# Fault-Tolerance Mechanisms for Glossy-based Wireless Communication Networks

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich


# Fault-Tolerance Mechanisms for Glossy-based Wireless Communication Networks

Semester Thesis

Alexander Dietmüller
adietmue@ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

**Supervisors:**
Romain Jacob
Prof. Dr. Lothar Thiele

2017-07-03

# Acknowledgements

I would like to thank my supervisor, Romain Jacob, for his dedicated support. His guidance and helpful critique, helped me to gain insight into the world of theoretical research and scientific writing and presentation.
Furthermore I would like to thank Prof. Dr. Thiele for the fruitful discussion during the semester, which helped to sharpen this thesis' focus.

# Abstract

Flooding-based communication with protocols such as Glossy is being used increasingly in low-power wireless communication for multi-hop networks because of its efficiency and reliability. For example, Glossy can achieve packet reception rates above 99.99% in real-life tests. However, faults like interference can still occur, causing a packet to get lost, and, up-to-date, most Glossy-based communication protocols do not take such faults explicitly into account. We address this by using the Low-Power Wireless Bus (LWB), a MAC protocol based on Glossy floods, to design fault-tolerance mechanisms. To this end, we classify the faults that can affect our LWB network, and analyze fault-tolerance mechanisms for data collection and data dissemination applications. We propose two different mechanisms for data collection and present general evaluation results, considering one data-collecting sink as well as multiple (redundant) sinks. Furthermore, we briefly discuss three approaches to fault-tolerant data dissemination. Glossy and LWB are simple protocols with nice statistical properties, which helps to model and analyze fault-tolerance mechanisms independently of network topology and traffic patterns. Additionally, the proposed mechanisms for data collection are able to leverage Glossy and LWB to increase efficiency. This general classification and fault-tolerance mechanisms can be used as a foundation for future analysis of more specific use-cases, e.g. designing a fault-tolerance mechanism tailored for a specific network topology. Additionally, an investigation on the assumptions we make for our analysis could provide further insight into the modeling of Glossy-based networks.

# Contents

Chapter 1

# Introduction

In recent years, network flooding based on *Glossy* has become an energy efficient and reliable method of communication for low-power wireless multi-hop networks, such as wireless sensor networks [1]. In network flooding, a sender broadcasts a packet to all receivers in range, which in turn repeat the packet, which thus gets *flooded* to the whole network.

Up-to-date, the design of Glossy-based protocols has focused on increasing the nominal reliability and Glossy itself can achieve packet reception rates above 99.99% in real-life tests. Nevertheless, packets can get lost, e.g. because of external interference, which can happen anywhere at any time. This unpredictable nature makes fault-tolerance mechanisms essential for reliable packet delivery [2].

This project investigates the design, modeling, and analysis of fault-tolerance mechanisms based on the *Low-Power Wireless Bus (LWB)* [3], a MAC-layer protocol using Glossy floods as a communication primitive.

We first introduce the basic principles and properties of Glossy and LWB in chapter 2. Furthermore, we describe and motivate several assumptions we make about the network for our analysis. Following this, we classify problems which can occur in LWB-based networks in chapter 3. Next, we briefly explain the fundamentals of fault-tolerance mechanisms in chapter 4 and introduce packet reception rate and bandwidth overhead factor as metrics we use for evaluation. In chapter 5, we focus on the data collection use-case. We analyze network with a single sink as well as multiple redundant sinks and present fault-tolerance mechanisms based on LWB and develop models to evaluate them. Finally, in chapter 6 we turn to data dissemination and briefly discuss several approaches in the context of LWB and Glossy based-networks.

Chapter 2

# Protocols and Assumptions

In this chapter, we first introduce the protocols on which this work is based:

- *Glossy* [1], a flooding communication primitive based on synchronous transmissions.
- *Low-Power Wireless Bus (LWB)* [3], a MAC-layer protocol based on Glossy.

Subsequently we discuss the statistical properties of Glossy-based communication and formalize the assumptions we consider in this work, e.g. regarding packet reception probability.

## 2.1 Glossy

Glossy is a protocol that uses concurrent transmissions to implement efficient flooding. This is possible by synchronizing the clock on all nodes during the flood. A synchronization in the µs range is achieved and causes the concurrent transmissions to interfere constructively, enabling reception of packets even if many nodes are sending simultaneously.

