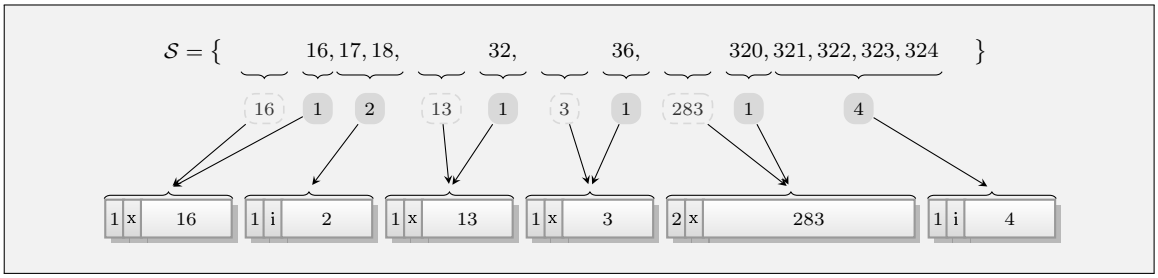


Figure 4.7: Example run-length coding of a set of slots for Type II

	Bytes required for encoding subsets $\mathcal{S}'$ of size						
	3	5	7	10	15	20	50
No Coding, 16 bits per slot	6.0	10.0	14.0	20.0	30.0	40.0	100.0
RLC Coding, $\lvert\mathcal{S}\rvert = 200$	4.5	6.4	8.2	11.0	15.8	20.7	47.9
RLC Coding, $\lvert\mathcal{S}\rvert = 100$	4.0	5.9	7.9	10.8	15.6	20.2	39.0
RLC Coding, $\lvert\mathcal{S}\rvert = 50$	3.9	5.9	7.8	10.6	14.7	18.1	2.9
RLC Coding, $\lvert\mathcal{S}\rvert = 10$	3.8	5.2	5.6	2.9	—	—	—

Table 4.1: Average byte usage for random subsets $\mathcal{S}' \subseteq \mathcal{S} \subseteq \{0, \ldots, 999\}$

Sending a set of slots must be done efficiently. An advanced run-length coding (RLC) scheme has been designed to meet this end. The first slot is used as an offset. Coherent subsets and gaps including the first slot behind the gap are coded as follows. The first bit of a block indicates the length of the block, which may be 1 or 2 bytes. The next bit determines, if the block encodes a consecutive subset of slots (i $\hat{=}$ **i**nclusive) or a gap (x $\hat{=}$ e**x**clusive). The remaining bits are used to encode the size of the subset or gap, respectively. This scheme is capable of coding up to 2^{14} slots. Making use of the implicit coding of the first slot after a gap has two benefits. Firstly, single slots are already coded by the corresponding gap. In addition, the size of a subset is decreased by one, which may save memory.

An example slot coding is shown in Figure 4.7. The first slot (or the initial gap of 16 slots) is coded as the offset. Since 16 is encodable with 6 bits, the block has size 1. Note that the first block always codes a gap (even if it is empty). The first subset has size 3, but only 2 slots have to be coded, as slot 16 is implicitly included in the prior block. It follows a gap of 13 slots. This gap, including the next slot 32 is coded in the third block. Coding of the fourth block is accordingly. The next gap has size 283, so that 6 bits do not suffice. Hence, the block size must be 2. The final block represents the last four slots, as the first slot of that subset has been coded in combination with the gap. The actual implementation makes use of a parameter that limits the available number of bytes, so that possibly only a subset can be coded.

The presented coding scheme is easy to implement and gives a compact representation. Table 4.1 compares the average number of bytes required for coding subsets $\mathcal{S}'$ of different sizes. The $\mathcal{S}'$ have been drawn independently from random sets $\mathcal{S} \subseteq \{0, \ldots, 999\}$, and each combination has been simulated 10000 times. It shows that savings are considerable even for large $\mathcal{S}$ and small $\mathcal{S}'$, i.e., few, rather not being consecutive, slots with a large range are encoded.

4.3 Implementation of the Framework

In the following, the implementation of the data-collection protocol explained in Section 4.2.4 will be sketched. In general, inter-layer communication in ns-2 is done by adding appropriate information to packet headers. Yet, this method is not very efficient and partly unsuitable for the protocol to be implemented, so that modification of the ns-2 core and the layered stack is required at some points. As a result, cross-layer interaction will be particularly focused in this section. The implementation of the protocol is designed to be highly configurable. An annotated list of all parameters can be found in Appendix A.1.3. Table 4.2 shows the implementational effort in lines of code for each of the layers.

4.3.1 Application Layer

The application layer is encapsulated by the class `TreeAppAgent`. It offers the method `createPackets` that takes the number of packets to create as an argument. Packets are immediately created and sent down the stack. Their payload size can be configured via OTcl, and the last-packet flag is set in the last packet. As application and routing can only communicate via the exchange of packets, this is the easiest way to inform the routing layer about the last local packet. The OTcl command `send-and-sleep` is provided by `TreeAppAgent` in order to make packet generation possible by the simulation script. The method `recv` accepts packets from the routing layer. `TreeAppAgent` provides two counters, one for the number of packets created and one for the number of packets collected. Figure 4.4(a) sketches the flow of data packets from a source node to the sink. Here, the shown intermediate node acts as the parent of the source and as the child of the sink.

4.3.2 Routing

Routing is split into two parts. The abstract base class `TreeRouting` defines a common routing interface and takes care of providing OTcl commands that are called by ns-2 during stack setup. `StaticTreeRouting` is derived from `TreeRouting` and contains all functionality as specified in Section 4.2.4. Adding a new routing agent requires changes in the ns-2 core. Stack setup must be altered in order to accept the new routing component and perform the needed initializations. Listing 4.1 shows how this task can be wrapped by a method that creates a new instance of `StaticTreeRouting` (it is bound to the OTcl class

`Agent/TreeRouting/Static` following ns-2 naming conventions), configures it with the node's address, and finally attaches the routing module to the node. An additional change is required in order to make ns-2 call this method, when checking which routing agent has been chosen.

```
Simulator instproc create-statictreerouting-agent { node } {
   set addr [$node node-addr]
   set ragent [new Agent/TreeRouting/Static]
   $ragent addr $addr
   $node set ragent_ $ragent
   return $ragent
}
```

Listing 4.1: Integration of a new routing module into ns-2

`StaticTreeRouting` uses a STL vector for duplicate identification. Entries consist of the source (not the last sender) of a data packet and its sequence number. The length of the vector can be configured via OTcl. Setting up the routing table is done with the OTcl commands `set-father` and `append-child`. The routing table has been implemented in a dedicated class `TreeRTable`. It provides methods to set and get the parent and the children. In addition, it keeps track of the state of the parent and the children. This information is required by the routing layer in order to decide when a node sends its last packet, as the last packet has to be marked by setting the last-packet flag. In addition, the routing must set the abort flag in an acknowledgment, if its buffer is full and its parent is not active, i.e., the link is interrupted.

`StaticTreeRouting` has to handle three different packet types: incoming and outgoing data packets as well as incoming acknowledgments. When outgoing packets are received from the application, the last-packet flag must be inspected. If it is set, the routing layer stores this information and clears the flag, unless it does not have any active children. In addition, the routing takes care of addressing the packet correctly and setting up a sequence number for duplicate identification at the routing layer. Finally, the packet is sent to the buffer. On reception of an incoming data packet, the routing must store it in the buffer, if it is not a duplicate, and issue an acknowledgment. In order to generate the latter, the routing layer needs to know the state of the buffer. Flow control demands for knowledge about its fill level and size, setting the success flag implies to know whether the newly received packet can be stored. Yet, as depicted in Figure 4.1, the link layer is blocking the way, so that there

is no direct connection between routing and the buffer. Hence, the ns-2 core must be modified to connect routing and buffer. However, this connection is only used to obtain the required information from the buffer. Storing packets and removing them is done by sending packets and received acknowledgments down the stack. This approach has been chosen in order to change the behavior of ns-2 as little as possible.

4.3.3 Buffer

A new buffer has been conceived in order to reflect the requirements from Section 4.2.4. The class `TreeRoutingQueue` realizes a buffer that may have an unlimited size and a soft limit for flow control. In addition, its interface suffices the needs of the just presented routing layer by providing the methods `isLimitObeyed` and `softLimit`. The corresponding variables have to be configured via OTcl. The ns-2 base class `Queue` additionally supplies the method `limit`. An important difference to the default behavior of this class is, that a packet is not removed from the buffer, when it is sent down to the MAC. This is necessary, for the routing layer requires the precise buffer size, as explained above. Hence, acknowledgments received from the parent must be passed from the routing layer to the buffer. The overall packet flow of an acknowledgment is shown in Figure 4.4(b). In order to be compatible to an arbitrary MAC, which was particularly helpful during testing, outgoing acknowledgments are treated as regular packets and are stored in the buffer. Yet, this is not desired in the real protocol, as receiving a data packet and sending the according acknowledgment has to be performed in the same slot. The class `TreeRoutingPrioQueue`, which is built on top of `TreeRoutingQueue`, introduces this feature. Outgoing acknowledgments are directly sent down to the MAC.

4.3.4 MAC Layer

The MAC layer has been split into several classes for the separation of general TDMA behavior, protocol implementation, and slot storage. Those issues will be discussed in the following.

Implementing a slot layout as described in Section 4.2.4 requires taking the findings from Sections 4.1.2 and 4.1.3 into consideration. Propagation delay is performed by the channel, but simulating packet reception and transmission must be done at the MAC along with collision detection. Hence, a detailed modeling of the MAC layer is required. This can be achieved by designing two state machines, one for sending and

one for receiving. Unused slots do not require further explanation, because a node simply sleeps until the next alloted slot.

Figure 4.8 displays the abstraction of a sending slot. It generally reflects the child's view in Figure 4.6. Transitions are labeled with events, important actions are embraced by brackets. Incoming packets are dropped, until the node has started waiting for the acknowledgment. This behavior is chosen, as the transceiver is assumed not to be capable of receiving a packet while sending. When receiving an acknowledgment, collision handling as described in Section 4.1.3 must be performed. Besides this, three transitions are of special importance. Firstly, if a node does not have a packet to send, it goes directly to the sleeping state. Secondly, waiting for an acknowledgment is given up after a timeout. Thirdly, if a single packet has been received, it must either be processed or discarded, depending on whether reception was successful.

The counterpart of the sending slot is illustrated in Figure 4.9. A parent initially waits for packet reception. If reception does not start within a given period, the parent goes to the sleep state. Note that this receive timeout must be chosen larger than that in the sending state machine due to the guard interval. While receiving a packet from a child, a parent node must perform collision handling. If reception has finished and no collision has taken place, the node must check if reception was successful, i.e., the packet has not been marked erroneous by the physical layer. This inspection must not be done earlier, as discussed in Section 4.1.2. If the packet has been received without errors, the transceiver is turned over to the sending mode. While transmitting the acknowledgment, all incoming packets are ignored again. Finally, the parent sleeps for the remainder of the slot and switches off its transceiver.

The just explained behavior is related to the protocol in Section 4.2.4 only in parts. Hence, the abstract base class `TreeTDMA` has been designed to encapsulate this basic TDMA behavior. It provides two interfaces, as shown in Figure 4.10. On the one hand, callback methods are defined. They must be overwritten by an inheriting class and include, e.g., signaling a new slot, completed transmission and reception. On the other hand, actions, such as transmission, determining failed receptions, and entering the sleep mode, are defined in the functionality interface and are provided by `TreeTDMA`. Hence, interaction between a derived class and `TreeTDMA` is to react on callbacks by invoking the required method. Here, `TreeTDMA` only initiates a new sending or receiving slot and handles incoming packets along with collision detection. All other steps have to be initiated by a derived class. In this context, the latter can access the current state and slot type provided by `TreeTDMA`. Detailed configuration of `TreeTDMA` is possible through OTcl (cf. Appendix A.1.3).

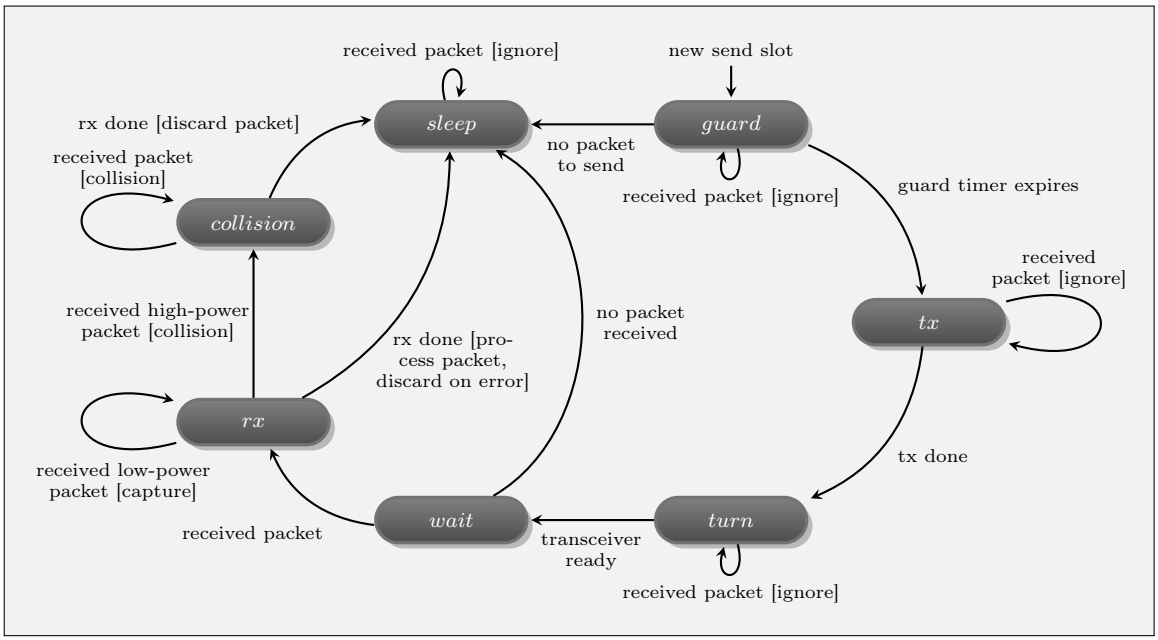


Figure 4.8: MAC layer state machine for a sending slot (child); the abbreviations rx and tx stand for reception and transmission

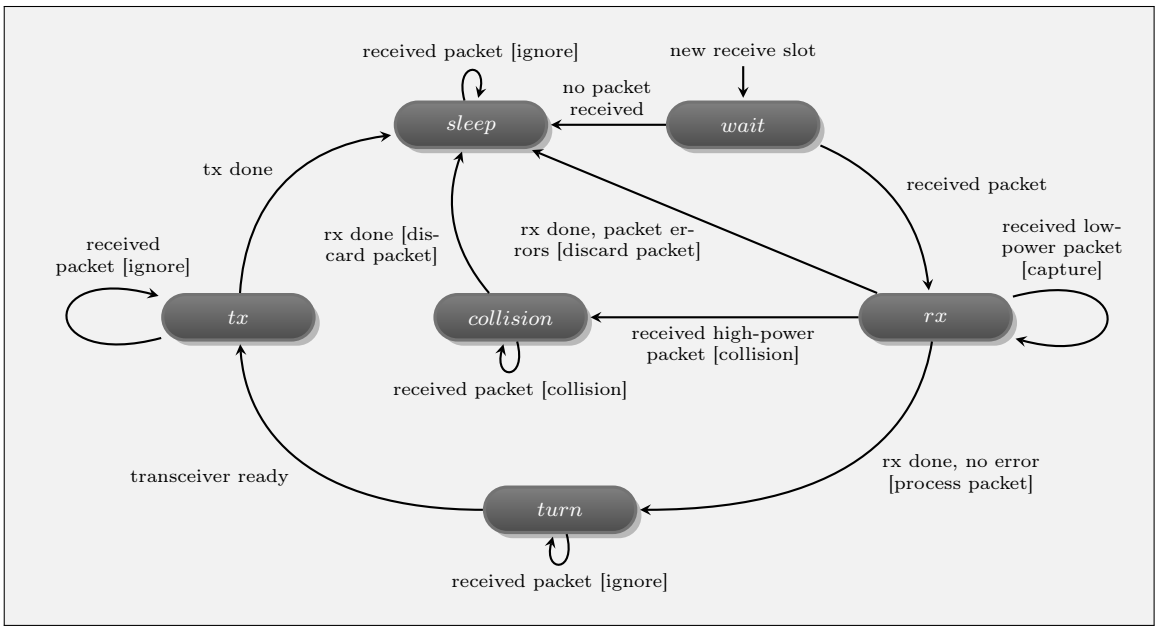


Figure 4.9: MAC layer state machine for a receiving slot (parent); the abbreviations rx and tx stand for reception and transmission

The class diagram in Figure 4.10 also shows the classes responsible for slot management. Their main purpose is to store slots and provide an interface to let `TreeTDMA` check, if the current slot is a sending or receiving slot. In addition, it takes care of identifying duplicate packets by the MAC sequence number and interrupted links by counting missed packets. It also stores information about slots to skip and if the last packet of a child has been received. If a node does not have an active parent and no active children, it does not have to participate in forwarding packets any longer and may sleep for the remainder of the collection phase. This decision has to be made by the MAC to reduce additional connections between ns-2 layers. On reception of an acknowledgment, the success flag of the acknowledgment and the last-packet flag of the corresponding data packet have to be checked. In addition, the MAC has to investigate the abort flag of an incoming acknowledgment and inform the slot management, if it is set. Because this functionality is protocol specific, it has to be implemented by a derived class. This is done by using the interface of the slot management. It mainly consists of methods to be called on packet or acknowledgment reception, missed packets, and when sending an acknowledgment.

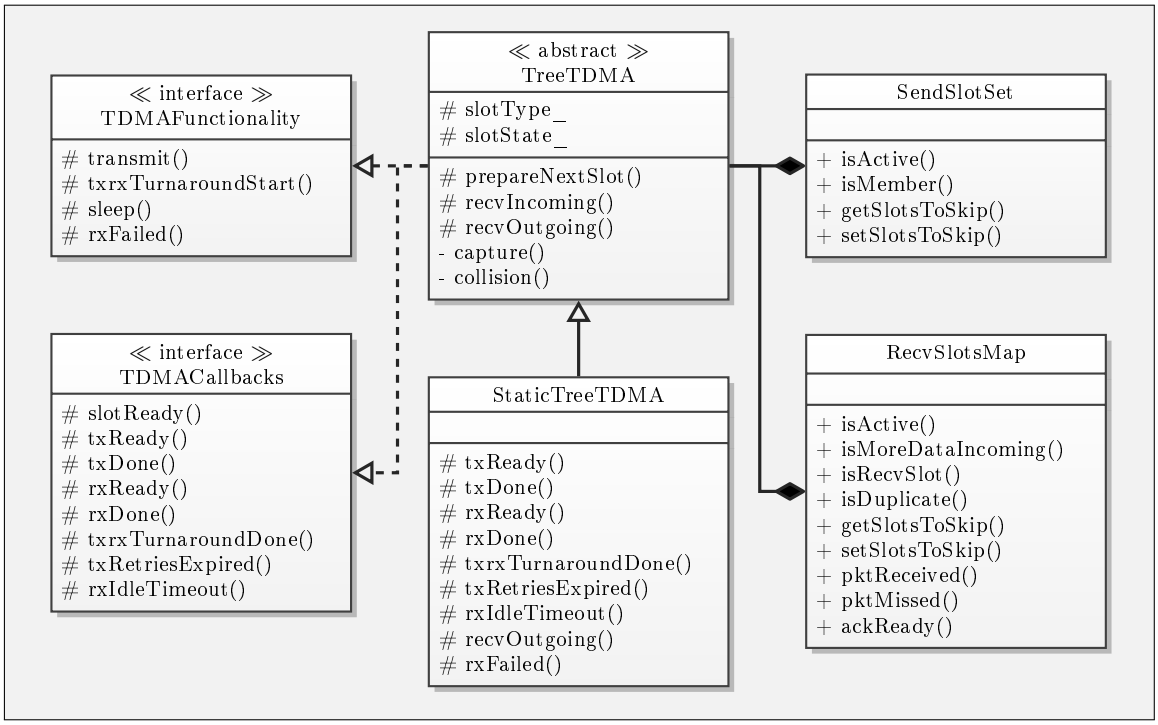


Figure 4.10: Basic class diagram of the TDMA protocol

Implementation of the basic data-collection protocol is realized by the class `StaticTreeTDMA`. To permit a convenient configuration of the TDMA schedules, it defines the OTcl commands `add-sendslot` and `add-recvslot`. Furthermore,

it actualizes all transitions as shown in Figures 4.8 and 4.9. The most relevant part of implementation is the correct handling of received packets as demanded by the protocol design.

The extensions required for Type II scheduling (cf. Section 4.2.5) are added by `SlotPassingTreeTDMA`. Configuration of the maximum number of bytes used for sending a set of slots is possible through OTcl. Two separate parameters exist for standalone packets and piggybacking. Furthermore, the forwarding strategy can be influenced by specifying the number of slots a node must forward before it may another slot itself. Slot compression, or coding, has been designed to be a plugable component to be configured via OTcl. The class `SlotPassingTreeTDMA` owns a pointer to the abstract base class `SlotCoder` that only defines the interface for slot encoding and decoding. An OTcl command is provided to attach an instance of a derived class. The described run-length coding is realized by the derived class `RLCSlotCoder`.