As a communication primitive, Glossy does not address issue like medium access, which is handled by LWB, which we describe in section 2.2.

### 2.1.1 Core Functionality

First, we discuss the core functionality of Glossy. Every *flood* in Glossy starts with one node, the initiator, sending a packet and thus *initiating* a flood. In the following, the flood is driven by radio events only: As soon as another node has successfully received a packet, it re-send the packet.

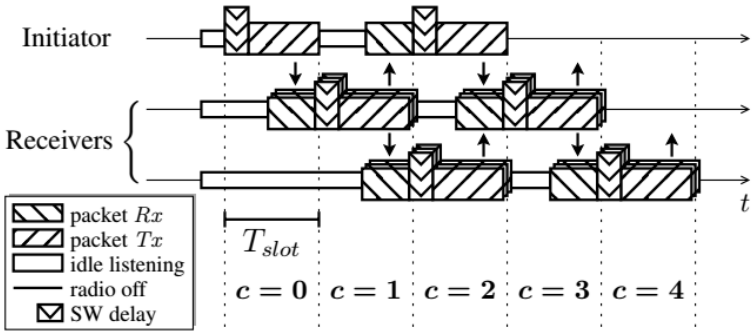



Figure 2.1: Example of a Glossy Flood with $N = 2$ transmissions per packet. E.g. the initiator does not transmit the packet at time $c = 3$, since it has already transmitted the packet $N = 2$ times. Reprinted from [1].

**Time Synchronization** As mentioned before, Glossy relies on synchronous, also called concurrent, transmissions – it is therefore essential to synchronize the different nodes. This is achieved by adding timing information: The initiator adds both its current time and a relay counter (initialized with 0) to a packet. Before each retransmission the relay counter is incremented. Every node can estimate the duration of a single transmission based on local radio information (e.g. time of interrupts). Combining the duration for a single transmission with the relay counter, a node can determine how much time has passed since the flood started. By adding the passed time to the initiator time stamp, every node can synchronize with initiator whenever a packet is received.

### 2.1.2 Mechanisms to Increase Reliability

Glossy increases transmission reliability by transmitting a packet up to $N$ times instead of once (Figure 2.1). Glossy uses $N = 3$ as a default setting and achieves reliability over 99.99% in real-life tests. [1]

## 2.2 Low-Power Wireless Bus

The Low-Power Wireless Bus (LWB) is a MAC-layer protocol that uses Glossy floods for communication.

Using Glossy floods, every packet reaches each node, which makes routing unnecessary and essentially turns the multi-hop network into a structure similar to a shared bus.

### 2.2.1 Core Functionality

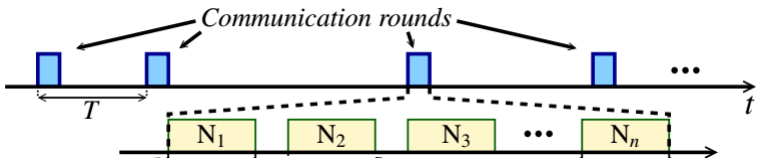



Figure 2.2: Schematic of LWB rounds consisting of $n$ slots (Glossy-floods). Reprinted from [3].

LWB uses a node, designated at design time, as *host* to compute and distribute a communication schedule for the other nodes to organize network access. The network communication is organized in *rounds* consisting of several *slots* (each slot in a LWB round is complete Glossy flood). Between rounds, nodes sleep to save energy.

At the end of round $k$ is a slot dedicated for the schedule-packet of round $k+1$. This packet is sent by the host node and contains two pieces of information:

- The starting time for round $k + 1$, defining the sleep time between the current and next round.
- The slots in the round and which nodes sends a packet in each round.

During the round, nodes follow the schedule, i.e. initiate a flood if its their turn, otherwise receive and retransmit, following the procedure of Glossy. If a node does not receive the schedule, it does not participate in the round.

**Data Streams** Nodes inform the host of their demand to send data by registering periodic *data streams*. Each data stream has a specified *Inter-Packet Interval (IPI)*, describing the time between two packets.