4.3.5 Physical Layer

The physical layer of ns-2 offers two methods for switching the transceiver on and off via the MAC. The original intention of this is to simulate energy consumption, so that calling these methods does not have an effect, unless an energy model is used. Removing this restriction accelerates simulation time considerably, because the physical layer immediately discards any incoming packet, if the transceiver is switched off. This is a striking performance improvement, as the wireless channel of ns-2 distributes a copy of a sent packet to each node within the delivery radius without looking at a receiving node's transceiver state. However, the required changes must be directly applied to the wireless physical layer of ns-2, because the affected methods cannot be overwritten by a derived class.

Simulating packet loss is done by using the bit error model from [Unt08], because the facility offered by ns-2 is erroneous. This requires to extend the wireless physical layer of ns-2 by deriving the class `WSNPhy`. It adds two base-class pointers to make a modulation and an encoding scheme available. Attaching the corresponding instances is done via OTcl. On reception of an incoming packet, `WSNPhy` obtains the bit error rate by invoking the modulation scheme. Next, the encoding scheme is used to calculate the packet error rate from the BER and the packet length. Finally, a random number in the interval $[0, 1]$ is computed and compared with the packet error rate. If the random number is smaller, the packet is marked erroneous.

Module	Language	Lines of Code
ns-2 Modules		**3905**
Application	C++	173
Routing	C++	820
Buffer	C++	292
TDMA Base Class and Slot Management	C++	1032
Basic TDMA Protocol	C++	812
Protocol Extensions for Type II	C++	776
Simulation Framework		**1560**
Startup Script	Perl	106
Initialization and Configuration	OTcl	657
Settings Parser Library	OTcl	334
Evaluation and Processing	OTcl	176
Post-Processing	Perl	287
Simulation Environment Creation		**2259**
Topology Creation	Perl	1030
Tree Construction	Perl	314
Fill Level Initialization	Perl	101
Slot Assignment	Perl	814
Sum		**7724**

Table 4.2: Lines of code

4.3.6 Evaluation and Logging

The logging system provided by [Unt08] has been used to generate detailed trace files for the different layers. This facility is helpful in order to understand internal operations of each node in a simulated network. It is also useful for debugging, but inappropriate for simulation evaluation. Continuous logging considerably affects runtime performance and log files consume large amounts of memory. Therefore, logging has only been used during the implementation and testing phase.

Evaluation of a simulation is achieved by a variety of counters and state variables that have been added to the different C++ classes. At the end of a simulation, for each node in the network, their values are written to an evaluation file. Among them are counters for sent and received packets, buffer utilization, and skipped slots. Furthermore, each node records its transceiver usage and the time at which it has entered the final sleep mode, i.e., stopped receiving and forwarding packets. In total, 54 evaluation variables are available. They are listed in Table A.5. The format of the corresponding files is explained in Appendix A.2.

Chapter 5

Simulation and Evaluation

In this chapter, the TDMA schedules discussed in Chapter 3 are compared via simulation using the framework described in Chapter 4. At first, the choice of simulation settings and parameters is addressed, followed by the evaluation of the simulation results. Finally, the characteristics of the different TDMA schedules are summarized.

5.1 Simulation Parameters

In the following, the configuration of the simulation framework is explained and the settings used for evaluation are described.

5.1.1 Configuration of ns-2

The configuration of ns-2 consists of the setup of the radio transceiver and wireless channel characteristics mainly. The choice of reasonable values is required here to make results meaningful. As the Scatterweb ESB nodes are a commonly employed hardware platform and appropriate parameters are available, configuration is based on these values.

The radio transceiver used for the ESB sensor nodes is the RF Monolithics TR1001 chip. The frequency used for transmission is at 868.35 MHz with a data rate of 19.2 kbit/s and On-Off-Keying. Maximum transmission power is settled at $0.347\,\mathrm{mW} \mathrel{\hat{\approx}} -4.6\,\mathrm{dBm}$ with a communication radius of $R_{com} = 40\,\mathrm{m}$, which is chosen according to real-world experiments with the ESB nodes. These parameters lead to $\theta_{rx} \approx 1.47 \cdot 10^{-7}\,\mathrm{mW} \mathrel{\hat{\approx}} -68\,\mathrm{dBm}$ and $\theta_{cs} = 1.47 \cdot 10^{-8}\,\mathrm{mW} \mathrel{\hat{\approx}} -78\,\mathrm{dBm}$ for $\theta_{int} \mathrel{\hat{=}} 10\,\mathrm{dBm}$. The latter is adopted from the IEEE 802.11 implementation for ns-2, since no value is available for wireless sensor nodes.

The two-ray ground propagation model is used for three reasons. It is more realistic than the free-space model and considerably faster than the shadowing model. It also enables a reliable communication, where packet loss can be introduced and is fully controlled by the used bit error model. Hence, simulation with and without packet loss, based on the same propagation model, becomes possible and thus permits an unbiased analysis of the influence of packet loss, as mandated in Section 3.4. The bit error model used for a part of the simulations is that from Equation 2.4. To determine the signal-to-noise (SNR) ratio (the real SINR is not computed as explained in Section 4.1.3), θ_{cs} is used as the noise, since it is the sensitivity of the transceiver. The maximum bit error rate (of a packet received with power θ_{rx} at the edge of R_{com}) evaluates to BER $\approx 8 \cdot 10^{-4}$, which is a realistic value for the TR1001 transceiver.

The layout of the transmitted packets is as follows. Preamble and postamble make up for 11 bytes (according to the Scatterweb platform, version 2.2). Another 10 bytes are consumed by the MAC header. 2 bytes each are used for the sender, receiver, and a checksum (that is not actually calculated). The packet type, the MAC sequence number, and the packet length each consume one byte. Another byte is due to additional flags, e.g., signaling the last packet and if a number of slots to skip is added. If the latter is true, the MAC header is increased by one byte. Hence, the maximum number of slots to skip is limited to 255. The payload of a data packet is 30 bytes, including the address of the source and the data packet sequence number (each 2 bytes). The remainder of the payload, e.g., 26 bytes, is used for sensor readings. The total number of bytes to be transmitted sums up to 51 bytes (52 bytes, if slots have to be skipped). For Type II, up to 4 bytes may be additionally used for piggybacking slots. Standalone packets are allowed to have a maximum number of 34 bytes for transmitting slots. In both cases, an additional byte is required to store the number of actually used bytes for slot encoding. Note that data packets and packets only used for slot transmission have the same maximum size. Acknowledgments have to carry the address of the source, the sequence number of the data packet, and the flags introduced in Section 4.2.4 on page 62. This results in a payload of 5 bytes and thus an overall size of 26 bytes (27 bytes, if slots have to be skipped). Keepalive packets only consist of the pre-/postamble and the MAC header, which includes the type.

A data packet as explained above leads to a packet reception rate of approximately 72%. Here, plain On-Off-Keying without the usage of error correcting codes is assumed. Choosing $r = 10$ assures a probability of successful packet delivery close to 1. This has been verified by simulation and is necessary, as runtime is only comparable, if all data packets are collected by the sink. However, for the simulations without

application of the bit error model, $r = 3$ is selected. This is a proper value in a real deployment, since it would lead to a packet delivery rate above 99%. An appropriate choice of r is required to simulate a realistic delay of slot reuse for Type II.

Timing of the TDMA slot as described in Section 4.2.4 is as follows. The guard interval lasts 2 ms, which is enough to switch on the transceiver and compensate for small clock drifts, i.e., synchronization errors. Turning around the transceiver is model led to take 25 μs. However, processing a received packet and creating an acknowledgments is simulated to take 1 ms, which actually outlasts the turnaround time. The maximum waiting time for packet reception is 5 ms. A parent prolongs this by the guard interval. Adding the guard interval, the packet processing time, and the highest possible propagation delay to the maximum time required for transmitting a packet and an acknowledgment, yields that a slot length of 40 ms is sufficient. However, the actual slot length is not relevant, since runtime T is measured in slots, which are of equal length for all schedules. Note that the maximum overhead of 5 bytes produced by piggybacking slots accounts for 2 ms only. Hence, it can be considered negligible, as simulation results will be afflicted with error of measurements anyway.

The packet history employed by the routing layer can hold up to 10 packets. To evaluate the impact of limited buffers, $B = 200$ and $B = \infty$ are compared. Here, the former is chosen to keep simulation time low in favor of a higher number of simulations, i.e., increase the number of samples to achieve a higher accuracy of the results. The soft limit is fixed at $\tilde{B} = 150$ according to simulational experience: It is possible that a node v_i advises its children to skip slots, but cannot forward any packets while its children are skipping. This occurs, if v_i is also required to skip slots. Hence, when its children restart sending, it is desirable to have space left in the buffer of v_i, so that no packets have to be dropped. This situation has been frequently observed for Type I schedules. Note that $\tilde{B}$ is not effective for $B = \infty$.

Finally, the forwarding strategy for Type II is selected according to the outcome of [TW07b]: Each node v_i always forwards $h_i - 1$ slots before reusing one itself. This strategy is designed to make slots available close to the sink with high priority in order to quickly overcome the bottleneck, i.e., low throughput σ as discussed in Section 2.3.2.

5.1.2 Settings

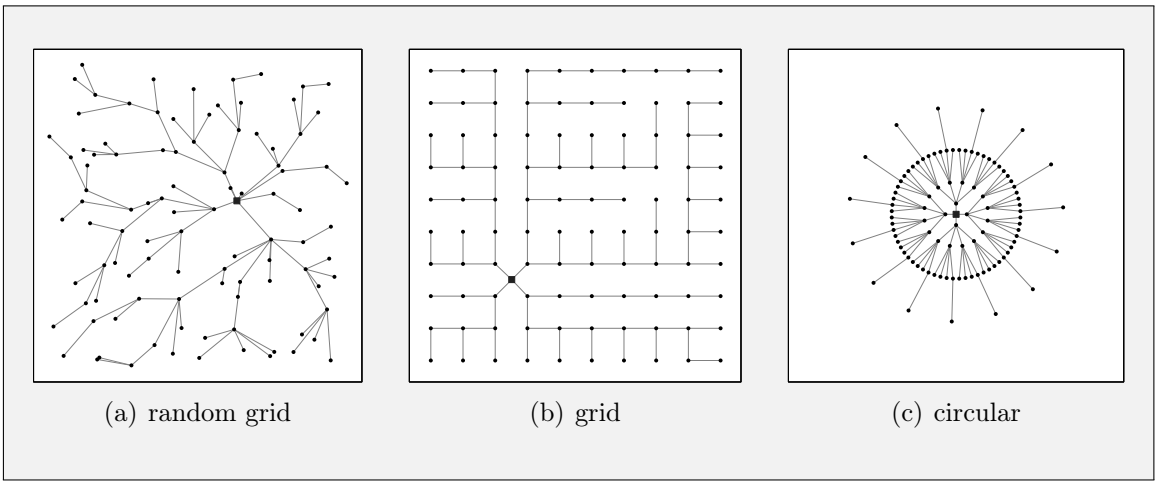
In order to simulate and compare the TDMA schedules presented in Chapter 3 with the parameters identified in Section 3.4.2, suitable simulation settings are created.

Type	ϱ	N	Count	C	L_i
random grid	6	$100, 300, \ldots, 900$	50	$(8), \infty$	$2, 20, 40, 2-40$
random grid	9	$100, 300, \ldots, 900$	50	$8, \infty$	$2, 20, 40, 2-40$
random grid	12	$20, 30, \ldots, 100, 300, \ldots, 900$	50	$8, \infty$	$2, 20, 40, 2-40, 50-150$
random grid	18	$100, 300, \ldots, 900$	50	$8, \infty$	$2, 20, 40, 2-40$
random grid	24	$100, 300, \ldots, 900$	50	$8, \infty$	$2, 20, 40, 2-40, 50-150$
grid	(5)	101	11	(4)	$2, 20, 40, 2-40, 50-150$
circular	—	$20, 30, \ldots, 100, 300, \ldots, 900$	1	$2-8$	$2, 20, 40, 2-40$

Table 5.1: Parameters of the created simulation settings

They are based on three different types of topologies: random grid, grid, and circular. The corresponding setups and their intention are discussed in the following. Table 5.1 gives a summary.

Since network density ϱ is an important characteristic of a network, values from 6 to 24 are considered in the random-grid topologies. These values have been identified empirically and are combined with various network sizes. Small networks ($N < 100$) are only produced with $\varrho = 12$, as density has less impact here. For each set of parameters, 50 topologies are created with the sink placed approximately in the center. Two different trees are built for every topology: one with an unlimited number of children per parent ($C = \infty$) and one with $C = 8$. The latter is a trade-off between ensuring connectivity and saving memory. For $\varrho = 6$, both trees are the same, so that only the ones with $C = \infty$ are used for simulation. An example random-grid topology with $N = 100$ and $C = 8$ is depicted in Figure 5.1(a). The sink is marked by a square, and the tree is indicated by gray edges.



(a) random grid (b) grid (c) circular

Figure 5.1: Topology examples with a corresponding tree

	Type I	Type II	Type III	SPR
Slot ordering	–	↑, ↓	↑, ↓	(↓)
Parameters	CCH 3-hop / $\gamma = 1.9$	–	no λ / $\lambda = 5, 10, 25$	$\kappa = 4, 5, 6$

Table 5.2: Slot assigment parameters

The objective of the grid topologies is to investigate the influence of the sink's position and the number of its children. For this purpose, a grid of 10×10 nodes is created with a horizontal and vertical spacing of 25 m. From this, 11 different topologies with varying position of the sink are derived. The sink is placed on the diagonal between the lower left corner, where it has one child only, and the center, where it has 4 children. From the center on, the sink is placed on the vertical line ending at the bottom of the topology. Except for the lower left and lower centered position, the sink is always placed in the center of its four children. Figure 5.1(b) shows one of these topologies with the sink placed on the diagonal. Particularly note the difference to the random-grid topology and the heavily unbalanced tree.

Circular topologies are created to verify the runtime estimations from Section 3.2.4. Nodes are aligned in circles around the sink, so that balanced, minimum-depth trees with different C result. Only one topology is created for different values of N, since node placement is deterministic. An example with $N = 100$ and $C = 4$ is shown in Figure 5.1(c). It already reveals that these topologies are impractical to validate the estimations for Type I, since density depends on C and N.

For all topologies, different initial buffer fill levels L_i are created, including equally distributed and varying fill levels among nodes. This is an important point, since especially Type III is expected to be sensitive to the fill level. At last, TDMA schedules are created for each combination of topology, tree, and buffer fill levels. Two different Type I slot assignments are created, both based on CCH, as explained in Section 4.2.2. For the interference-free variant, $\gamma = 1.9$ can be derived from R_{com} and the delivery radius calculated by ns-2. It is regarded as CCH $\gamma = 1.9$ throughout this chapter. Slot assignments for (enhanced) Type II and III are produced in both ascending and descending order from leafs to the sink. For Type III, the load-aware extension is used with $\lambda = 10$. For topologies with $\varrho = 12$, $\lambda = 5$ is also applied. In the case of buffer fill levels between 50 and 150 packets, $\lambda = 25$ must be used, since R is likely to become greater than $100,000$, but slots are encoded in 2 bytes. A brief summary of the parameter values is provided in Table 5.2.

5.2 Detailed Simulation Results

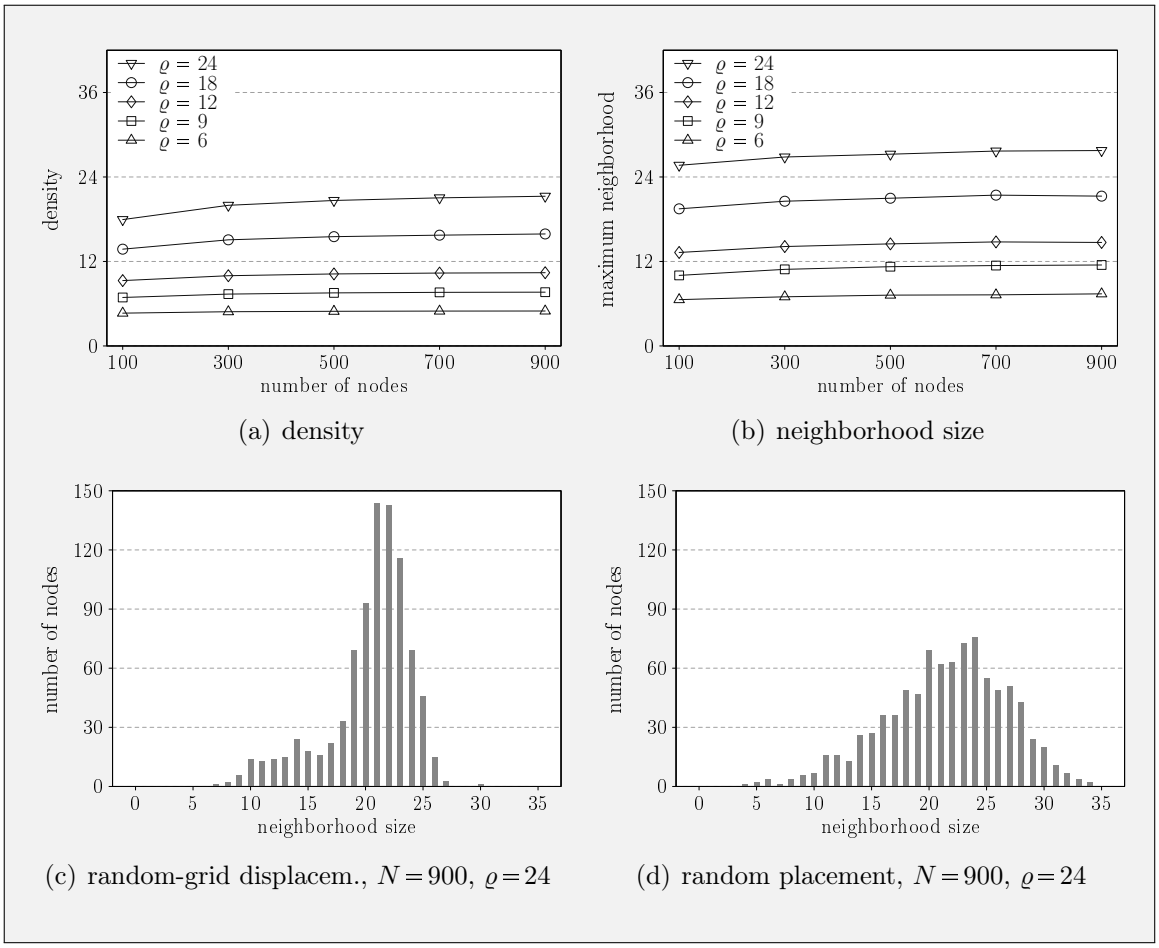
In this section, the simulation results, obtained from more than 400,000 individual simulation runs, are presented. First, the created settings are analyzed. For clearness, a runtime analysis for each of the different TDMA schedules is carried out individually as the second step. A comparison of runtime and energy-efficiency is presented afterwards. Runtime will be referred to as the number of slots per created packet. This simplifies comparison and gives a more compact representation of simulation results. To be consistent, energy-efficiency is also scaled using the overall number of packets, i.e., dividing energy by L_0^*. All runtime and energy figures show the mean of the results obtained from the 50 different topologies each.

5.2.1 Topology and Tree Characteristics

The random-grid topologies and trees created for simulation are analyzed in the following, since their characteristics have an important impact on the expected runtime, as discussed in Chapter 3.

The tool for creating topologies with a specified density has been introduced in Section 4.2.2. Figure 5.2(a) shows the average density of the generated topologies. Apparently, the real densities fall somewhat short as compared to the specified ones, but the relation between them is approximately kept and the standard deviation is at most 0.3. Hence, the specified values will be used to refer to the corresponding topologies. The actually achieved densities are almost independent of the number of nodes. A small decline can be observed for $N = 100$ only. The same observations are true for the average maximum number of nodes inside a communication circle as depicted in Figure 5.2(b). Here, the standard deviation is between 0.4 and 1.3. The reason for the dependency on N is the way the script works. It sets up the spacing of the initial grid so that the expected neighborhood size of a node in the center of the topology meets the specified density. As a result, the actual density falls short, because nodes at the edge have fewer neighbors than those in the center. This has a higher impact in small networks, as they exhibit a larger fraction of nodes close to the edges.

Although the actual densities depend slightly on the network size, the corresponding topologies ensure more equally sized neighborhoods than those created by a completely random placement strategy. This is an important feature, because it reflects the desired close-to-uniform distribution in many data-gathering scenarios. Figures 5.2(c) and 5.2(d) show histograms of the neighborhood sizes found in two



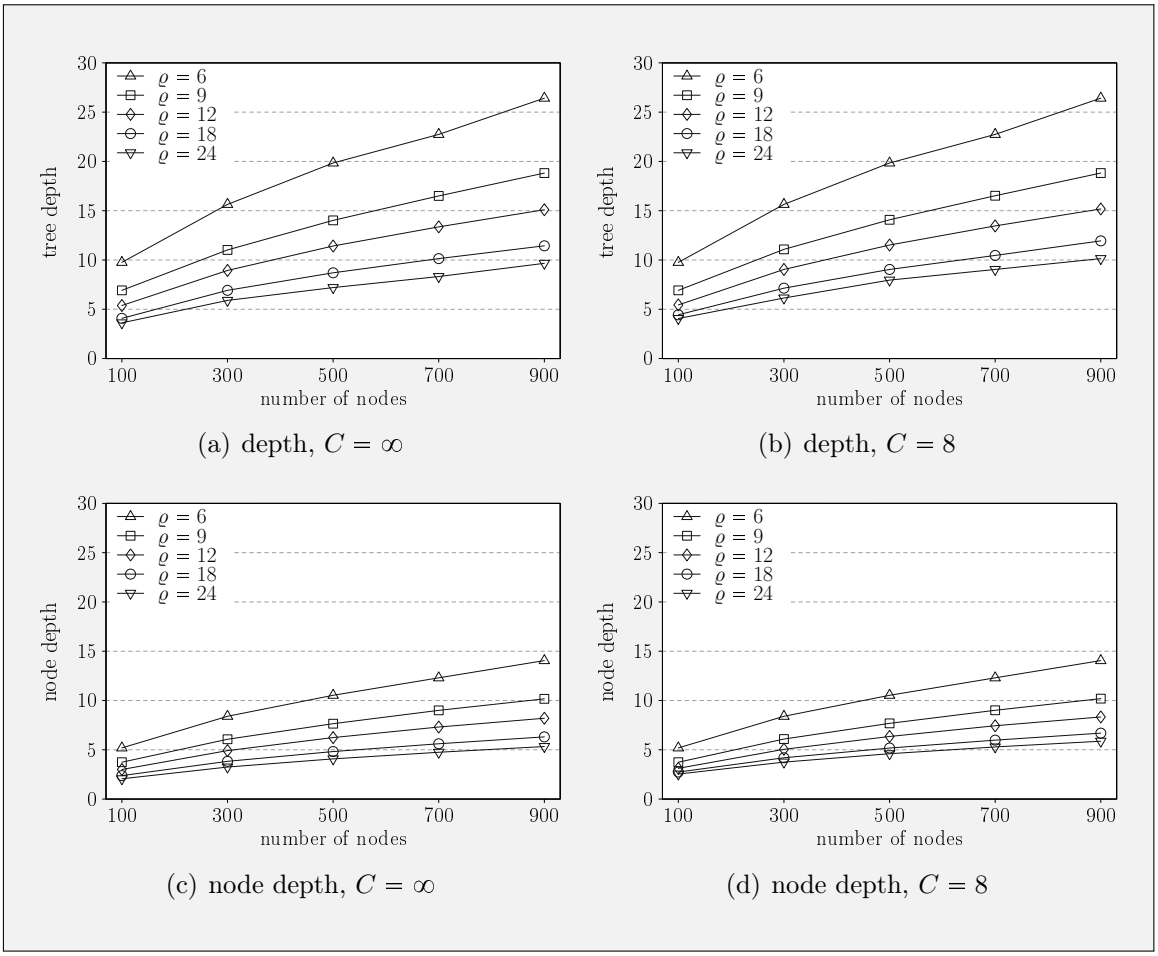
(a) density (b) neighborhood size

(c) random-grid displacem., $N = 900$, $\varrho = 24$ (d) random placement, $N = 900$, $\varrho = 24$

Figure 5.2: Average density and neighborhood size

networks with 900 nodes. The former has been created using random-grid displacement with specified density $\varrho = 24$. The nodes of the second one have been placed using a uniform random distribution. Here, the square area has been chosen in order to achieve a similar overall density as for the first one. The grid-based solution produces a considerably smaller deviation. 58% of all nodes have between 21 and 25 neighbors as opposed to 37% in the completely random topology.

The significant characteristics of the trees built upon the just analyzed topologies are their depth and the number of leafs. Both influence the number of slots created and the runtime achieved by the TDMA schedules under consideration. Figures 5.3(a) and 5.3(b) show that there is almost no increase in tree depth h, if the maximum number of children is restricted to $C = 8$. As a matter of fact, minimum depth $h^\star$ as discussed in Section 3.2.1 is not achieved. This is no surprise, since the minimum depth actually depends on the maximum distance between a node of the network and the sink (cf. Section 2.2.2). In addition, the average depth of individual nodes as



(a) depth, $C = \infty$

(b) depth, $C = 8$

(c) node depth, $C = \infty$

(d) node depth, $C = 8$

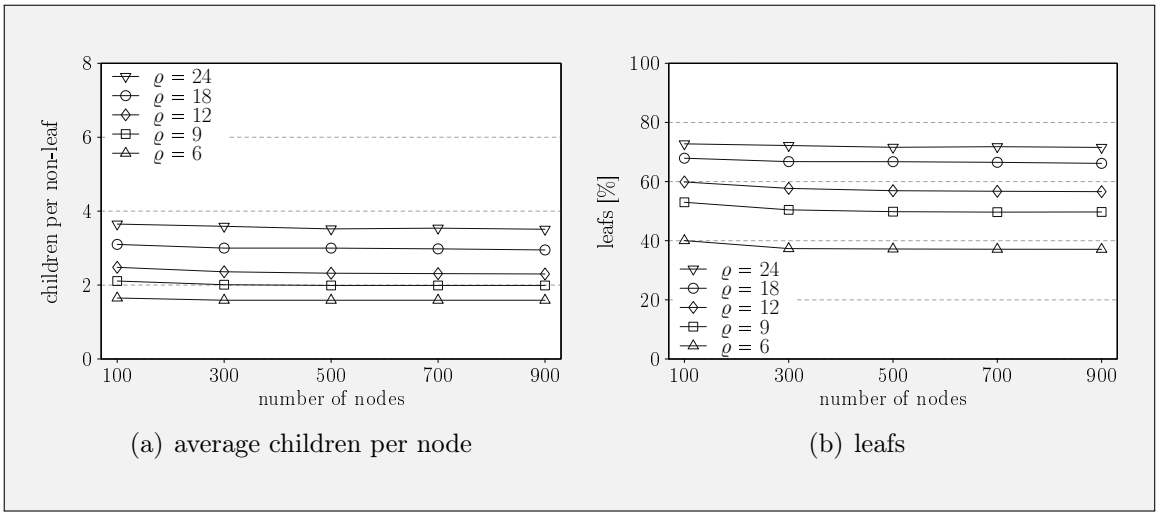
Figure 5.3: Average tree and node depth

shown in Figures 5.3(c) and 5.3(d) is hardly increased by limiting $C = 8$. The highest difference is observed for $\varrho = 24$. In the case of $N = 900$, the average node depth increases from 5.33 by approximately 10% to 5.87.

The just addressed average node depth gives an estimate of minimum runtime, if slots are not spatially reused and initial fill levels do not vary among nodes. As this runtime is the same as the lower bound for Type II and Type III, divided by the network load L_0^*, it follows from Equations 3.9 and 3.10 the minimum average runtime (in slots) per packet.

$$T_{min} = \sum_{v_i \in \mathcal{V}} L\, h_i \quad \Longrightarrow \quad \overline{T}_{min} = \frac{T_{min}}{(N-1)\, L} = \frac{1}{N-1} \sum_{v_i \in \mathcal{V}} h_i \tag{5.1}$$

This result is of benefit in order to evaluate runtime performance of the different TDMA schedules.



(a) average children per node

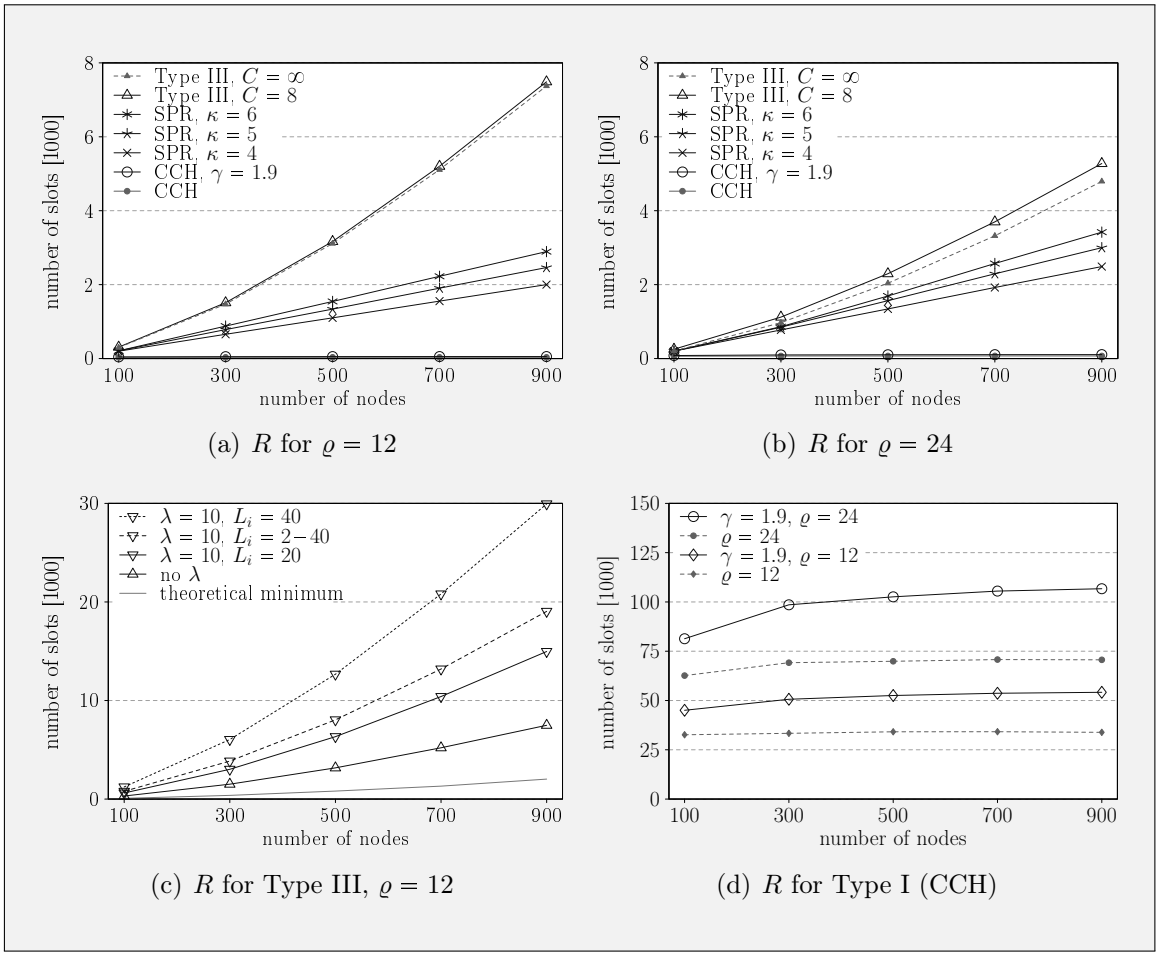
(b) leafs

■ **Figure 5.4:** Average number of children and leafs for $C = 8$

The number of children per node influences runtime as described in Section 3.2.4. The average number per non-leaf is depicted in Figure 5.4(a). Again, node density has a considerable influence. As outlined in Section 2.2.2, the average stays well below the possible maximum, so that the restriction of $C = 8$ does not have a great influence on this aspect. Yet, $C = 8$ restricts the possible number of children of the sink particularly in dense networks, where $C = \infty$ produces values of C_0 as large as 22. However, the impact is expected to be small, because many of those additional children are either leafs or have small subtrees. The fraction of leafs, shown in Figure 5.4(b), is almost independent of N. This finding is closely correlated with the average number of children, which can be derived from Equation 3.19. Note that this fraction stays almost unchanged for $C = \infty$, so that it is not displayed.

5.2.2 Number of Slots

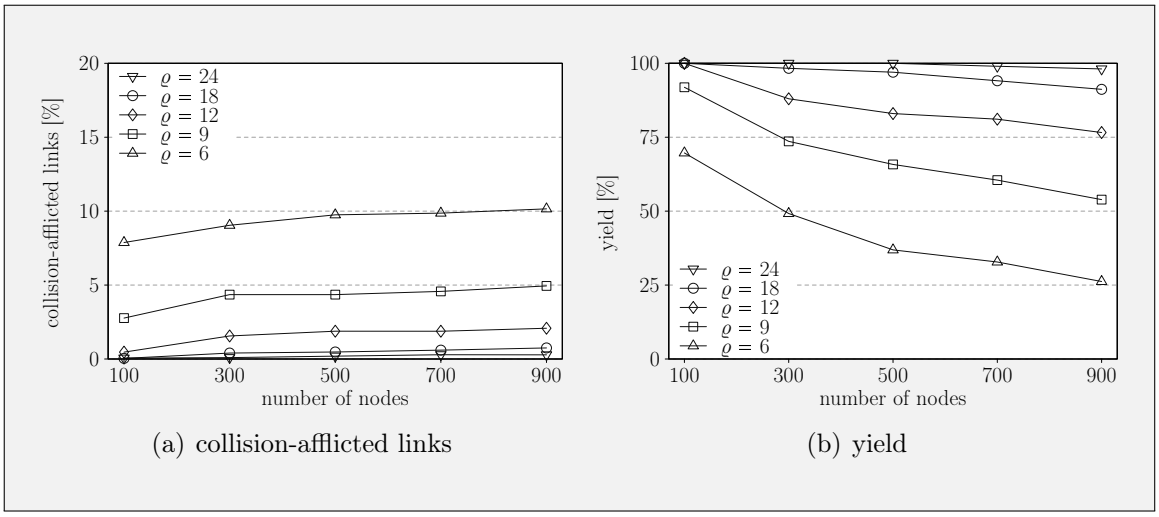
In Chapter 3, lower and upper bounds for the number of slots produced by the different slot assignments have been estimated. Figures 5.5(a) and 5.5(b) show the average number of actually produced slots for $C = 8$ and densities $\varrho = 12$ and $\varrho = 24$, respectively. They disclose that SPR produces more slots with increasing density, which is caused by the higher fraction of leafs (cf. Figure 5.4(b)). With fixed density, the number of slots grows linearly with N. Hence, the behavior as formulated in Equation 3.19 on page 43 appears to be valid, although the upper bound is not reached.



■ **Figure 5.5:** Average number of slots for $C = 8$

Since density influences the average node depth of the generated trees, it also affects the number of slots R_{III} produced by Type III. This is explained by Equation 3.5. $C = 8$ leads to a considerable increase for $\varrho = 24$ only. Figure 5.5(c) gives a detailed overview about R_{III}. It exhibits that plain Type III produces a multiple of the estimated minimum of slots given by Equation 3.7, but the gradients are proportional. Hence, the estimation gives a good approximation of the general behavior. The load-aware variant of Type III produces just a multiple of the plain version, as discussed in Section 3.2.2 on page 31.

The detailed look at CCH in Figure 5.5(d) reveals that the round length is actually proportional to the density, as estimated in Section 3.2.2 on page 29. This also explains the dip for $N = 100$. The interference-free variant CCH $\gamma = 1.9$ increases the overall number of slots with a factor between 1.1 and 1.7. This factor is generally smaller in dense and in small networks. Note that the limitation of $C = 8$ has almost no influence on R_{I} and is therefore not displayed.



(a) collision-afflicted links (b) yield

■ **Figure 5.6:** Percentile of collision-afflicted links and yield for CCH 3-hop, $C = 8$

5.2.3 Type I

Type I schedules are generally collision-afflicted, so that this issue requires analysis. Figure 5.6(a) depicts the number of links experiencing collisions for different network sizes and densities. Up to 10% of all links suffer from collisions in a network with density $\varrho = 6$. This high value comes from the fact, that the exchange of k-hop information is particularly insufficient in sparse networks: Two nodes may just be outside communication range and there is no k-hop connection between them. Note that $k = 3$ for our implementation. The percentile of collision-afflicted links approaches zero for large values of ϱ. Yet, 0% is achieved in small networks only, because of the declining likelihood of nodes (or links) being assigned the same slot. This is true, because R_{I} is just influenced by the density.

As can be obtained from Figure 5.6(b), the yield (cf. Section 3.4.1) is severely affected. Even for $N = 900$ and $\varrho = 12$, where less than 3% of all links are collision-afflicted, it is as low as 76%. This can be explained as follows. If a link suffers from a collision, the latter will occur every time that link is used, because each link is assigned just one slot. Hence, a collision-afflicted link leads to a link failure, which provokes the disconnection of the complete subtree and the collection of its data becomes impossible. This insight and the results of Figure 5.6(b) render a runtime evaluation for CCH bootless. In consequence, evaluation will be restricted to the collision-free variant CCH $\gamma = 1.9$.

Runtime of Type I depends on the round length R and the largest subtree of the sink (cf. Section 3.2.4). The latter is found to grow almost linearly with N for

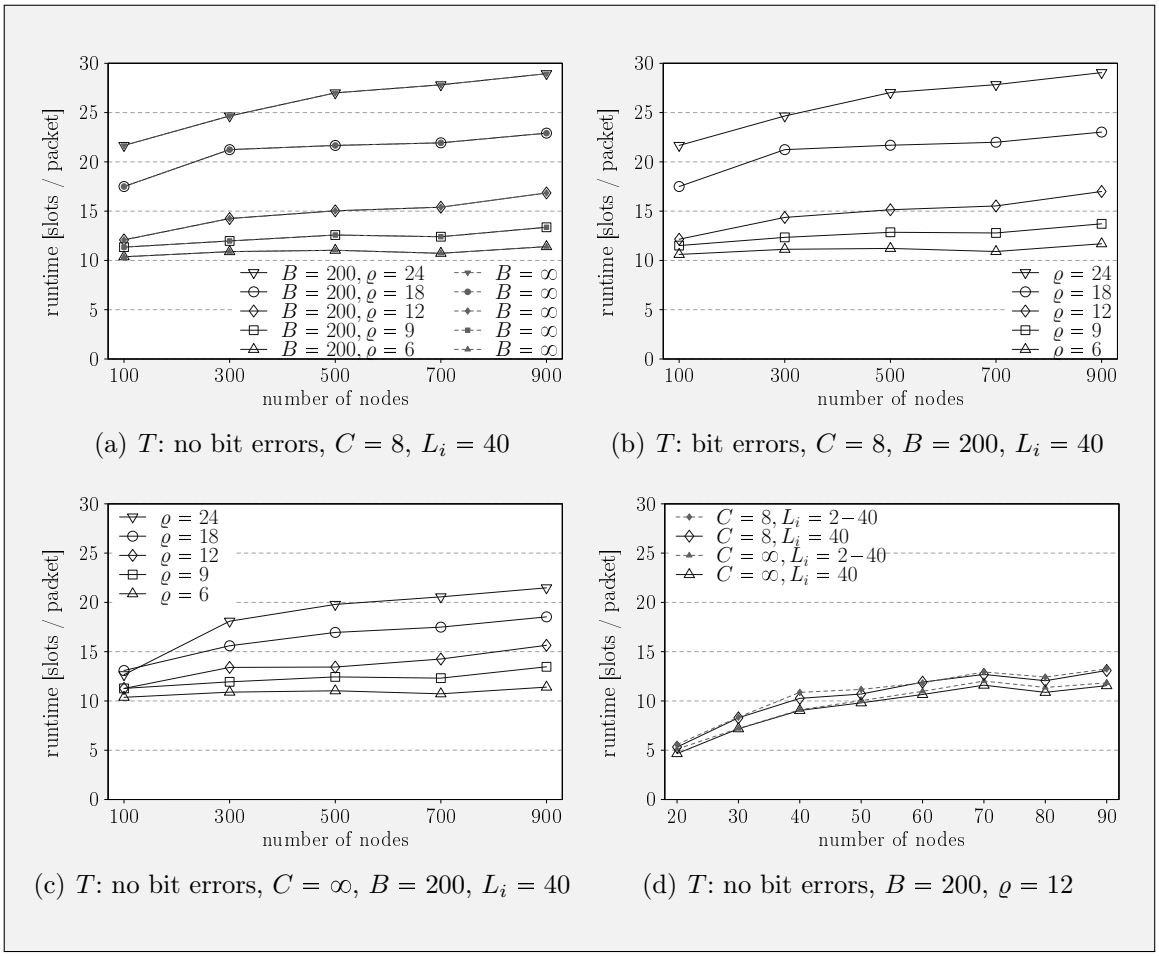
the constructed trees. Hence, runtime is affected by the round length mainly, which is supported by Figure 5.7(a). It depicts the runtime of CCH $\gamma = 1.9$ for different network densities, $C = 8$, and equally distributed buffer fill levels in the absence of bit errors. The relation of densities is reflected by the runtime. The figure also shows that runtime is not influenced by the buffer size B. The small dip in runtime for $N = 700$ is caused by slightly smaller largest subtrees. This appears to be due to the limited size of topology and tree samples. In addition, the grid topology with varying sink positions approves Equation 3.8 up to a scale factor, which is the ratio between round length and density.