### 2.2.2 Mechanisms to Increase Reliability

First of all, the schedule is sent again at the beginning of each round to add redundancy. For our analysis, we do not consider this additional schedule, since it is not essential for LWB. Furthermore LWB specifies the following additional behavior for nodes missing a schedule: It starts listening earlier in the next round in case the schedule was missed due to synchronization problems.

## 2.3 Statistical Properties and Assumptions

In this section, we state the statistical properties of packet reception in Glossy and describe and motivate additional assumptions.

### 2.3.1 Subsequent Reception of Packets is Independent

It has been shown that the reception of packets by a given node in subsequent Glossy floods can be modeled as independent and identically distributed Bernoulli-trials (fig. 2.3a) [4]. A packet is received during a Glossy flood with the probability $p$, which mainly depends on

- The size of the packet.
- The number $N$ of retransmissions per node (see section 2.1).

For the sake of simplicity, we assume $p$ to be equal for all packets, i.e. we assume that all packets have the same size and we use the same number of retransmissions $N$ for all packets.

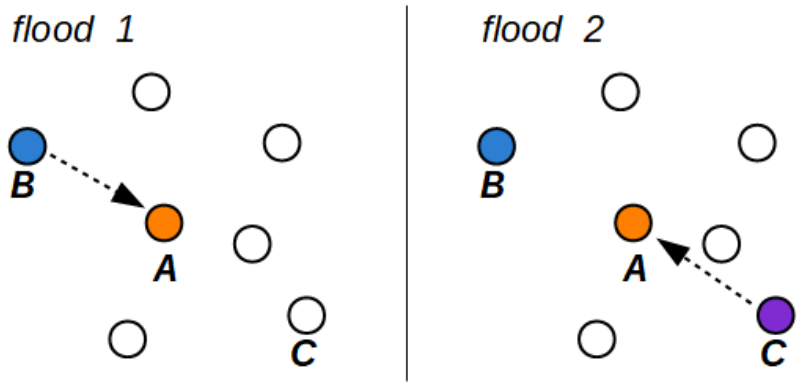
### 2.3.2 Packet Reception is Assumed to be Independent from Number of Initiating Nodes

In 5.3.2 we analyze several nodes initiating a flood at the same time. This can be interpreted as a flood which is initiated by a single virtual node sending its packet to all real initiators (fig. 2.3b). Since the resulting flood is identical, we assume that a packet is received with probability $p$, regardless of how many nodes initiate a flood.

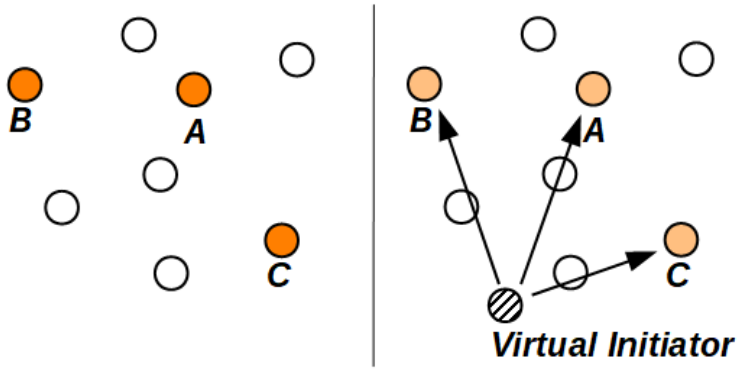
### 2.3.3 Packet Reception by Different Nodes is Assumed to be Independent

One of the strengths of LWB (and Glossy) is the independence of topology. We do not want to introduce this dependency again for our analysis, therefore we assume that the reception of a packet by different nodes in the network is independent (fig. 2.3c). In particular, this applies for the LWB schedule: We assume that the reception of the schedule packet is independent for each node.