As depicted in Figure 5.7(b) and argued in Section 3.2.6, packet loss increases runtime scarcely. Since the children of the sink can be considered the bottleneck, solely their links to the sink may slow down runtime in the case of lost packets. However, these children are comparably close to the sink—this is ensured during tree construction—, so that packet loss is rare. This is true, because the bit error rate depends on signal strength, which gains with closeness (cf. Section 2.1.3).

The limitation of $C = 8$ increases runtime of CCH $\gamma = 1.9$, as illustrated by Figure 5.7(c) with $C = \infty$. The effect of a limited C is larger for greater densities, but is more distinct than suggested by the corresponding change of average node depth. The explanation of this is found in Section 5.2.1 on page 81. In dense networks, the sink is able to claim more children. Although the corresponding subtrees are quite small and the overall tree becomes unbalanced, the size of the largest subtree is reduced. Thus, runtime decreases.

Figure 5.7(d) gives information about runtime in small networks. Even here, a small influence of C is observable. In contrast, the graph shows that there is almost no difference in runtime between varying and equally distributed buffer fill levels. For small N, runtime is comparably low. This is caused by the small round length. Note that the round length actually increases from an average of 19 for $N = 20$ to 37 for $N = 40$. The dip at $N = 80$ is due to the variation of largest subtrees again.

The findings in this section support the theoretical analysis. In particular, the round length is generally proportional to the observed density. Thus, runtime depends on the density mainly, since subtrees grow almost linearly with N. Although a limitation of C influences runtime, its impact is smaller than suggested by Equation 3.8. Finally, the size of B does not play a role, nor does packet loss pose a severe hazard, if the bit error rate stays comparably low. Most importantly, it shows that the 3-hop approach is incapable of producing collision-free schedules, which heavily impairs yield. CCH $\gamma = 1.9$ solves this problem, but cannot be implemented distributedly.



(a) T: no bit errors, $C = 8$, $L_i = 40$

(b) T: bit errors, $C = 8$, $B = 200$, $L_i = 40$

(c) T: no bit errors, $C = \infty$, $B = 200$, $L_i = 40$

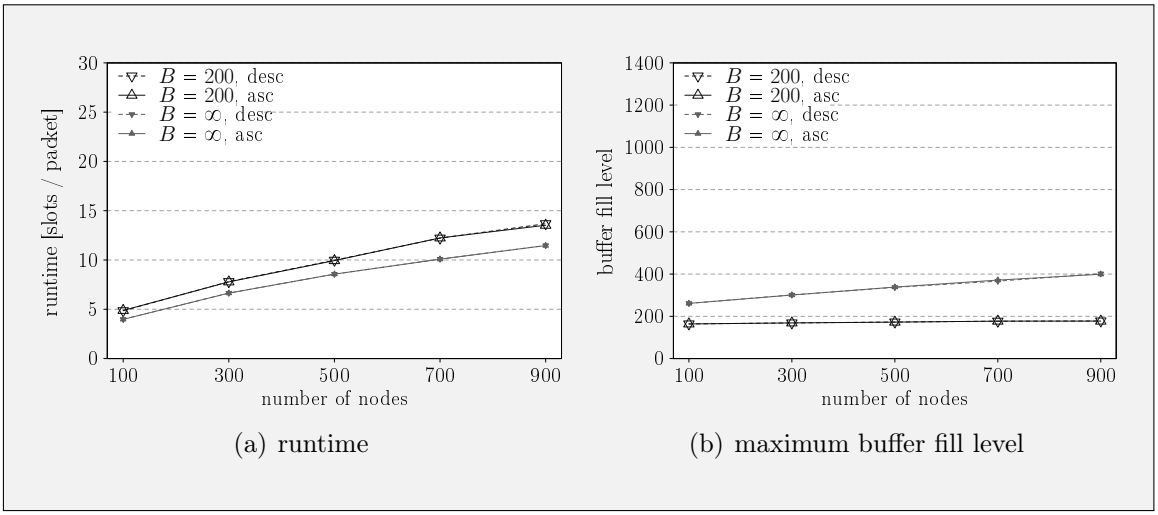
(d) T: no bit errors, $B = 200$, $\varrho = 12$

Figure 5.7: Type I CCH $\gamma = 1.9$ runtime

5.2.4 Type *II* Enhanced

The first question regarding (enhanced) Type II concerns the slot ordering as described in Section 2.3.2 on page 23. The answer is presented in Figure 5.8(a): Both variants achieve the same runtime. This observation holds for equal and variable buffer fill levels, if the L_i are well above the number of possible retransmissions r. If this is not the case, slots are sparsely reused and the ascending order is faster. The explanation is according to the consideration in Section 3.2.4 on page 33 for Type III. As a result, it can be expected that Type II will achieve the same runtime even without explicit slot assignment, i.e., every node uses its unique identifier to determine its sending slot. Besides tree construction, no extra effort is required then, as a node can tell the slots of its children by their identifier, too.

Unlike Type I, the limited buffer increases runtime of Type II. The explanation is as follows. If buffers are limited, they fill up quickly on nodes close to the sink, because



(a) runtime

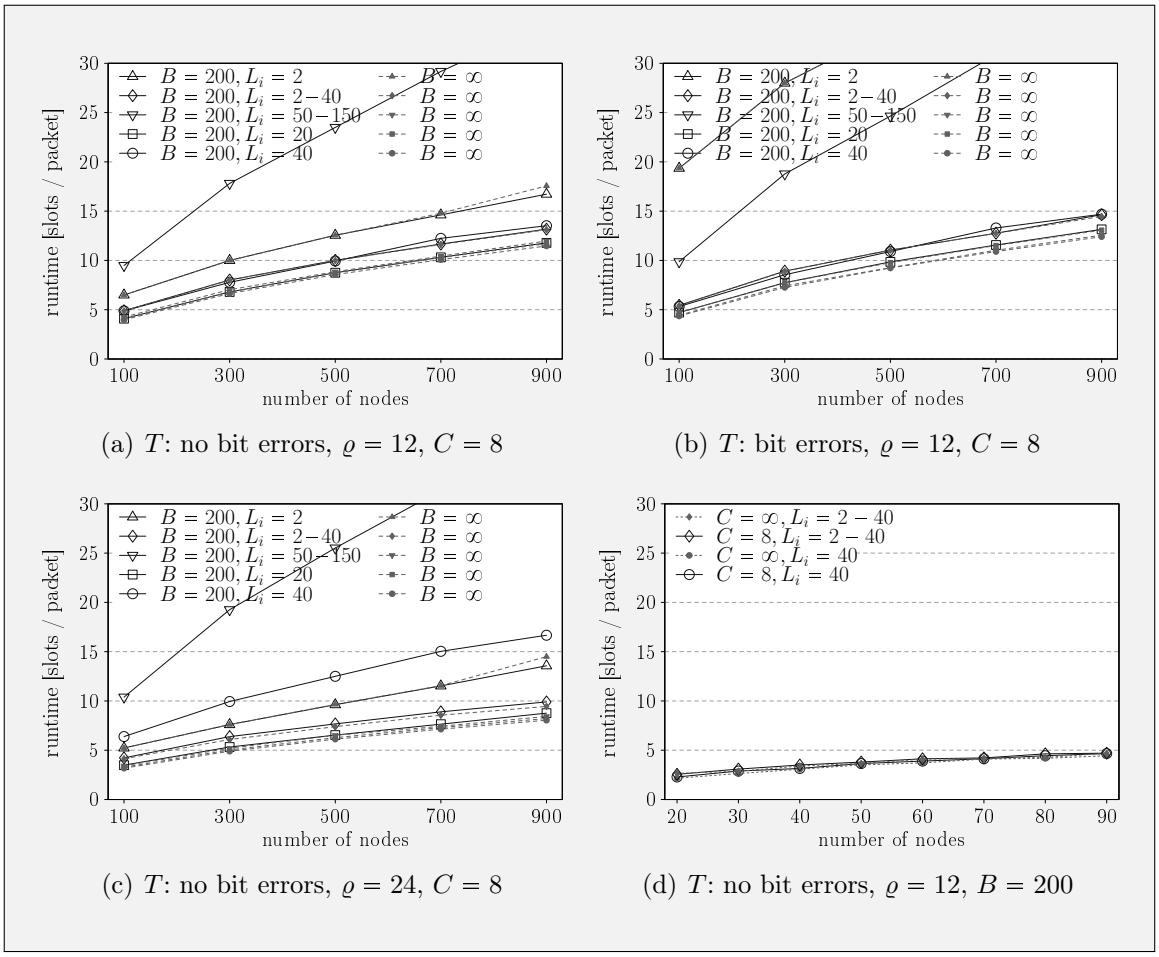
(b) maximum buffer fill level

Figure 5.8: Type II, influence of slot ordering for $C = 8$, $\varrho = 12$, $L_i = 2-40$

they have more incoming than outgoing packets (cf. Figure 2.3 on page 14). Flow control then requires the children of these nodes to wait, resulting in an increased likelihood of buffer overflow on those children. This procedure spreads in opposite direction of packet flow. In conclusion, the average time a node needs to forward all data of its subtree escalates. As a result, reusing slots is delayed and runtime increases. This problem is aggravated in larger networks, since the load of each node is increased. Figure 5.8(b) depicts the actually observed maximum buffer fill level for $B = \infty$. It shows that $\tilde{B}$ has been always exceed for $B = 200$. The graph for $B = \infty$ indicates the minimum required buffer size to avoid the just explained runtime threat.

Runtime of Type II is sensitive to the initial buffer fill levels, as shown in Figure 5.9(a). For high loads, e.g., $L_i = 50-150$, runtime is heavily elevated. As stated in the preceding paragraph, this is due to buffer congestion and the hereby caused delayed slot reuse. This is substantiated by the behavior in case of unlimited buffers. Note here, that runtime is close to the optimum as indicated by the average node depth and behaves qualitatively as estimated in Section 3.2.4 on page 32. Although the results do not reveal a detailed relation between initial fill level and runtime, it can be inferred that Type II performs slightly better in the presence of initial buffer fill levels of equal size. This is presumably related to later slot reuse in subtrees with higher load. However, in a realistic setup, leafs can be expected to have less packets left from a previous collection phase than nodes close to the sink. In this situation, Type II performs better, as slots from leafs can be reused earlier than in a completely random distribution of fill levels.

Bit errors, see Figure 5.9(b), have a noticeable linear influence on runtime, which



(a) T: no bit errors, $\varrho = 12$, $C = 8$

(b) T: bit errors, $\varrho = 12$, $C = 8$

(c) T: no bit errors, $\varrho = 24$, $C = 8$

(d) T: no bit errors, $\varrho = 12$, $B = 200$

Figure 5.9: Type II runtime

is larger as compared to Type I. Yet, this is partly due to the increased number of retransmission r for ensuring connectivity: Reusing slots of a node imposes a delay of r slots (cf. Section 4.2.5 on page 63). Caused by lower average node depth, runtime decreases with growing density. The improvement, shown in Figure 5.9(c), is according to the corresponding graphs in Figure 5.3(d) on page 80 for medium L.

In contrast to Type I, C does not considerably impair runtime in case of limited buffers, which is shown for small networks in Figure 5.9(d). This observation has two reasons. Firstly, larger subtrees are assigned more slots, so that reusing them compensates for the higher load. Secondly, a larger number of children may lead to fast buffer congestion. In turn, slots can be reused later.

Runtime analysis as conducted in Section 3.2.4 comes close to the observed behavior of Type II during simulation. The minimum-tree experiments approve its correctness. Yet, it stays open, if a different reuse strategy may improve performance. Type II is very fast in small networks, but runtime performance in large networks is reduced

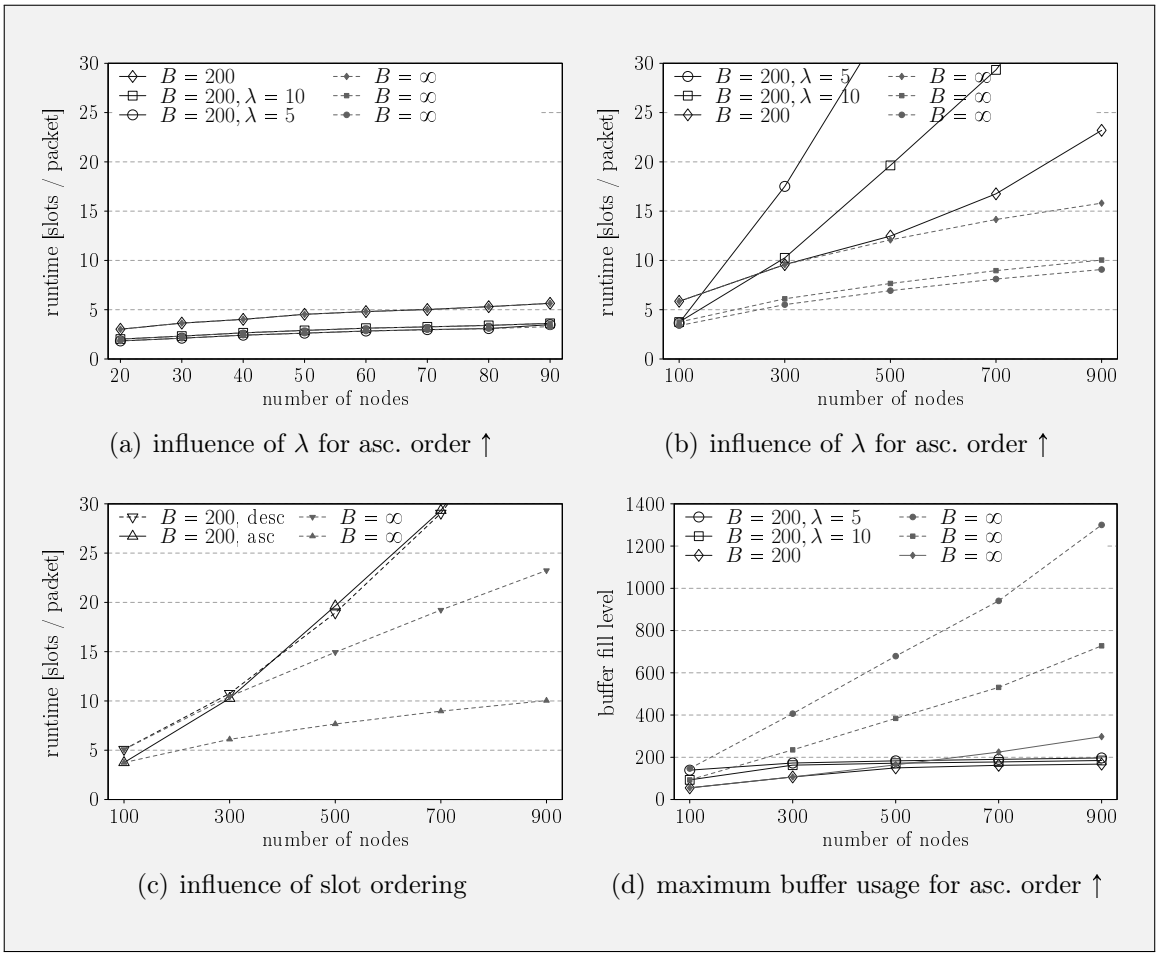
by buffer limitations. In addition, runtime is affected by initial buffer fill levels. This includes both equally distributed and varying fill levels among nodes. Supported by the grid experiment, an important characteristic of Type II is that the position of the sink is important. A centered sink leads to a lower average node depth and thus better runtime.

5.2.5 Type *III*

Two variants of Type III have been introduced in Section 2.3.2 on page 25: the plain and the load-aware one. Runtime of both is compared in Figures 5.10(a) and 5.10(b) for variable initial buffer fill levels. As expected, the load-aware approach is faster, because slots are assigned depending on individual node loads. A smaller λ allows for better adaptation to the actual load and reduces runtime further. Yet, the smaller λ becomes (and the higher the L_i get), the more slots are produced. At some point, a subset of nodes is alloted a set of (consecutive) slots exceeding B. In this case, runtime is dramatically elevated, because a node cannot use all of these slots for sending.

Figure 5.10(b) also shows the theoretical runtime performance for unlimited buffers. However, this is not a realistic situation, so that the apprehension from Section 3.2.5 is justified. In consequence, the load-aware approach is suitable only for small networks with low load. Figure 5.10(d) reveals the actually required buffer sizes in the given scenario in order to achieve best possible runtime for Type III. Another aspect on the same matter is that of flow control, which may increase runtime unintentionally. The current strategy causes early waiting advices, which is not suitable for Type III. If the sum of slots assigned to a node's children is above $\tilde{B}$, at least one child cannot use all of its slots. This is true, because that node receives more than $\tilde{B}$ packets per round. As it cannot forward packets before having receiving packets from all of its children, it will advise at least one child to wait for the remainder of the current round. Furthermore, in the case of buffer congestion, subtrees are not emptied with the same pace. This implies, that some subtrees will have forwarded all data to the sink, whereas others still store a huge amount of data. To collect these packets, the slots of the empty subtree cannot be used and thus prolong runtime.

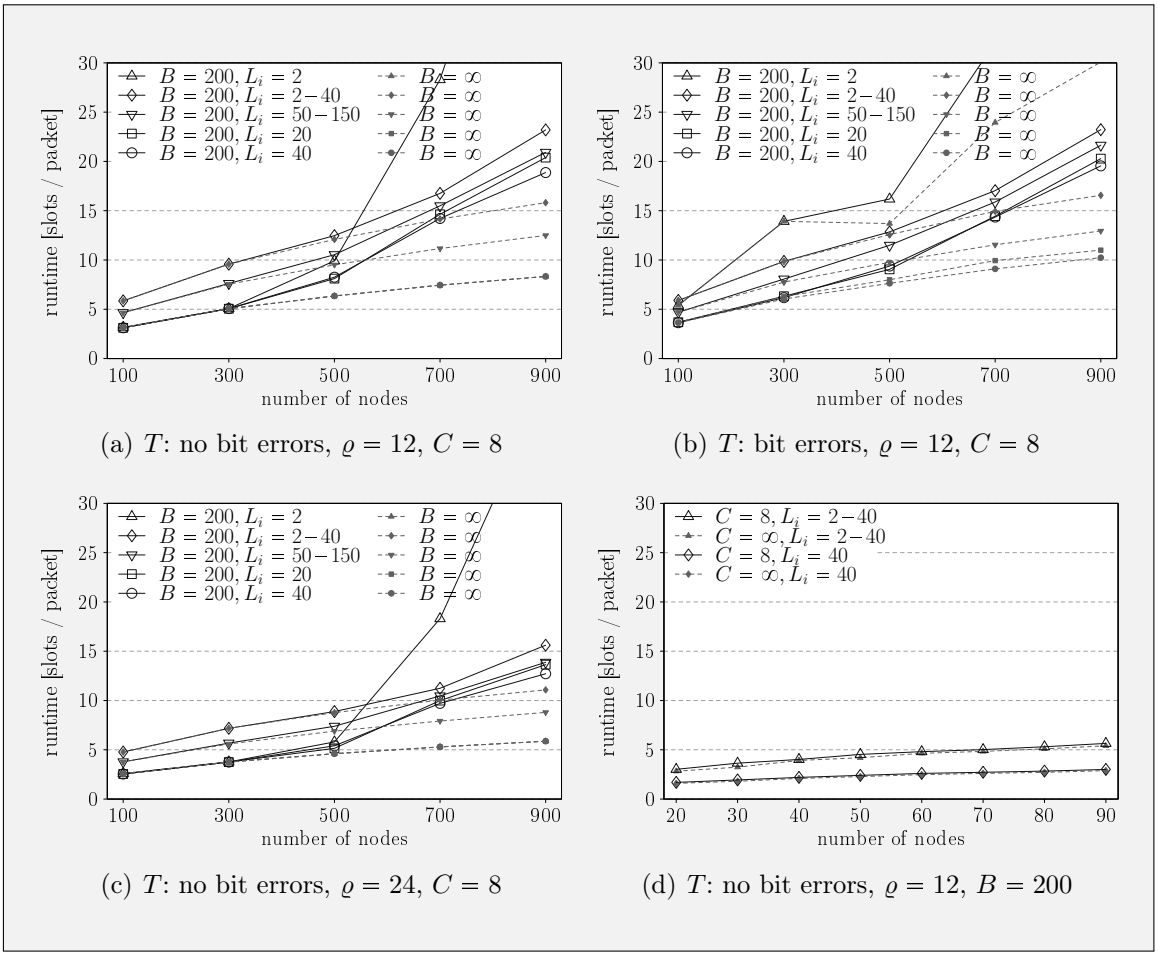
In contrast to the results for Type II, slot ordering has a decisive influence on runtime performance of Type III. According to the analysis in Section 3.2.4, Figure 5.10(c) shows that the ascending variant is generally superior. This is substantiated by the results for $B = \infty$. In contrast, both variants reach an almost equal, poor runtime, when suffering from buffer overflow. In the remainder of this section, only



■ **Figure 5.10:** Type III, influence of load-awareness factor λ and the ordering of slots, $\varrho = 12$, $C = 8$, $L_i = 2-40$, no bit errors

the ascending variant will be looked at, since it is faster, when buffer limitations do not have an impact.

In the following, runtime of the plain Type III schedule is analyzed. Figure 5.11(a) shows runtime performance for different initial buffer fill levels. As mentioned above, the limited buffer size leads to a significantly higher runtime per packet as soon as buffer congestion occurs. This depends on the network size and average tree depth, as this influences the number of slots per node (cf. Section 3.2.2 on page 30). For $\varrho = 12$, buffer limitations start to become an issue in the region of 300 to 500 nodes. As runtime is scaled by the number of packets, it is particularly affected for small initial fill levels. The plot also shows that the highest initial fill level determines overall runtime. The time required per packet is almost doubled from $L_i = 40$ to $L_i = 2{-}40$. As the second can be expected to give approximately half of the overall load of the first, this implies that absolute runtime is in the same order of magnitude. An



(a) T: no bit errors, $\varrho = 12$, $C = 8$

(b) T: bit errors, $\varrho = 12$, $C = 8$

(c) T: no bit errors, $\varrho = 24$, $C = 8$

(d) T: no bit errors, $\varrho = 12$, $B = 200$

Figure 5.11: Plain Type III runtime

important aspect on this matter is the actual distribution of the L_i. If leafs have high values of L_i, runtime is severely affected, as leafs are assigned just one slot. In contrast, inner nodes have more slots at their disposal, so that a large L_i has less influence. This explains why $L_i = 50{-}150$ has better per-packet runtime performance than $L_i = 2{-}40$. In the first case, the probability of a leaf node being assigned the upper limit is less than in the second case. Hence, if dividing by the overall network load, the expectancy of the runtime per packet for $L_i = 2{-}40$ is higher. In conclusion, Type III does not perform well in the presence of variable fill levels.

The influence of packet loss is depicted in Figure 5.11(b). As expected in the analysis in Section 3.2.6 on page 35, it is comparably high. In the case of unlimited buffers, $L_i = 40$, and $N = 900$, runtime is prolonged by almost 23%. For $N = 300$, where the buffer size is just sufficient, runtime is still lifted by more than 19%. Yet, the impact of packet loss is less severe for variable initial fill levels. Here, some nodes usually waste slots in the last rounds, and those slots can be used to send delayed

packets, where the delay is caused by packet loss.

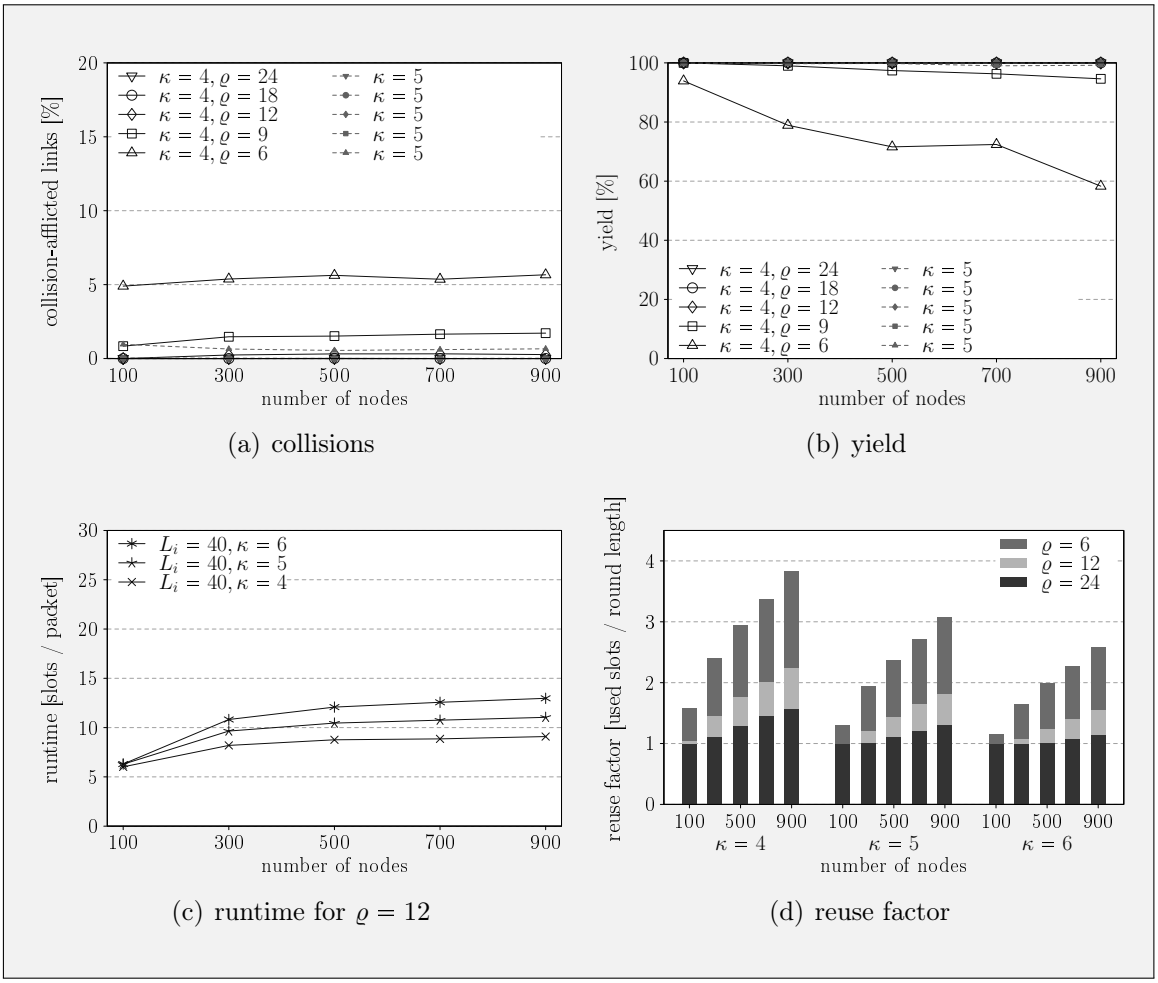
Since average node depths decreases in dense networks, runtime improves. Moreover, for equal N the effect of buffer limitation is alleviated, as less slots are generated. The corresponding plots are depicted in Figure 5.11(c). The minimum-tree and grid experiments support these results. They additionally imply that runtime depends on the position of the sink. The center of the network is the favorable position, since tree and node depth are minimized then.

With $C = 8$ runtime is increased by more than 20% for some simulations of large networks with $\varrho = 24$. However, small networks with density $\varrho = 12$ show almost no sensitivity to the choice of C, as is outlined by Figure 5.11(d). This is relevant, as Type III has its strengths in small networks, since buffer limitations do not pose a threat on runtime here.

The simulation results generally confirm the theoretical analysis. As expected, the ascending variant achieves lower runtime than the descending one. Type III is fast in small and medium networks, but severely suffers from limited buffers in large networks. This is even more dramatic for the load-aware variant, which is applicable only in small networks. Here, it performs well for variable loads. The buffering problem depends on ϱ and C, as these two parameters influence the average node depth. A smaller value for the latter is desirable. Furthermore, runtime of Type III is elevated by packet loss.

5.2.6 SPR

SPR reuses slots an paths, so that intra-path collisions are possible. Figure 5.12(a) shows the percentile of collision-afflicted links by network size for $\kappa = 4$ and $\kappa = 5$. The number of links suffering from collisions depends on the density and partly on C, which is not depicted. If $C \ll \varrho$, nodes on a single path will be closer together, so that interference occurs for small values of κ. Hence, κ must be chosen carefully in context of network density and C. However, this problem occurs close at the sink exclusively (cf. Section 2.2.2). The plot infers that $\kappa = 4$ produces a considerable amount of collisions, whereas $\kappa = 5$ is capable of generating collision-free schedules in dense networks. The yield depicted in Figure 5.12(b) is high in comparison with 3-hop CCH, which can be explained as follows. Nodes close to the sink have many slots at their disposal, which belong to different paths. A node close to the sink and its subtree may only get disconnected from the network, if consecutive slots, and thus



(a) collisions (b) yield

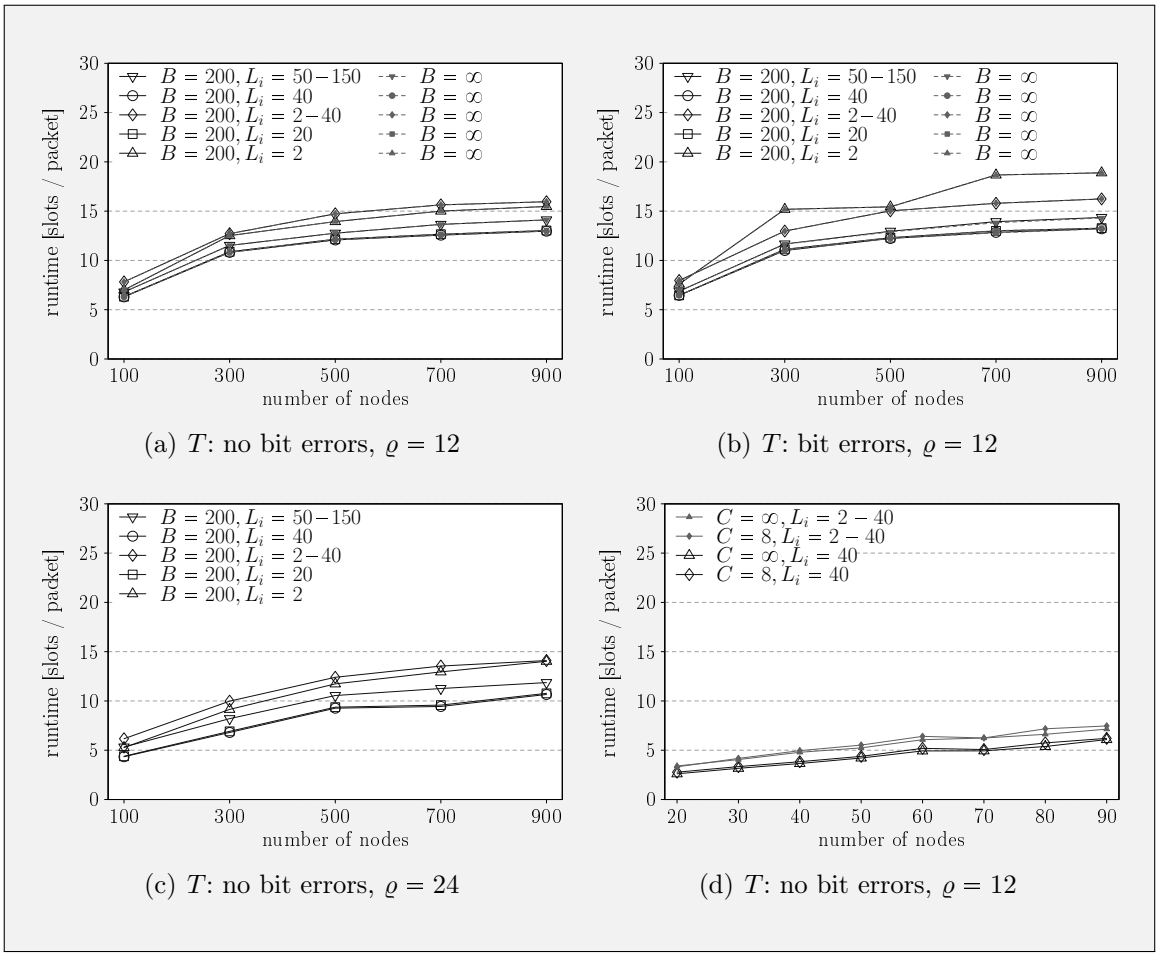
(c) runtime for $\varrho = 12$ (d) reuse factor

Figure 5.12: SPR, influence of κ for $C = 8$, $B = 200$, and no bit errors

paths, of that node are suffering from collisions. Hence, the disconnection of large subtrees is less likely in comparison with Type I.

Further investigation of the simulation results reveals, that for SPR $\kappa = 5$ and $\kappa = 6$ no interrupted links are observed for densities $\varrho \geqslant 12$. For $\varrho = 6$, SPR $\kappa = 6$ is still able to collect all data from the network in more than 75% of all simulation runs. Hence, a runtime analysis based on all successful simulation runs is possible and legitimate. Note that it is not surprising, that sparse networks increase the likelihood of collision, as voids can be more frequently observed. This has been discussed in Section 3.3.1 on page 38 and also in Section 5.2.3 on page 83.

The influence of κ on runtime is displayed in Figure 5.12(c). The differences are correlated with the slot reuse factor in Figure 5.12(d). For $N = 100$ and $\varrho = 12$, no slots are reused, so that the overall number of slots is the same. Hence, runtime is equal, too. For the other network sizes, runtime is inversely proportional to the



(a) T: no bit errors, $\varrho = 12$

(b) T: bit errors, $\varrho = 12$

(c) T: no bit errors, $\varrho = 24$

(d) T: no bit errors, $\varrho = 12$

■ **Figure 5.13:** SPR runtime, $C = 8$

reuse factor. Note that the same observation is true for the overall number slots (cf. Section 5.2.2). While high slot reuse may lead to collisions (in sparse networks), it is an indicator for low runtime, as slots are spatially reused, which already supports Equation 3.20 on page 43.

Runtime of SPR $\kappa = 6$ is shown in Figure5.13(a). It is not influenced by buffer limitation, as discussed in Section 3.3.1 on page 38. In fact, buffer overflows have not been observed. Moreover, SPR gives good results for varying buffer fill levels. However, the per-packet runtime is higher for variable loads, but just slightly for $L_i = 50-150$. The same explanation as given for Type III on that matter is valid here. As indicated by the slot reuse statistic, (per-packet) runtime stays almost constant in large networks. The same results are provided by the minimum-tree experiment. This proves the limiting effect of κ (cf. Equation 3.20).

The effect of bit errors, shown in Figure 5.13(b), is notably low. This is a striking advantage over Type III. The comparably high impact for $L_i = 2$ is caused by the

small number of packets. Here, a single lost packet has higher influence on runtime.

Figure 5.13(c) shows that SPR becomes faster in dense networks, which is due to the decreased average node depth. Note that at the same time, SPR produces a higher amount of slots. This does not appear to have a negative effect on runtime. Because slots are scarcely reused for $\varrho = 24$, per-packet runtime increases with growing N. For larger networks, this increase can be expected to almost vanish.

Runtime in small networks, see Figure 5.13(d), is well above the possible minimum as given by the average node depth. This is caused by the fact, that slots are not spatially reused in these networks, so that the advantage of SPR over Type II and Type III does not come into operation. In addition, this implies that many slots are wasted, if nodes with high depth have sent all packets of their subtree, but nodes close to the sink still have to forward data. Although slots are wasted in large networks, too, if most nodes have completed forwarding all data, its influence is comparably low due to the huge amount of packets. Finally, it shows that limiting C does not impair runtime.

The simulations for the grid topology and varying sink positions show that SPR produces the fastest schedules, if the sink is placed on the edge. Here, SPR profits especially from frequent slot reuse, because the tree has high depth. Runtime with a centered sink is still high due to the balanced tree. Intermediate positions lead to an increased runtime, if the tree is heavily unbalanced. The explanation of this is according to the preceding paragraph: slots are numerously wasted at the end of the collection phase.

In conclusion, SPR generally fulfills the expectations and design goals presented in Section 3.3. It is fast for variable buffer fill levels and robust against packet loss. Buffer utilization is as low as expected, which is an important key feature in data-gathering. The grid experiment reveals that SPR performs particularly well, compared to the other schedules, if the sink is placed at the edge of even small sized networks. Yet, there are two drawbacks. SPR is relatively slow in small networks with a centered sink, and κ must be chosen appropriately with regard to network density in order to avoid collisions. However, $\kappa = 6$ has been shown to give collision-free schedules in networks with $9 \leqslant \varrho \leqslant 24$.

5.3 Comparison

After the detailed runtime discussion of the different TDMA schedules, a concluding comparison is provided. Besides runtime, energy-efficiency is particularly addressed,

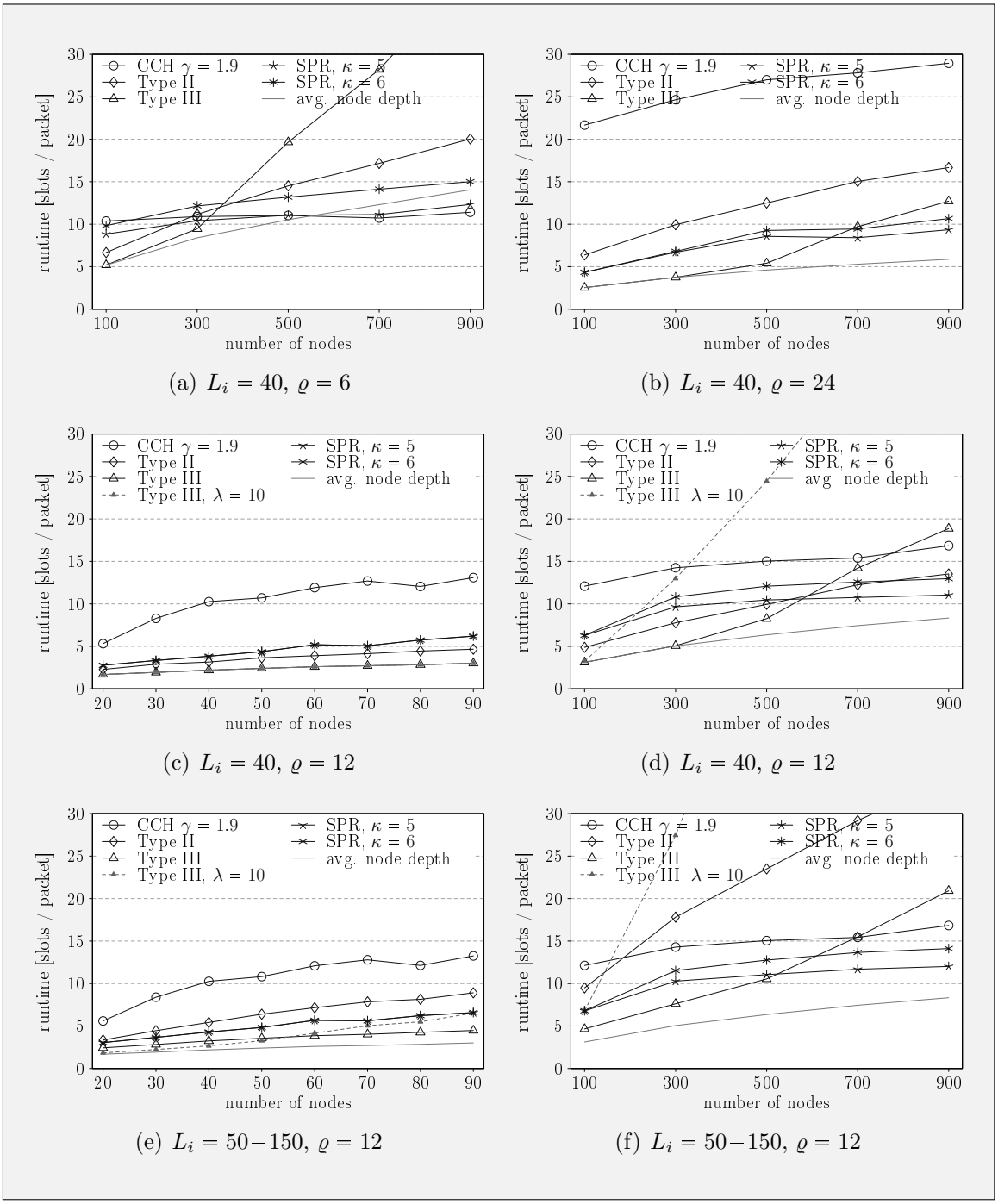
as suggested in Section 3.4.1. CCH 3-hop and SPR $\kappa = 4$ are not considered, as their collision affliction precludes comparing them. To provide a meaningful comparison, limited buffers ($B = 200$) and children ($C = 8$) are assumed. Furthermore, only high values of L_i are of interest here, as they reflect the actual situation found in a collection phase.

5.3.1 Runtime

Per-packet runtime of the different scheduling schemes is depicted and compared in Figure 5.14. In addition to the runtime of the different schedules, all plots show the minimum runtime as found in Equation 5.1. Note that this minimum is equivalent to the ideal runtime of Type III.

For low density and equal buffer fill levels, as seen in Figure 5.14(a), plain Type III performs best up to a network size of 300 nodes. Note that the load-aware variant cannot improve runtime here. As indicated by a look at the average node depth, Type III starts suffering from buffer congestion for $N = 300$, so that at this point, it loses its leading position. CCH $\gamma = 1.9$ and SPR ($\kappa = 5$) start profiting from the spatial reuse of slots for $N = 500$ and larger networks. However, even SPR with $\kappa = 6$ is likely to produce collision-afflicted schedules in these sparse networks, so that its application will require additional effort, e.g., adapting κ, if collisions occur. The problem of CCH $\gamma = 1.9$ is that it cannot be implemented distributedly, if at all. The k-hop variant of CCH is unusable, as it is generally incapable of producing collision-free schedules. Finally, Type II is suboptimal, too. It has a high runtime compared to the average node depth, which is caused by the fact that nodes have few children and that slots are not reused by nodes with a high depth in the tree. This is caused by the employed reuse strategy. Furthermore, slot storage cannot be performed by Type II in large networks (cf. Section 3.2.3). As a result, more investigation is required for this kind of network.