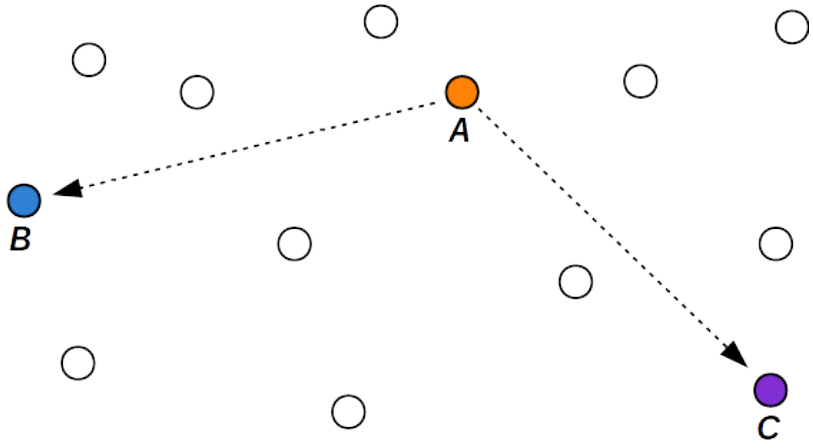
We expect that the validity of this assumption depends on network topology, especially where the sending/receiving nodes are located in the network. A closer analysis of this assumption is of great interest, but finding a general model might be difficult or nearly infeasible.




(a) $B$ sends a packet to $A$ in flood 1, $C$ sends a packet to $A$ in flood 2. $A$ receives each packet independently with probability $p$.




(b) Left: $A$, $B$ and $C$ initiate a flood together. Right: A virtual node initiates the flood by sending a packet to $A$, $B$ and $C$. The resulting flood is identical in both cases.




(c) $A$ sends a packet. We assume the reception of this packet by $B$ and $C$ to be independent.

Figure 2.3: Statistical Properties and Assumptions

### 2.3.4 Packet Reception in LWB is Assumed to be Independent

To simplify modeling, we assume that if $A$ sends $i$ packets to $B$ using LWB, the reception of all packets is independent.

While this would be valid for Glossy, the scheduling of LWB introduces some difficulties. As described, a node only participates in a round, if it has received the round schedule. Therefore, if the $i$ packets are transmitted during the *same* round, all $i$ packets depend on the same schedule and are not independent.

Nevertheless, we assume independence and in the following we show that this assumption is not completely off, but underestimates the probability to receive packets deterministically.
First of all, a *single* packet sent by $A$ is received by $B$ if:

- $A$ receives the schedule (and sends the packet).
- $B$ receives the schedule (and listens for the packet).
- $B$ receives the packet sent by $A$

All packets are received independently with probability $p$, leading to a total probability of $p^3$ that $B$ receives $A$'s packet. (If one node is the host, it knows the schedule, therefore a packet is received with probability $p^2$.)

If $A$ sends and $B$ receives multiple packets during the same round, they all depend on the same schedule. If both $A$ and $B$ receive the schedule, the $i$ packets themselves are received independently. Otherwise, no packets can be received. Therefore, the probability to receive all $i$ packets is $p^{2+i}$ ($p^{1+i}$ if one node is is the host).

Contrary to this, if we assume all $i$ packets to be independently received, the probability to receive all packets is $p^{3i}$ ($p^{2i}$).

Comparing the results, if $i > 1$, then $p^{3i} < p^{2+i}$ ($p^{2i} < p^{1+i}$), which means that the probability that $B$ receives $i$ packets sent from $A$ in a single round decreases if we assume independence.

### 2.3.5 Discussion

While the statistical properties of packet reception in Glossy are nice, we require additional assumptions to simplify modeling of packet reception in various situations we encounter.

Independence of network topology is one of the general strengths of Glossy and LWB, and with the assumptions present, we are able extend this strength to our analysis, which allows us to provide general, topology independent, models and results in the following chapters.

Investigating the assumptions further is out of scope for this work, but might provide valuable insights for the modeling of Glossy and LWB-based communication and about the validity of our work.

Chapter 3

# Classification of Faults, Errors and Failures

The aim of this chapter is to describe possible mis-behaviors or problems that can occur in LWB. In the literature, such problems are classified into faults, errors, and failures [5]. First, we state the general classification. Next, we take a closer look at faults, errors and failures as they can occur in a wireless multi-hop network using LWB as communication protocol and model their relationship with fault trees.

Finally, recalling the assumptions made in chapter 2 we use the fault-tree analysis to model the lifetime of each packet independently with a finite-state machine to illustrate at which points problems occur and where fault-tolerance mechanisms come into play.

## 3.1 General Classification

*Faults*, *Errors* and *Failures* are defined below based on [5]. Figure 3.1 illustrates the relationship between them.


Figure 3.1: The relationship between faults, errors and failures.