If network density is increased to $\varrho = 24$, the performance of CCH $\gamma = 1.9$ is declining rapidly (Figure 5.14(b)), because round length increases. Besides, Type I is inapplicable in dense networks, because storing and managing neighborhood information, based on hops or inference, is infeasible. In contrast, SPR profits in dense networks. As seen for the sparse networks, there is a crossover network size from which on SPR becomes faster than Type III. However, SPR is not able to achieve the minimum runtime: average node depth is below κ, so that slots are rarely reused. Hence, Type III could theoretically beat SPR for a larger B even in large, dense networks.



(a) $L_i = 40$, $\varrho = 6$

(b) $L_i = 40$, $\varrho = 24$

(c) $L_i = 40$, $\varrho = 12$

(d) $L_i = 40$, $\varrho = 12$

(e) $L_i = 50{-}150$, $\varrho = 12$

(f) $L_i = 50{-}150$, $\varrho = 12$

Figure 5.14: Runtime comparison for $C = 8$, $B = 200$, no bit errors

However, as soon as packet loss occurs, runtime of Type III is considerably affected, so that SPR may be considered the best choice. Again, Type II cannot compete, as buffer congestion delays slot reuse.

Figures 5.14(c) and 5.14(d) compare runtime for $\varrho = 12$. Here, CCH $\gamma = 1.9$ is left behind by its opponents, as slots are rarely reused in small networks. Type III generates the fastest schedules, but as it assigns too many slots (as compared to buffer size) to nodes close to the sink, it cannot maintain performance in large networks. Type II may be considered an alternative in small networks, if initial fill levels of nodes are moderate, and if massive packet loss is observed. Given moderate fill levels, runtime of Type II has been shown to increase relative to Type III. If packet loss is observed, runtime of Type III is severely elevated, whereas Type II is less affected. In combination, runtime comes to a tie. Note that this is valid in small networks only, as slot storage of Type II does not pose a problem for the restricted amount of a node's memory here. In large networks, SPR is perfectly suited, as it outperforms all other schedules, even in the case of bit errors. Moreover, Figure 5.14(d) indicates that relative runtime of SPR cannot be competed with in even larger networks, because it approaches—and will eventually get below—the theoretical no-slot-reuse optimum.

Due to packet loss and interrupted links, buffer fill levels of individual nodes are not likely to be equally distributed at the beginning of a collection phase. Moreover, initial buffer fill levels can be expected to be high, possibly close to the soft limit $\tilde{B}$. Figures 5.14(e) and 5.14(f) show the standings of the discussed schemes for this case. As explained in Section 5.2.3, the performance of CCH $\gamma = 1.9$ is not influenced by the initial fill levels. Yet, it is not the fastest solution. Although Type II is generally load-adaptive, it is severely affected by the high load and hereby caused buffer congestion, since slot reuse is heavily delayed. Hence, Type II is not an option in this situation, either. Load-aware Type III shows best performance in very small networks, but huge buffers are required to make it applicable in medium sized networks. Adoption in large networks is impossible with current memory limitations of wireless sensor nodes. Plain Type III is slower than its load-aware variant in very small networks, yet is later affected by limited buffers. As discussed in Section 5.2.5, it is slightly slower than in the situation of uniform buffer fill levels. SPR exhibits low runtime for all network sizes and considerably undercuts runtime of Type III in large networks. If high packet loss is observed, and if buffers are small, SPR may be considered an alternative for Type III even in small networks.

5.3.2 Energy-Efficiency

Plain discussion of runtime has revealed the advantages of Type III and SPR. Concerning runtime only, the former can generally be considered as the first choice in small networks, and the latter outperforms its opponents in large networks. Yet, network lifetime is crucial, so that energy-efficiency must be considered, too. Figure 5.15 depicts the overall energy-consumption of all nodes per created packet. Precisely, the cumulative time spent outside the sleep mode is shown. Assuming equal power required for sending, receiving, and listening—a justifiable assumption as mentioned in Section 2.1.1 on page 7—this value is proportional to the actual amount of energy consumed.

Most of the graphs show similar behavior and clearly evince a close relation to average node depth, which is proportional to the minimum energy to collect all packets. The corresponding minimum time spent outside the sleep mode can be obtained by multiplying average node depth with twice the time—it must be counted for both communication partners—elapsing from the beginning of a slot until completed reception of the acknowledgment. The packet lengths and slot timings as presented in Section 5.1.1 are used for this calculation. The resulting curve is also visualized in the plots to give a better impression of communication overhead and actual energy-efficiency of the different scheduling schemes. An immediate outcome of this is the justification of the observation that energy consumption is affected by network density. This can be followed by comparing the sequence of Figures 5.15(a), 5.15(d), and 5.15(b).

Particularly in large networks, CCH $\gamma = 1.9$ causes high energy consumption, or wastage. It grows with increasing initial buffer fill levels. In conclusion, energy consumption for Type I rises with overall network load. This is caused by buffer overflows that cannot be prevented by the flow control. It is possible to considerably alleviate this problem by applying a more suitable flow control scheme. Currently, buffer congestion is avoided by calculating the difference between the current buffer fill level and the soft limit. This does not take into consideration, whether a node will be able to forward any packets to its parent in the near future.

The energy consumption of Type II is above the possible minimum, but not as much as Type I. A small fraction is caused by sending slots up in the tree. However, it is comparably low. The main source of energy wastage is caused by buffer overflows. Although reusing slots leads to fewer buffer overflows as compared to Type I, its effect is aggravated by a high network load. The initial load of each node influences the reuse delay, as explained in Section 5.2.4. For this reason, the results in 5.15(f)

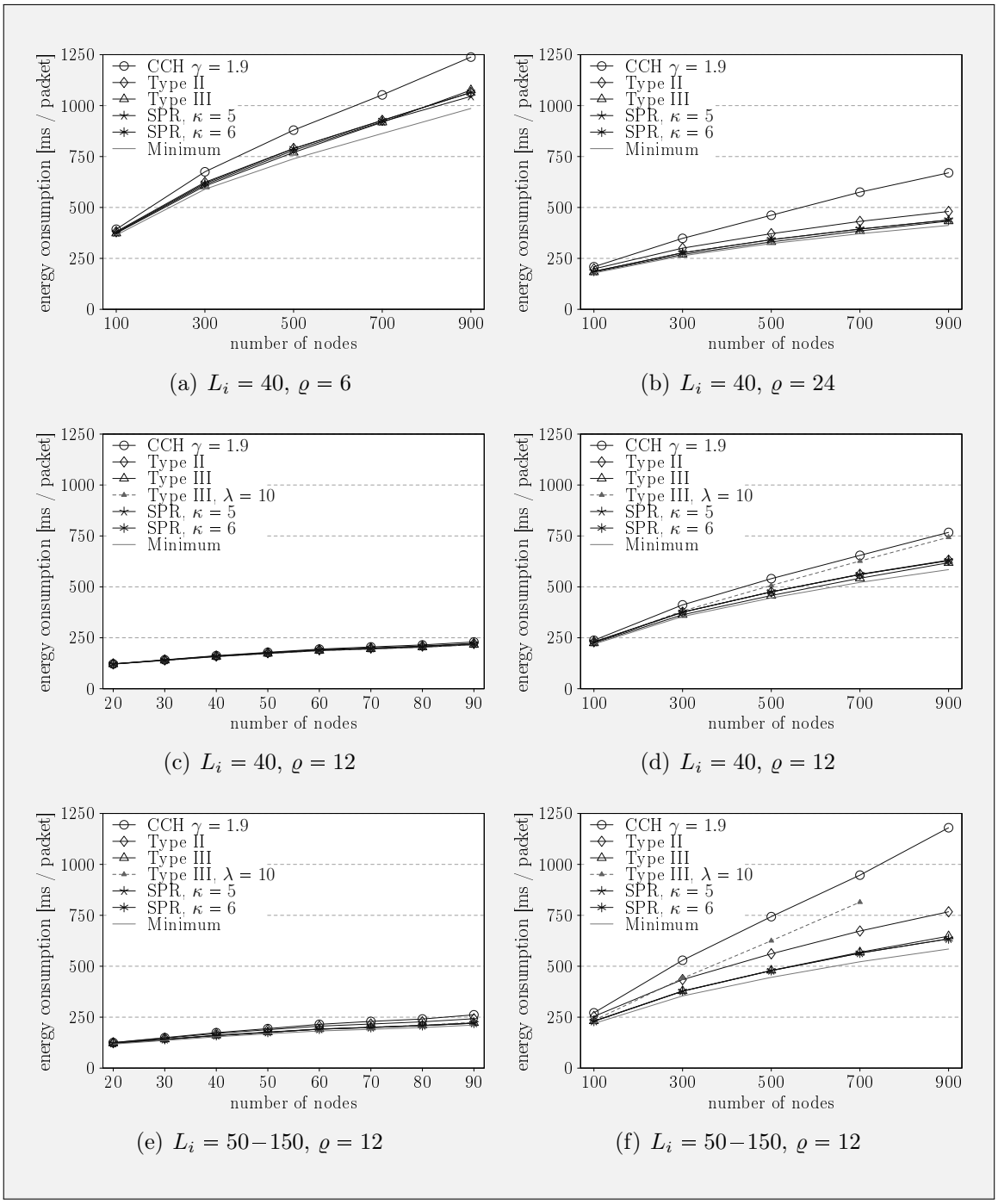


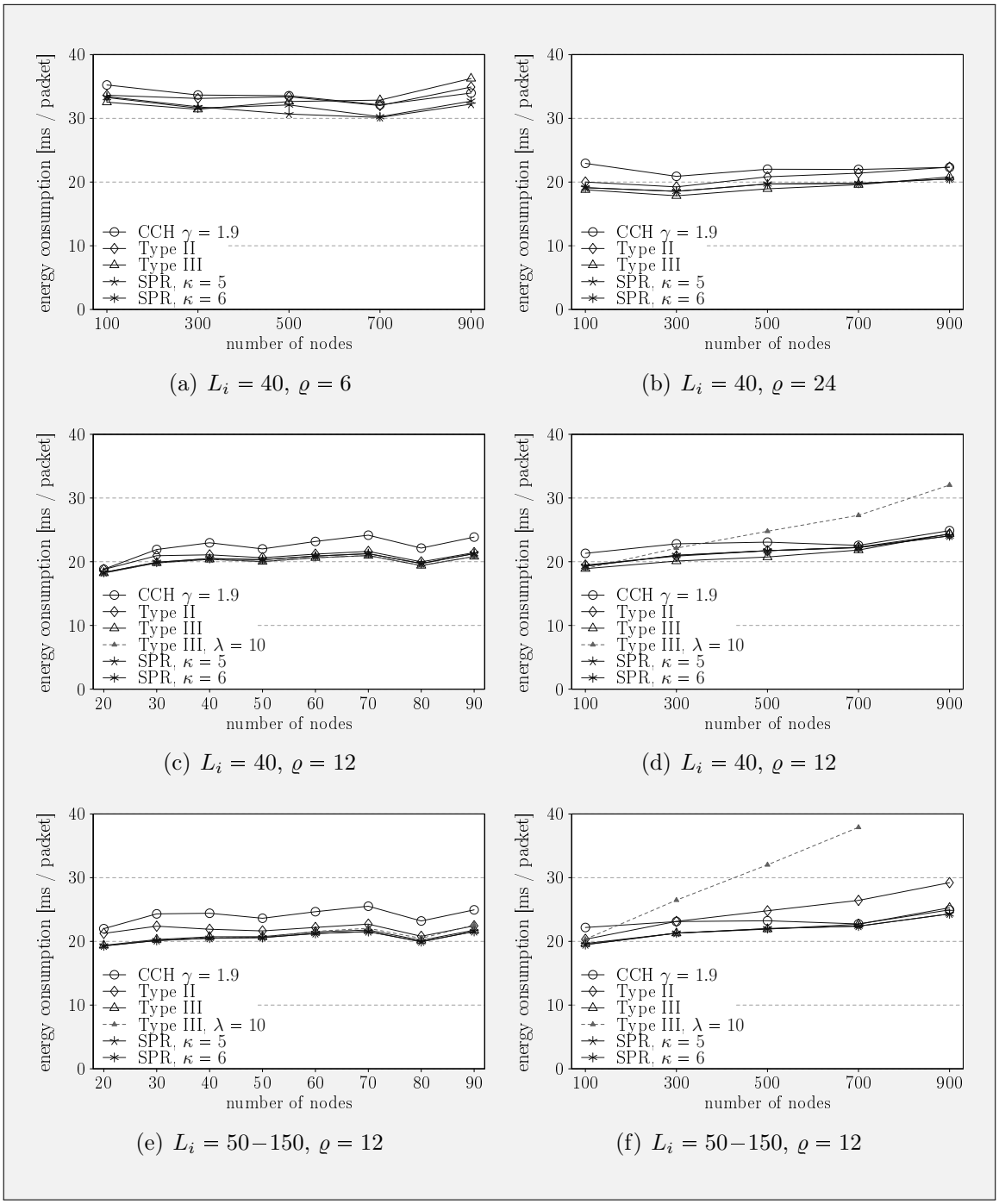
Figure 5.15: Overall energy consumption for $C = 8$, $B = 200$, and no bit errors

exhibit a worse energy-efficiency of Type II than those in 5.15(d). Type II lends itself as a suitable solution concerning energy-consumption in small networks, e.g., see Figure 5.15(c), because it performs close to the minimum and does not require explicit slot assignment and thus no additional communication overhead.

The flow control in use produces very few buffer overflows for plain Type III. Precisely, their may be at most one dropped packet on each link in every round. Because this is small as compared to round length, energy consumption of plain Type III is very low. Particularly in large networks, e.g., as shown in Figure 5.15(d). This proves the energy-efficiency of Type III. Load-aware Type III is not as energy-efficient as the plain variant in large networks and with varying node fill levels, as depicted in Figure 5.15(f). Here, many slots cannot be used, particularly at the end of a data collection phase. This provokes the sending of keepalives. Note that for $\lambda = 10$, values for $N = 900$ are not available (cf. Section 5.1.2). Indeed, flow control could be improved by attaching the number of slots to skip ω_p to the last data packet, if a child cannot use the remainder of its slots in the same round. This promises to be a large improvement, providing load-aware Type III with the energy-efficiency of the plain variant.

SPR is as energy-efficient as Type III. The applied flow control causes few keepalive packets only. The heuristic for skipping slots in case of buffer underrun thus shows to be suitable, so that the amount of energy consumed is not influenced by κ. This is true for all investigated network sizes and densities as well as for different and equal buffer fill levels. Furthermore, Type III and SPR have the same low overhead for slot assignment, so that overall energy-efficiency can be attested.

As stated in Section 3.4.1, the maximum energy consumed by a single node is also an important metric. The corresponding data is presented in Figure 5.16 and is again expressed as the time spent outside the sleep mode. The sink is not taken into consideration here, because it is usually equipped with unlimited or high energy resources. Maximum energy consumption is related to network density, because the latter influences the number of children at the sink. Since these nodes are exposed to the highest load in the network—they have to forward all data of their subtree–, their energy consumption is the highest in the network and is hence depicted in the corresponding plots. They reflect the behavior of the just described overall energy consumption. For Type I, energy consumption is the highest due to massive buffer overflows. Type II suffers less from this problem and the influence of sending slots up in the tree is comparably low. Type III and SPR are approximately at the same low level.



(a) $L_i = 40$, $\varrho = 6$

(b) $L_i = 40$, $\varrho = 24$

(c) $L_i = 40$, $\varrho = 12$

(d) $L_i = 40$, $\varrho = 12$

(e) $L_i = 50{-}150$, $\varrho = 12$

(f) $L_i = 50{-}150$, $\varrho = 12$

Figure 5.16: Maximum per-node energy consumption for $C = 8$, $B = 200$, and no bit errors

5.3.3 Summary

As a conclusion, the key findings of the presented analysis and comparison are summarized in the following. They are enriched with the outcome of Chapter 3. Table 5.3 can be used as a decision guidance for the choice of the most appropriate scheduling scheme for a given scenario.

Breadth-first search produces suitable data-gathering trees, in which the limitation of C does not have a large influence on tree depth and individual node depth. Only close to the sink and in dense networks, a limited C becomes noticeable, so that average subtree size is slightly increased. Finally, the number of leafs increases with network density.

Unlike frequently stated in the literature, Type I does not reveal itself as particularly suitable for data-gathering. In contrast, it consumes more energy than its competitors and thus demands for an advanced flow control. Furthermore, k-hop approaches are not capable of producing collision-free schedules and the used CCH $\gamma = 1.9$ approach cannot be implemented distributedly. Hence, arguing that runtime of Type I is below its opportunities due to the larger number of slots created by CCH $\gamma = 1.9$ is bootless. Although interference-based solutions for Type I exist, they are difficult to implement and presumably sensitive to signal power deviation and overlapping. Besides, exchange of neighborhood information may not be possible in sparse networks. This is true, because these networks contain many voids that prevent exchanging data within a limited number of hops. In contrast, neighborhood storage and management is not viable in dense networks due to limited memory.

Type II can only be used in small networks, because the limited amount of memory resources is in contradiction to the demands of this scheduling scheme for large networks. It has been shown that the ordering of slots does not have a significant impact on runtime, so that nodes could simply use their unique identifiers in order to determine their sending slots. Having calculated the routing tree, sending slots of a node's children are also known. As sending slots up in the tree for reuse does not cause high overhead, Type II can be successfully applied in small networks. However, runtime does not achieve the possible minimum and the influence of packet loss and buffer overflows must not be underestimated.

Best performance in small networks is achieved by Type III with ascendingly ordered slots. The load-aware variant is capable of getting close to the optimum, even if initial buffer fill levels vary among nodes. Type III is slightly affected by the position of the sink, where the center is the best position. Yet, more severe drawbacks have been identified. Firstly, packet loss is likely to elevate runtime. Secondly, Type III

produces a huge number of slots, which is particularly grave for the load-aware variant with a small λ. As soon as the number of slots assigned to a single node exceeds its buffer size, slots are wasted and runtime is dramatically increased. As each node in the network is assigned a set of consecutive slots with the size of its subtree, buffer size has to be in the order of N as a worst case estimate. Due to the limited amount of memory and computing power, an appropriate buffer management is not feasible in large networks. This is particularly true, if the L_i vary considerably among nodes, which requires a small λ to achieve low runtime at the cost of a huge number of slots generated.

The newly introduced SPR scheduling scheme has revealed itself as being highly energy-efficient in all kinds of networks. Slot assignment comes at a small overhead and slot storage does only depend on a node's number of children and κ. Low runtime is achieved especially in large networks, which is accomplished by spatially reusing slots on paths from leafs to the sink. In particular, SPR is able to outperform all other approaches at a given network size depending on the density. Unlike Type III, SPR guarantees low buffer utilization and is hardly influenced by packet loss. Furthermore, SPR performs well for high network load and varying buffer fill levels among nodes. However, application in sparse networks is difficult due to collisions, but not impossible.

	Type I	Type II, enhanced	Type III, plain	Type III, load aware	SPR
Network Size N					
Small	⊖⊖	⊕	⊕⊕	⊕⊕	⊖
Medium	○	○	⊕	○	○
Large	⊖	⊖⊖	⊖⊖	⊖⊖	⊕⊕
Density ϱ					
Low	⊖⊖	○	⊖	⊖⊖	⊖
Medium	○	○	○	○	⊕
High	⊖⊖	○	⊕	○	⊕
Initial Fill Level L_i					
Low	○	⊕	⊕⊕	⊕⊕	⊕
High	⊖	⊖	⊖⊖	⊖⊖	⊕⊕
Variation of Initial Fill Levels L_i					
Low	⊕⊕	⊕	⊖	⊕	⊕⊕
High	⊕⊕	⊖	⊖⊖	⊕	⊕⊕
Collisions and Packet Loss					
Collisions / Yield	✓/ ⊖⊖	–/ ⊕⊕	–/ ⊕⊕	–/ ⊕⊕	✓/ ○
Packet Loss	⊕	○	⊖⊖	⊖⊖	⊕
Sink Position					
Center	⊕	⊕	⊕⊕	⊕⊕	⊕
Edge	⊖⊖	○	⊖	⊖⊖	⊕⊕
Arbitrary	⊖	○	○	○	⊖
Limitations					
Limited Children C	⊖	⊕⊕	⊕	⊕	⊕⊕
Limited Buffer B	○	○	⊖	⊖⊖	⊕⊕
Additional Requirements	Ext. Flow Control	Ext. Flow Control	$B \geqslant \max_i L_i + \lvert\mathcal{T}_i\rvert$	$B \geqslant \max_i L_i + \lceil L_i^*/\lambda \rceil$	$5 \leqslant \kappa < \max_i \frac{d_{0,i}}{R_{com}}$

Table 5.3: Characteristics of the TDMA schedules: Decision Guidance

Chapter 6

Conclusion and Outlook

In recent years, wireless sensor networks have been frequently adopted for data-gathering, as they facilitate the collection and permit the generation of high resolution data. Due to limited resources, energy-efficiency is mandatory to prolong network lifetime, which particularly implies an effective operation of the radio transceiver. Employing dedicated data-collection phases in combination with Time-Division-Multiple Access (TDMA) for scheduled transmission offers to meet this end. A variety of schedules exists, but all of them exhibit weaknesses. Moreover, a detailed comparison between the distinct types of schedules is not available. Therefore, it is neither known, which one to prefer in a given data-gathering scenario, nor if they are appropriate at all. In order to overcome this deficiency, the objectives of this thesis have been to investigate the existing approaches theoretically, to derive an advanced scheduling scheme from the results, and to accomplish a detailed comparison under realistic conditions via simulation.

These objectives have been achieved as follows. An analytical investigation of the existing schedules has been carried out, being the first to permit a detailed comparison. It provides an elaborate runtime analysis and reveals susceptible weaknesses. In particular, all schedules provoke buffer congestion, which is likely to affect runtime and energy-efficiency. Furthermore, packet loss poses a severe runtime threat on some of the schedules. Based on the results of this analysis, the new Spatial Path-Based Reuse (SPR) scheme has been devised. Its main advantages are low runtime in various scenarios, high energy-efficiency, and an outstanding low buffer utilization. Furthermore, SPR generates schedules with low overhead and a small memory footprint, while its distributed implementation is frugal. Although the analytical investigation provides a deeper insight and understanding of the schedules, a simulative compar-

ison of the existing approaches and SPR is needed to validate its outcome and to address open issues. Therefore, a highly flexible and extensible simulation framework has been designed and implemented using the ns-2 network simulator. Finally, extensive simulations consisting of more than 400,000 individual runs have been conducted. The obtained results enable a detailed investigation of the different approaches and underline their strengths and weaknesses in diverse data-gathering scenarios.

The simulation results substantiate that TDMA generally permits low runtime and energy-efficiency close to the possible minimum. Unlike frequently stated in the literature, Type I schedules that minimize the round length perform poorly. The generated schedules suffer severely from collisions and lead to an unacceptable yield, rendering runtime comparison infeasible. To make a comparison possible anyhow, a revised Type I variant has been implemented. It guarantees collision-free schedules by making use of the interference radius. As the latter is usually not available, this approach cannot be implemented distributedly, if at all. Although the round length of the according schedules is still small, the achieved runtime is high and energy-efficiency unsatisfactory, which is caused by heavy buffer congestion. In conclusion, Type I has failed to prove premature expectancies stated elsewhere, and is currently unusable for data-gathering.

Even though Type II is a trivial approach, it offers better runtime performance particularly in small networks. Yet, it cannot be used in large networks due to the high memory consumption for slot storage. Furthermore, Type II can achieve low runtime solely in the case of low network load. Yet, as it is independent of any slot ordering, nodes can use their unique identifiers to determine their slots, so that no overhead is produced for slot assignment. This may lead to high overall energy-efficiency in small networks with low load and without the demand for low runtime. If the opposite is the case, Type III is favorable. It achieves the best runtime in small networks with energy-efficiency close to the possible minimum, if buffer fill levels are equal. Yet, increasing network size causes a larger amount of slots assigned to nodes close to the sink. At some point, the number of slots exceeds the buffer size of some nodes. This is fatal, as slots are assigned consecutively, so that some slots are wasted and runtime is therefore dramatically elevated. If nodes have different initial buffer fill levels, the load-aware variant of Type III is preferable. Yet, it even intensifies the buffering problem and is thus applicable solely in very small networks.

The newly developed SPR slot assignment produces collision-afflicted schedules almost exclusively in sparse networks, which is caused by paths bending around voids. This partial shortcoming can be overcome with low overhead, but requires further

investigation. With increasing density, collisions vanish and SPR achieves the lowest runtime of all schedules in large networks. It is potentially able to undercut the theoretical optimum of Type III, since slots are spatially reused. SPR is the only scheme that does not cause buffer overflows, so that low runtime is preserved even in the case of small buffers. Moreover, it achieves high energy-efficiency close to the possible minimum. In combination with its low assignment overhead and sophisticated slot storage, SPR confirms its high adequacy for data-gathering.

As a result of the large amount of simulations and obtained results, a variety of issues has been identified for further investigation. The first topic regards network density. A deeper and finer grained analysis on this matter is required to provide a more precise comparison of Type III and SPR. In this context, locally varying densities should also be investigated. As all schemes perform poorly in sparse networks, more research is demanded here. In particular, a strategy for avoiding collisions or repairing collision-afflicted schedules produced by SPR is promising, because runtime of Type III cannot be increased in these networks. Moreover, large networks with more than 1,000 nodes have not been considered so far. Even though SPR promises preeminent runtime, it is not known when and under which conditions it is able to actually undercut the ideal performance of Type III. In small networks, the performance of Type II may not have been at its optimum, because only one reuse strategy has been considered. This points out the relevance to develop and test new strategies. In addition, devising and implementing an advanced flow control will boost runtime and energy-efficiency of Type II and presumably of Type III. Further optimizing energy consumption of SPR is possible by improving the slot-skipping mechanism.

For the comparison performed in this thesis, a two-phase data-collection strategy has been employed. This solution disencumbers from continuous tree maintenance, which is required by its periodical, on-demand counterpart. As tree maintenance implies additional wireless communication, the two-phase strategy promises higher energy-efficiency. One direction of research would be to analyze overall energy-consumption of both strategies in order to examine carefully, if and when this holds. Another one would be to compare the different types of TDMA schedules in an on-demand scenario, since the two-phase strategy is possible exclusively in the case of delay-tolerant data. Note that the enhanced Type II scheme can solely be employed as part of a two-phase strategy due to slot reuse. Furthermore, the influence of multiple sinks is an important area of research for data-gathering, since recent applications pose an according demand, where multiple sensors and actors, i.e., sinks, are employed. Besides, multiple sinks decrease the high load and therefore energy

consumption of nodes close to the sink, which is a severe threat to network lifetime especially in large networks. Finally, the applicability of TDMA in general and the discussed schemes in particular is an completely open issue in networks with limited or partial mobility.

Bibliography

[AHM+06] G.-S. Ahn, S. G. Hong, E. Miluzzo, A. T. Campbell, and F. Cuomo. Funneling-MAC: A Localized, Sink-Oriented MAC for Boosting Fidelity in Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, Boulder, CO, USA, October 2006.

[ASD+06] M. Ali, U. Saif, A. Dunkels, T. Voigt, K. Römer, K. Langendoen, J. Polastre, and Z. A. Uzmi. Medium Access Control Issues in Sensor Networks. *ACM Computer Communication Revue*, 36(2), 2006.

[AY06] T. R. Andel and A. Yasinac. On the Credibility of Manet Simulations. *IEEE Computer Magazine*, 39(7), 2006.

[BBLV05] G. Barrenetxea, B. Beferull-Lozano, and M. Vetterli. Efficient Routing with Small Buffers in Dense Networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, Los Angeles, CA, USA, April 2005.

[BGLA01] L. Bao and J. J. Garcia-Luna-Aceves. A New Approach to Channel Access Scheduling for Ad Hoc Networks. In *Proceedings of the 7th International Conference on Mobile Computing and Networking (MobiCom '01)*, Rome, Italy, July 2001.

[BR04] P. Basu and J. Redi. Effect of Overhearing Transmissions on Energy Efficiency in Dense Sensor Networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN '04)*, Berkeley, CA, USA, April 2004.

[BRD+07] K. L. Bryan, T. Ren, L. DiPippo, T. Henry, and V. Fay-Wolfe. Towards Optimal TDMA Frame Size in Wireless Sensor Networks. Technical report, Department of Computer Science and Statistics, University of Rhode Island, Kingston, RI, USA, March 2007.

[BvRW07] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, Cambridge, MA, USA, April 2007.

[CMGL05] S. Cui, R. Madan, A. Goldsmith, and S. Lall. Energy-Delay Tradeoffs for Data Collection in TDMA-Based Sensor Networks. In *Proceedings of the 40th International Conference on Communications (ICC '05)*, Seoul, Korea, May 2005.

[CO05] R. Cardell-Oliver. ROPE: A Reactive, Opportunistic Protocol for Environment Monitoring Sensor Networks. In *Proceedings of the Second Workshop on Embedded Networked Sensors (EmNetS '05)*, Sydney, Australia, May 2005.

[DEA06] I. Demirkol, C. Ersoy, and F. Alagoz. MAC Protocols for Wireless Sensor Networks: A Survey. *IEEE Communications Magazine*, 44(4), 2006.

[DG07] A. Dhamdhere and J. Grönkvist. Joint Node and Link Assignment in an STDMA Network. In *Proceedings of the 65th Vehicular Technology Conference (VTC '07)*, Dublin, Ireland, April 2007.

[DH03] H. Dai and R. Han. A Node-Centric Load Balancing Algorithm for Wireless Sensor Networks. In *Proceedings of the 46th Global Communications Conference (Globecom '03)*, San Francisco, CA, USA, December 2003.

[DML03] E. J. Duarte-Melo and M. Liu. Data-Gathering Wireless Sensor Networks: Organization and Capacity. *Computer Networks*, 43, 2003.

[EB04] C.-T. Ee and R. Bajcsy. Congestion Control and Fairness for Many-to-One Routing in Sensor Networks. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, MD, USA, November 2004.

[ECPS02] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1), 2002.

[EV05] S. Coleri Ergen and P. Varaiya. TDMA Scheduling Algorithms for Sensor Networks. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA, July 2005.

[FV08] K. Fall and K. Varadhan. The *ns* Manual. http://www.isi.edu/nsnam/ns/ns-documentation.html, January 2008. Last visited: 05/06/2008.

[GDP05] S. Gandham, M. Dawande, and R. Prakash. Link Scheduling in Sensor Networks: Distributed Edge Coloring Revisited. In *Proceedings of the 24th Joint Conference of the IEEE Computer and Communications Societies (Infocom '05)*, Miami, FL, USA, March 2005.

[Grö04] J. Grönkvist. Comparison Between Scheduling Models for Spatial Reuse TDMA. In *Proceedings of the Second Workshop on Affordable Wireless Services and Infrastructure (AWSI '04)*, Stockholm, Sweden, June 2004.

[Grö06] J. Grönkvist. Novel Assignment Strategies for Spatial Reuse TDMA in Wireless Ad Hoc Networks. *Wireless Networks*, 12(2), 2006.

[IGE00] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of*

the 6th International Conference on Mobile Computing and Networking (MobiCom '00), Boston, MA, USA, August 2000.

[JE07] J. Jeong and C.-T. Ee. Forward Error Correction in Sensor Networks. In *Proceedings of the International Workshop on Wireless Sensor Networks (WWSN '07)*, Marrakesh, Morocco, June 2007.

[Joe05] I. Joe. Optimal Packet Length with Energy Efficiency for Wireless Sensor Networks. Kobe, Japan, May 2005.

[KPC+06] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Wireless Sensor Networks for Structural Health Monitoring. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, Boulder, CO, USA, October 2006.

[KS06] K.-H. Kim and K. G. Shin. On Accurate Measurement of Link Quality in Multi-Hop Wireless Mesh Networks. In *Proceedings of the 12th International Conference on Mobile Computing and Networking (MobiCom '06)*, Los Angeles, CA, USA, September 2006.

[LDCO06] W. L. Leer, A. Datta, and R. Cardell-Oliver. FlexiMAC: A Flexible TDMA-Based MAC Protocol for Fault-Tolerant and Energy-Efficient Wireless Sensor Networks. In *Proceedings of the 14th International Conference on Networks (ICON '06)*, Singapore, September 2006.

[Liu] K. Liu. Understanding the Implementation of IEEE MAC 802.11 Standard in ns-2. http://www.cs.binghamton.edu/∼kliu/research/ns2code/index.html. Last visited: 05/06/2008.

[LKR04] G. Lu, B. Krishnamachari, and C. Raghavendra. An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Sensor Networks. In *Proceedings of the International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN '04)*, Santa Fe, NM, USA, April 2004.

[MPR+05] K. Martinez, P. Padhy, A. Riddoch, H. L. R. Ong, and J. K. Hart. Glacial Environment Monitoring Using Sensor Networks. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN '05)*, Stockholm, Sweden, June 2005.

[MPS+02] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the First International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, Atlanta, GA, USA, September 2002.

[NGM07] M. Nunes, A. Grilo, and M. Macedo. Interference-Free TDMA Slot Allocation in Wireless Sensor Networks. In *Proceedings of the 32nd Conference on Local Computer Networks (LCN '07)*, Montreal, Canada, October 2007.

[PHC04] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, MD, USA, November 2004.

[Ram97] S. Ramanathan. A Unified Framework and Algorithm for (T/F/C)DMA Channel Assignment in Wireless Networks. In *Proceedings of the 16th Conference on Computer Communications (Infocom '97)*, Kobe, Japan, April 1997.

[ROGLA03] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys '03)*, Los Angeles, CA, USA, November 2003.

[ROGLA05] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-Efficient, Application-Aware Medium Access for Sensor Networks. In *Proceedings of the Second International Conference on Mobile Ad-Hoc and Sensor Systems (MASS '05)*, Washington D.C., USA, November 2005.

[RR04] F. J. Ros and P. M. Ruiz. Implementing a New Manet Unicast Routing Protocol in NS2. Technical report, Department of Information and Communications Engineering, University of Murcia, Spain, December 2004.

[RR08] K. Römer and C. Renner. Aggregating Sensor Data from Overlapping Multi-Hop Network Neighborhoods: Push or Pull? In *Proceedings of the 5th International Conference on Networked Sensing Systems (INNS '08)*, Kanazawa, Japan, June 2008.

[RRDM08] V. Ramamurthi, A. S. Reaz, S. Dixit, and B. Mukherjee. Link Scheduling and Power Control in Wireless Mesh Networks with Directional Antennas. In *Proceedings of the 43rd International Conference on Communications (ICC '08)*, Bejing, China, May 2008.

[RWAM05] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: A Hybrid MAC for Wireless Sensor Networks. In *Proceedings of the Third International Conference on Embedded Networked Sensor Systems (SenSys '05)*, San Diego, CA, USA, November 2005.

[RWMX06] I. Rhee, A. Warrier, J. Min, and L. Xu. DRAND: Distributed Randomized TDMA Scheduling for Wireless Ad-hoc Networks. In *Proceedings of the 7th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06)*, Florence, Italy, May 2006.

[RXMC05] B. Ren, J. Xiao, J. Ma, and S. Cheng. An Energy-Conserving And Collision-Free MAC Protocol Based on TDMA for Wireless Sensor Networks. In *Proceedings of the International Conference on Mobile Ad-Hoc and Sensor Networks (MSN '05)*, Wuhan, China, December 2005.

[SAA03] Y. Sankarasubramaniam, Ö. B. Akan, and I. F. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proceedings of the 4th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '03)*, Annapolis, MD, USA, 2003.

[SAM03] Y. Sankarasubramaniam, I. F. Akyildiz, and S. W. McLaughlin. Energy Efficiency Based Packet Size Optimization in Wireless Sensor Networks. In *Proceedings of the First International Workshop on Sensor Network Protocols and Applications (SNPA '03)*, Anchorage, AK, USA, May 2003.

[SH03] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications (SNPA '03)*, Anchorage, AK, USA, May 2003.

[SLR+05] J. Schiller, A. Liers, H. Ritter, R. Winter, and T. Voigt. ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS '05)*, Hawaii, HI, USA, January 2005.

[SPMC04] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a Sensor Network Expedition. In *Proceedings of the First European Conference on Wireless Sensor Networks (EWSN '04)*, Berlin, Germany, January 2004.

[SRS90] S. Y. Seidel, T. S. Rappaport, and R. Singh. Path Loss and Multipath Delay Statistics in four European Cities for 900 MHz Cellular and Microcellular Communications. *Electronics Letters*, 26(20), September 1990.

[SWC+07] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. LUSTER: Wireless Sensor Network for Environmental Research. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys '07)*, Sydney, Australia, November 2007.

[SYL06] W.-Z. Song, F. Yuan, and R. LaHusen. Time-Optimum Packet Scheduling for Many-to-One Routing in Wireless Sensor Networks. In *Proceedings of the Third International Conference on Mobile Adhoc and Sensor Systems (MASS '06)*, Vancouver, Canada, October 2006.

[TRV+05] V. Turau, C. Renner, M. Venzke, S. Waschik, C. Weyer, and M. Witt. The Heathland Experiment: Results and Experiences. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN '05)*, Stockholm, Sweden, June 2005.

[TTS05] N. Thepvilojanapong, Y. Tobe, and K. Sezaki. On the Construction of Efficient Data Gathering Tree in Wireless Sensor Networks. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS '05)*, Kobe, Japan, May 2005.

[TW07a] V. Turau and C. Weyer. Long-Term Reliable Data Gathering Using Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS '07)*, Braunschweig, Germany, June 2007.

[TW07b] V. Turau and C. Weyer. Scheduling Transmission of Bulk Data in Sensor Networks Using a Dynamic TDMA Protocol. In *Proceedings of the International Workshop on Data Intensive Sensor Networks 2007 (DISN '07)*, Mannheim, Germany, May 2007.

[TW07c] V. Turau and C. Weyer. TDMA-Schemes for Tree-Routing in Data Intensive Wireless Sensor Networks. In *Proceedings of the First International Workshop on Protocols and Algorithms for Reliable and Data Intensive Sensor Networks (PARIS '07)*, Pisa, Italy, October 2007.

[TWR08] V. Turau, C. Weyer, and C. Renner. Efficient Slot Assigment for the Many-to-One Routing Pattern in Sensor Networks. In *Proceedings of the International Workshop on Sensor Network Engineering (IWSNE '08)*, Santorini Island, Greece, June 2008.

[Unt08] S. Unterschütz. Network Simulator (NS-2), Institute of Telematics, Hamburg University of Technology, Germany. http://wiki.ti5.tu-harburg.de/wsn/ns2/intro, 2008. Last visited: 05/06/2008.

[WC01] A. Woo and D. Culler. A Transmission Control Scheme for Media Access in Sensor Networks. In *Proceedings of the 7th International Conference on Mobile Computing and Networking (MobiCom '01)*, Rome, Italy, July 2001.

[WCK02] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. In *Proceedings of the First International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, Atlanta, GA, USA, September 2002.

[WCS+07] T. Wark, P. Corke, P. Sikka, L. Klingbeil, Y. Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley. Transforming Agriculture through Pervasive Wireless Sensor Networks. *IEEE Pervasive Computing*, 6(2), 2007.

[WR05] A. Warrier and I. Rhee. Stochastic Analysis of Wireless Sensor Network MAC Protocols. Technical report, Computer Science Department, North Carolina State University, Raleigh, NC, USA, August 2005.

[WTC03] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys '03)*, Los Angeles, CA, USA, November 2003.

[WWJ+05] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the Capture Effect for Collision Detection and Recovery. In *Proceedings of the Second Workshop on Embedded Networked Sensors (EmNetS '05)*, Sydney, Australia, May 2005.

[XK04] F. Xue and P. R. Kumar. The Number of Neighbors Needed for Connectivity of Wireless Networks. *Wireless Networks*, 10(2), 2004.

[YH04] W. Ye and J. Heidemann. Medium Access Control in Wireless Sensor Networks. *Wireless Sensor Networks*, 2004.

[YHE02] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 21st Conference on Computer Communications (Infocom '02)*, New York, NY, USA, June 2002.

[YW08] N. B. Shroff Y. Wu, S. Fahmy. On the Construction of a Maximum-Lifetime Data Gathering Tree in Sensor Networks: NP-Completeness and Approximation Algorithm. In *Proceedings of the 27th Conference on Computer Communications (Infocom '08)*, Phoenix, AZ, USA, April 2008.

[ZHSA05] G. Zhou, T. He, J. A. Stankovic, and T. Abdelzaher. RID: Radio Interference Detection in Wireless Sensor Networks. In *Proceedings of the 24th Joint Conference of the IEEE Computer and Communications Societies (Infocom '05)*, Miami, FL, USA, March 2005.

[ZK03] C. Zhou and B. Krishnamachari. Localized Topology Generation Mechanisms for Wireless Sensor Networks. In *Proceedings of the 46th Global Communications Conference (Globecom '03)*, San Francisco, CA, USA, December 2003.

[ZK04] M. Zúñiga Zamalloa and B. Krishnamachari. Analyzing the Transitional Region in Low Power Wireless Links. In *Proceedings of the First Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON '04)*, Santa Clara, CA, USA, October 2004.

[ZK07] M. Zúñiga Zamalloa and B. Krishnamachari. An Analysis of Unreliability and Asymmetry in Low-Power Wireless Links. *ACM Transactions on Sensor Networks*, 3(2), June 2007.

Appendix A

Simulation Framework: Additional Material

The simulation environment introduced in Chapter 4 facilitates detailed configuration. This chapter provides an overview of the available parameters and options. In addition, simulation result files and the log file format is explained.

A.1 Simulation Parameters

Running a simulation requires configuration, which consists of setting up the simulation script as well as specifying the ns-2 options and parameters. An overview is provided in the following.