**Definition 3.1 (Fault)** *A Fault is an underlying problem, e.g. interference. A fault can lead to an error.* ◇

**Definition 3.2 (Error)** *An Error is an indeterminate or wrong state of the system. Being in an error state can lead to a failure of (a part of) the system.*◇

**Definition 3.3 (Failure)** *A Failure is the observable* result *of an error, e.g. a packet is lost.* ◇

## 3.2 Faults, Errors and Failures in LWB

A network is not 100% reliable, at any point in time a packet can get lost. We call loosing a packet sent by an application *packet failure*. Nevertheless, a packet failure might be corrected, e.g. by re-sending. Only a application packet which is lost and cannot be recovered is definitely lost, which we consider to be a *network failure*.

In this section, we first take a look at network failures before analyzing packet failures more closely.

### 3.2.1 Network Failure

It is obvious that in order to definitely loose a packet, the system must be in the state of having lost a packet. Using the terminology introduced in section 3.1, a packet failure is an error state of the network, which can lead to a network failure.

The goal of a fault-tolerance mechanisms, which we describe in more detail in chapter 4), is to prevent the transition from error to failure. Therefore, in the context of this analysis, a non-existing or not-working (e.g. because limits are reached) fault-tolerance mechanism is an error state of the network as well.

And in conclusion, a network failure occurs if both a packet failure happens and the fault-tolerance mechanism cannot handle it (fig. 3.2a).

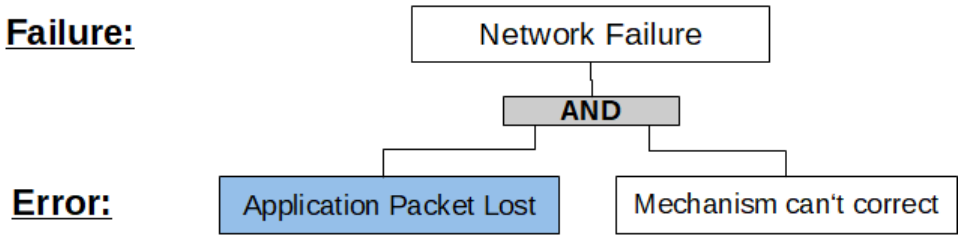
### 3.2.2 Packet Failure

In this section we model the previously introduced packet failure by considering faults and errors leading to the loss of an application packet.

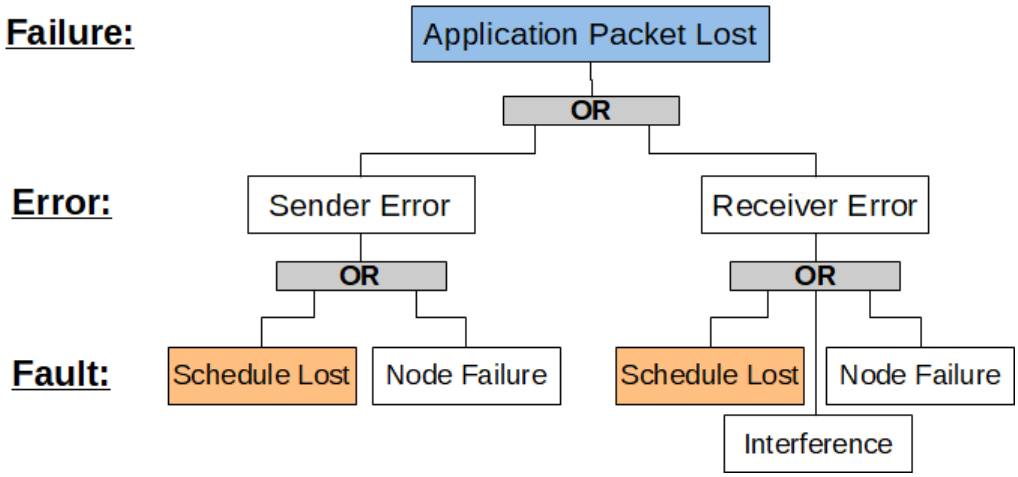
First of all it is important to consider that LWB does not only send application packets. To organize the network, the schedule has to be sent at the end of every round (see section 2.2). A schedule packet can also get lost, which we call *schedule failure*. Therefore, in the following we analyze both packet and schedule failures.