A.1.1 Basic Simulation Configuration

The simulation script introduced in Section 4.2.3 is configured via the environment variables listed in Table A.1. `NS2_SIM_BASEDIR` sets the path to the simulation environment. `NS2_SETTINGS_PATH` specifies the path at which reading the simulation setting is started. The simulation script reads, for each setting component, the first file found on the way from `$NS2_SETTINGS_PATH` up to *`$NS2_SETTINGS_PATH`*/*data*/*settings*. The other variables are used for sanity checks, specific setup of the simulation run, and result-file naming (cf. Section A.2). E.g., the area size and the number of nodes are compared with the corresponding values read from the found setting files. The slot assignment provided by `NS2_SLOTS` additionally determines the employed MAC. `SlotPassingTreeTDMA` is used for

Environment-Variable	Description
NS2_SIM_BASEDIR	Path to the simulation base directory Example: *~/treemacframework/sim*
NS2_SETTINGS_PATH	Path for starting the search for simulation setting files Example: *~/treemacframework/sim/data/settings/area_1000x1000/nodes_0100/cnt_007/tree_8/buf_200_25_75/slots_1_cch*
NS2_AREA_SIZE	Area size in meters, must match the specification of a topology's filename Example: *1000x1000*
NS2_NUM_NODES	Four digit encoded number of nodes Example: *0100*
NS2_COUNTER	Three digit encoded experiment counter Example: *007*
NS2_TREE	Value of C of the corresponding tree, leave empty for unspecified C Example: *8*
NS2_BUFFER_SIZE	Size of buffer and the percentages of lowest and highest fill level Example: *200_25_75*
NS2_SLOTS	Type and name of a slot assignment as provided by the slot assignment script Example: *1_cch*

Table A.1: Environment variables for the simulation script *Simulation.tcl*

Type II, `StaticTreeTDMA` else. `NS2_COUNTER` distinguishes between topologies with the same parameters.

A.1.2 Options of the Simulation Script

The main part of configuration is done via OTcl variables that are read from *rc_default.tcl* and the individual configuration files for the simulation run (cf. Section 4.2.3). They are used by the simulation script to set up ns-2 accordingly. Table A.2 lists all configuration variables required in Chapter 5. A variable `var` is set as the key of the global `opt` array, i.e., `opt(var)`.

Most configuration is required for the radio chip and the propagation model. Note that the receive threshold θ_{rx} and the carrier sense threshold θ_{cs} are not explicitly configured. ns-2 provides a tool for calculating θ_{rx} from the remainder of the variables, particularly the propagation model and the communication radius. θ_{cs} is determined by θ_{rx} and θ_{int} (cf. Section 4.1.3). If an encoding and a modulation class are configured, they are used to calculate bit errors that result in the simulation of packet errors. Currently, θ_{cs} is used as the noise floor during computation of the bit error rate (cf. Equations 2.3 and 2.4).

Key in `opt(.)`	Type / Unit	Description
Radio Parameters, MAC and Physical Layer		
`antennaHeight`	meter	Height of the antenna
`bandwidth`	bits/s	Communication bandwidth
`commRadius`	meter	Communication radius
`csGain`	dB	Carrier sense gain as compared to receive threshold
`encoding`	class name	Encoding scheme used to calculate packet error probabilities from the given BER
`frequency`	Hertz	Communication frequency
`macCaptureDB`	dB	Interference threshold θ_{int}
`macMaxRetries`	int	Maximum number of retry attempts r
`modulation`	class name	Modulation scheme used to calculate bit error rates
`pathLoss`	decimal	Pathloss exponent for the shadowing model
`propagation`	class name	Used propagation model
`radiochip`	chip name	Radio chip configuration file; must exist in the *config* folder of the simulation environment. Options set in the radio chip file have precedence over default values
`shadowDB`	dB	Deviation of random signal power part for shadowing model
`slotcoder`	class name	The class used for slot coding required by Type II
`txPower`	mW	Transmission power common to all nodes
Buffer Setup		
`queueHardLimit`	int	Buffer size B
`queueSoftLimitPerc`	fraction	Fraction of the soft buffer limit and its size: $\tilde{B}/B$
`queueUseAutoLimit`	bool	If true, the buffer size provided by the settings file is used to overwrite `queueHardLimit`
`queueUseHardLimit`	bool	If the buffer size is to be obeyed or not; if set to false, the buffer is infinite
Simulation Evaluation		
`writeConfig`	bool	Whether to write the full configuration to a file
`writeLogfiles`	bool	Whether to create log files
`writeSetting`	bool	Whether to create a settings file

Table A.2: OTcl variables for simulation configuration via *Simulation.tcl*

A.1.3 OTcl Variables

The implementation of the data-collection protocol presented in Section 4.3 is designed to be highly configurable via OTcl. Table A.3 provides a summary of all available variables, including their type and a short description. Some of them are initialized by *Simulation.tcl* using the simulation script options in Table A.2.

OTcl Binding	Type	Description
Phy/WirelessPhy/WSNPhy		
sleepAtStartup_	bool	Determines, if the transceiver is initially off
preambleLength_	bytes	MAC packet overhead in bytes (MAC header + preamble + postamble)
Queue		
limit_	int	Size of the queue in number of packets
Queue/TreeRouting		
obeyLimit_	bool	If set to false, packets can be queued beyond the specified queue size
softLimit_	int	If obeyLimit_ is set to true, this value is used as the buffer soft limit $\tilde{B}$
Queue/TreeRouting/Prio		
ackWaveThrough_	bool	If set to true, ACKs are not queued, but sent directly to the MAC
Mac		
bandwidth_	bandwidth	Bandwidth of the Wireless Communication
delay_	time	Simulated delay between packets leaving the MAC and arriving at the link layer
Mac/TreeTDMA		
maxRetries_	int	Maximum number of transmission retries
maxSlotSkipping_	int	Maximum number of skipped slots in flow control
nSlotsPerFrame_	int	TDMA round length
slotTime_	time	Overall size of a sending slot, excluding the initial guard interval
slotGuardTime_	time	Length of the initial guard interval
txrxTurnaroundTime_	time	Time required by the transceiver to switch between sending and receiving
rxIdleTimeout_	time	Time a node waits for a reception; after expiration, the transceiver is switched off
firstSlotTime_	time	Initial MAC setup time
macPktOverhead_	bytes	Preamble and postamble for transmission
sendKeepalives_	bool	If set to true, keepalive packets are sent
Mac/TreeTDMA/Static/SlotPassing		
maxPiggybackedSize_	bytes	Maximum number of additional bytes allowed for piggybacking slots
maxStandaloneSize_	bytes	Maximum number of bytes allowed for sending slots in standalone packets
fwdPerKept_	bool	Determines the number of forwarded slots per kept one
Agent/TreeRouting/Static		
maxHistSize_	int	Size of packet history
Agent/TreeAppAgent		
payloadSize_	bytes	Simulated payload of a data packet in bytes; including application and routing header

Table A.3: Class variables available via OTcl

```
cntRx_ 358 2 21 61 26 15 17 52 39 100 36 54 39 108 7 20 94 37 6 2
cntTx_ 358 2 21 61 26 15 17 52 39 100 36 54 39 108 7 20 94 37 6 2
cntTxAcks_ 358 0 0 21 0 0 0 20 0 41 0 19 0 43 0 0 45 0 0 0
cntRxAcks_ 0 2 21 40 26 15 17 32 39 59 36 35 39 65 7 20 49 37 6 2
timeDisabled_ 0 4.335916911570501 40.235916822406281 ↙
   75.135916842003013 49.83591676093549 29.335916816446183 ↙
   32.035916773527219 60.135916755052698 74.235916777235104 ↙
   111.93591690553001 68.635916777909472 65.73591688348057 ↙
   74.035916731991648 123.13591683077306 14.235916892199578 ↙
   38.035916834193699 92.635916866046045 69.735916759807253 ↙
   12.03591685770416 4.5359169283514733
cntPktCollected_ 358 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
cntPktCreated_ 0 2 21 19 26 15 17 12 39 18 36 16 39 22 7 20 4 37 6 2
```

■ **Listing A.1:** Part of an evaluation file of a simulation run

A.2 Format of the Simulation Result Files

For each simulation run, four result files are generated in the directory in which the simulation script is executed. They follow a naming scheme based on the environment variables explained in Section A.1.1 in combination with suitable prefixes. Using the values from Table A.1, an example for such a base name would be *area_1000x1000,-nodes_0100,count_007,tree_8,buffer_200_25_75,slots_1_cch*. Note that this naming scheme reflects the path structure introduced in Section 4.2.2.

The main result file has the extension *.dat*. For each node in the network, it includes the values of counter and state variables produced during a simulation. The file contains one line for each of those variables. A line commences with the variable name and is followed by the values of each node, sorted ascendingly by the node identifiers and separated by whitespaces. Listing A.1 shows an extract of such a file. The counter `cntPktCreated_` gives information about the number of packets created per node, whereas `cntPktCollected_` tells the number of packets collected. Note that this value is nonzero only for the sink node with identifier 0. The file also shows that no packets have been lost, as the counters `cntTx_` and `cntRx_`—indicating the number of sent and received packets—do not differ for individual nodes. Table A.5 gives an overview about all available variables and the classes they are defined in. Documentation can be found in the corresponding C++ header files.

The remaining result files are as follows. One result file (*.set*) contains the paths to the used setting files. Another one (*.cfg*) includes the complete OTcl configuration, i.e., all key-value pairs of the `opt` array. The creation of both files can be switched on

```
0|9.8|3:5|phy wakeup
7|9.8|3:5|trying to get a new tx pkt
7|9.8|3:5|phy wakeup
7|9.802|3:5|tx TreeAppData to node 0, txtime 0.0213, attempt 1, skip 0
0|9.8233|3:5|recv new pkt from 7 in TreeTDMA::sendUp()
0|9.8243|3:5|tx TreeRoutingACK to node 7, txtime 0.0116667, attempt ↙
   1, skip 0
0|9.8359|3:5|sleep
0|9.8359|3:5|phy sleep
7|9.8359|3:5|recv good routing ack
7|9.8359|3:5|recv new routing ack from 0 in StaticTreeTDMA::sendUp()
7|9.8359|3:5|sleep
7|9.8359|3:5|phy sleep
```

Listing A.2: Part of a log file

or off via `opt(writeConfig)` and `opt(writeSetting)` in the OTcl configuration (cf. Section A.1.2). A report file (*.rep*) gives a brief summary of the simulation. It includes information about simulated runtime, the number of created and collected packets, and which nodes have not been able to clear their buffers.

In addition, log files are created for the different protocol layers (cf. Section 4.3.6), if `opt(writeLogfiles)` is set to true (cf. Table A.2). Their names can be configured using the variables shown in Table A.4. These files provide a detailed trace of the simulation, so that they can be used for a deeper analysis. However, log files become very large and slow down simulation time, so that it is recommended to disable them, unless they are required. An extract from such a log file is shown in Listing A.2. Each line has four fields, separated by the pipe symbol `|`. The first field holds the node identifier, the second one shows the simulation time (in seconds), and the third one provides the current TDMA round and slot. It follows the actual logging message. The example traces the communication between node v_7 and v_0 (the sink). First, both nodes wake up in slot 5 of round 3, and v_7 acquires a new packet for transmission. Line 4 shows the details about the sent packet, e.g., the packet type and the time required for transmission. The packet is received by v_0 as indicated in line 5. It follows the transmission of the acknowledgment. Note that v_0 goes to sleep after the packet has been completely sent, and v_7 reports completed reception almost simultaneously. The propagation delay applied by ns-2 is so small that it cannot be inferred from the simulation time in the second field. After v_7 has sent the packet up (to the link layer), it sleeps, i.e., it turns off its transceiver.

OTcl Variable	Default Value	Comment
log(app)	app.log	Application messages, esp. created and collected messages
log(ll)	linklayer.log	Link layer logging
log(mac)	mac.log	Messages produced by all classes of the MAC layer, includes a detailed tracing of all actions
log(queue)	queue.log	Log messages produced by the buffer, includes a trace of each buffer's fill level
log(root)	root.log	All loggings that are not bound to a specific layer
log(routing)	routing.log	Routing information

Table A.4: Log-file configuration

Evaluation Counters and State Variables		
Queue/TreeRouting		
avgFillLevel_	cntDequeued_	cntDropped_
cntLimitExceeded_	cntQueued_	maxFillLevel_
Queue/TreeRouting/Prio		
cntAcksDropped_	cntAcksQueued_	cntAcksRecv_
Mac/TreeTDMA		
cntMixedSlotsUsed_	cntRecvSlotsUsed_	cntRx_
cntRxBroken_	cntRxCaptures_	cntRxCollisions_
cntRxDuplicates_	cntRxIgnored_	cntRxInterrupted_
cntRxKeepalives_	cntRxOverhearings_	cntRxRetriesExpired_
cntRxSentUp_	cntRxTimeouts_	cntSendSlotsNoPkt_
cntSendSlotsSkipped_	cntSendSlotsUsed_	cntSkipRequests_
cntSleepSlotsUsed_	cntTx_	cntTxKeepalives_
cntTxRetries_	cntTxRetriesExpired_	timeDisabled_
timeLastTx_	timeOn_	timeRx_
timeTx_		
Mac/TreeTDMA/Static		
cntRxAcks_	cntRxAcksSentUp_	cntRxAckTimeouts_
cntRxKeepaliveAcks_	cntTxAcks_	cntTxKeepaliveAcks_
Mac/TreeTDMA/Static/SlotPassing		
cntFwdSlots_	cntFwdSlotsTx_	cntFwdSlotsAcked_
cntNewSendSlots_	cntNewSendSlotsTx_	cntNewSendSlotsAcked_
cntNewRecvSlots_	cntRxSlotPassings_	cntRxSlotPassingAcks_
Agent/TreeAppAgent		
cntPktCollected_	cntPktCreated_	

Table A.5: Evaluation counters in alphabetical order, grouped by class membership

Appendix B

Scripts for Creating Simulation Settings

For the creation and basic analysis of simulation setups, various scripts have been developed. They are introduced in this chapter. Detailed information on script usage can be obtained by executing the corresponding script with options `-man` or `-help`.

B.1 Topology Generation

The topology scripts introduced in the following produce or use a topology description file with common format: For each node in the network, its identifier, starting with 0, and position (x-, y-, z-coordinates in meters) are written to a new line and separated by single whitespaces.

createGridTopology.pl produces a square grid topology with a specified distance between horizontally and vertically adjacent nodes.

createDenseTopology.pl creates a connected, randomized grid topology with a specified density (in the center of the network). Based on a unit grid, nodes are distracted randomly from their initial position. Three different random distributions for node displacement are supported: normal, triangular, and uniform distribution. Their confidence, which is the probability with that a node does not leave its unit square, can be customized. Before nodes are distracted, the initial grid is scaled to approximate the desired node density with regard to the communication radius.

createRandomTopology.pl creates a connected, random topology. Nodes are randomly arranged in a given area using a uniform distribution. The sink is placed in the center of the area, unless otherwise specified. A placement rectangle is then set up

with edge length according to the double communication radius and its center being the sink. The next node is randomly placed inside this rectangle, and, depending on its position, the rectangle is enlarged. This enhancement increases the chance of connectivity in a sparse network and thus improves script runtime, as repair has to be applied, if the topology is not connected.

createCircularTopology.pl generates a circular topology. The script places circles of C^ℓ nodes around the sink, where the radius of the circles increases with ℓ. In the last circle, there may be less nodes. The purpose of this placement strategy is to create topologies that can be used to obtain minimum-depth trees.

calcDensity.pl calculates and outputs for each node of the network the number of nodes inside its communication radius (including the center node). From this, the network density as defined in Section 2.1.2 can be calculated.

calcDensityStat.pl calculates and outputs a statistical analysis of network density. For each node in the network, the script determines the number of nodes inside its communication circle (including the node itself) and generates the following statistical data (in this order): number of nodes; minimum, maximum, and average number of nodes in a communication circle, standard deviation of the average, and the median.

calcDistance.pl calculates and outputs the distances between each two nodes.

showTopology.pl displays the network defined by the specified topology file or creates a gnuplot or tikz file from it. The script is capable of marking individual nodes and showing the tree, if a tree specification file is provided.

editTopology.tcl is a network graph editor for existing topologies. Nodes can be moved via drag and drop.

B.2 Tree Construction

The tree scripts introduced in the following produce a tree description file with common format. For each node in the network, its identifier, depth, and the identifiers of its children in the tree are written to a new line and separated with single whitespaces.

createTree.pl creates a (routing) tree rooted in the sink for the specified topology using a breadth-first search. Starting with the sink at tree level 0, the script determines the closest nodes inside communication range and makes them the sink's children. If no more children can be added to the sink, the script continues with the newly added nodes at tree level 1. From all nodes that are not yet connected to the tree, the one with closest distance to a level 1 node is added as that node's child. If all nodes inside communication range have been added as children of a level 1 node, the algorithm proceeds in the same fashion with the newly added level-2 nodes. This procedure continues, until all nodes have been added to the tree. If required, the script can be configured to restrict the maximum number of children per node. In this case, unconnected nodes are added to the closest parent that has not reached the maximum number of children so far. Additionally, the script proceeds to the next tree level as soon as new parent-child connections become impossible in the current level.

calcBalance.pl computes the balance of the specified tree using the Chebyshev Sum [DH03] as metric.

B.3 Buffer Initialization

The buffer creation script introduced in this section produces a buffer description file. For each node in the network, its identifier and initial buffer fill level are written to a new line and separated by a single whitespace.

createBufferLevels.pl creates random buffer fill levels for each node (except the sink) of the given topology. The script takes the buffer size and the minimum and maximum fill level (in percent) as input. The fill level for each node is then drawn independently from a uniform distribution with values inside the given bounds.

B.4 Slot Assignment

The slot assignment scripts introduced in the following produce an assignment description file with common format: For each node in the network, its identifier and its (sending) slots are written to a new line and separated with single whitespaces. Slots are represented as integers starting from 0. The sink is assigned slot -1 to explicitly indicate that it has no slot.

assignSlots-type1-cch.pl performs a 2-hop node coloring using the Color Constraint Heuristic (CCH). The script calculates the interference radius from γ (cf. Section 2.1.3) and the communication radius R_{com} in order to produce a completely collision-free schedule. This is achieved by computing the 1-hop neighborhoods from the interference radius and using those to construct the required 2-hop neighborhoods.

assignSlots-type1-tcch.pl creates a 3-hop link coloring based on both a topology and a corresponding tree using the Color Constraint Heuristic (CCH).

assignSlots-type2.pl creates a Type II slot assignment using the specified tree. Via a depth-first search in that tree, slots can be assigned in either ascending or descending order from leafs to the sink.

assignSlots-type3.pl performs a Type III slot assignment by doing a depth-first search based on the specified tree. The direction of assignment, i.e., either ascending or descending order from leafs to the sink, can be configured. Additionally, the script offers support for taking initial buffer fill levels into consideration.

assignSlots-type3-reduced.pl establishes an unoptimized SPR slot assignment as described in Section 3.3. Along with the tree, the path reuse number κ has to be provided. All paths consume exactly κ slots, which are always assigned in descending order from leafs to the sink.

assignSlots-type3-condensed.pl establishes an optimized SPR slot assignment. Along with the tree, the path reuse number κ has to be provided. Paths shorter than κ consume only as many slots as they are long. Slots are always assigned descendingly from leafs to the sink.

Appendix C

Contents of the DVD

Attached to this thesis is a DVD containing the complete simulation environment, the source code, the used simulation settings, and extracts of the simulation results. In the following, the directory structure of the DVD will be pointed out. Furthermore, installation and setup of the simulation framework will be explained briefly.

C.1 Directory Structure

The DVD contains five directories with the following content. *ns-2* contains the unmodified ns-2 package used for development of the simulation framework and the required patches. The tools for creating simulation settings as described in Appendix B can be found in *tools*. The folder *common* contains the logging component and the modulation/encoding scheme for simulating bit error rates. The simulation environment, including the ns-2 source code for the data-collection protocol, resides below the *treemacframework* folder. The results obtained from all simulation runs are stored in *results*.

C.2 Installing and Setting up the Simulation Environment

Firstly, the folders *treemacframework*, *common* and *tools* must be copied into an arbitrary directory. Secondly, the environment variables as described in Section A.1.1 have to be set in order to run a simulation. Additionally, *treemacframework/sim* and *tools* should be added to the `PATH` environment variable.

The simulation environment has been developed for version 2.31 of ns-2. It is included in the archive *ns-allinone-2.31.tar.gz* and must be installed first. Detailed

information about this can be found on the ns-2 homepage. Next, the patch *ns2-treemac.patch* has to be applied in the subdirectory *ns-2.31*. It also affects the ns-2 *Makefile* in order to make *$NS2_TREEMAC_EXT/treemacframework/ns2/cc* and *$NS2_TREEMAC_EXT/tools* available, including all subdirectories. Hence, the variable NS2_TREEMAC_EXT must point to the correct directory. Finally, ns-2 must be recompiled. An example for installing and patching ns-2 is shown in Listing C.1

```
tar zxvf ns-allinone-2.31.tar.gz
cd ns-allinone-2.31
./install
cd ns-2.31
patch -p2 < /media/cdrom/ns-2/ns2-treemac.patch
export NS2_TREEMAC_EXT=~/treemac
make clean && make
```

Listing C.1: Installing and patching ns-2