**Faults and Errors leading to Packet Failure** An application packet is sent from one node to (one or several) other nodes. Therefore, two fundamental errors are possible: Either the packet is not sent, i.e. a sender error happens, or the packet is not received by its indented recipients, i.e. a receiver error happens. A receiver error can have the following underlying faults:

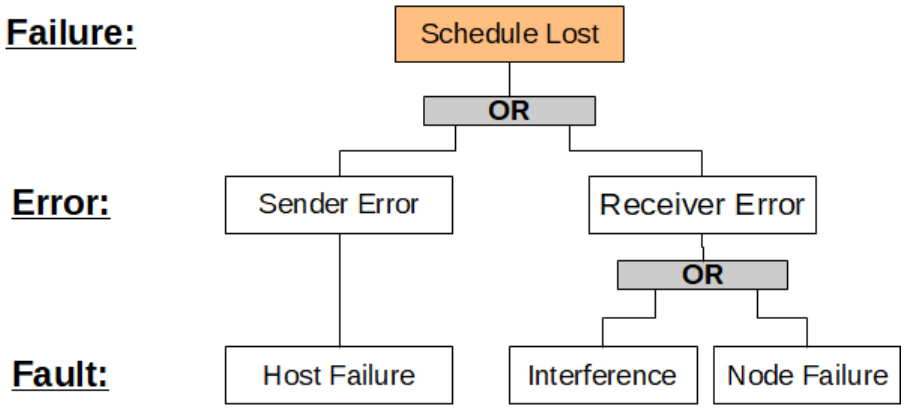
- Interference.




(a) If a packet is lost and there either is no fault-tolerance mechanism or the mechanism cannot correct the loss, the packet is lost for good, which we consider a *network failure*.




(b) An *packet failure* is a lost application packet, which can either be caused by a sender error or receiver error. A sender error can happen if either the sending node does not receive the schedule (schedule failure) or if the sending node fails. A receiver error can happen because of interference, if the schedule has not been received or if the recipient node fails.




(c) A *schedule failure* is a lost schedule packet. This equals 3.2b, except for the fact that a schedule packet does not require to be scheduled.

Figure 3.2: Fault trees for network-, schedule packet- and application packet failures. (The coloring highlights connections between fault trees.)

- Node failure, e.g. a empty battery.
- The receiver has not received the schedule (schedule failure).

A sender error can either be caused by:

- Node failure.
- The sender has not received the schedule (schedule failure).

These dependencies can be illustrated with a fault tree (fig. 3.2b).

**Host Node** The host computes the schedule, therefore at the host node a schedule failure cannot happen. If the host node either sends or receives a packet, the respective schedule failure can be disregarded.

**Schedule Failure** Schedule failures can cause packet errors, therefore it is useful to analyze them as well. Since LWB is based on Glossy and Glossy doesn't treat the schedule packet any different, the fault tree for schedule failure looks just like the fault tree for application packets, with one important distinction: The sending of the schedule does not depend on any other schedule, it is always sent by the host at the end of each round. Therefore, a sender error only occurs if the host node fails (fig. 3.2c).

### 3.2.3 We Only Consider Interference

One of the reasons for packet failure in the previous section is the failure of a node, which in turn can have many underlying faults again, e.g. a broken radio, empty batteries, etc. Because of the variety of underlying faults, modeling node failure is out of scope of this work. Furthermore just by modifying our communication protocol, there is not much to do about node failures.

Therefore we do not consider node failure for the rest of the analysis, which in turn simplifies the presented fault trees for packet and schedule failure (fig. 3.3). The only fault remaining fault is interference.

## 3.3 Modeling with a Finite State Machine

In the previous section, we model the relationship of faults, errors and failures in LWB with fault trees. Based on this analysis, we are able to model the 'lifetime' of a packet with a finite-state machine (FSM, fig. 3.4) and because of the assumptions made in chapter 2, the FSM for each packet is independent of the FSMs of other packets.

Successful reception of a packet depends on reception of schedule[1] and packet. Both can get lost due to interference, causing the FSM to enter an error state (packet failure). Finally, a network failure is a trace ending in the failure state and, as mentioned in section 3.2.1, the objective of a fault-tolerance mechanism is to prevent the transition into the failure state.

In conclusion, the FSM illustrates at which points interference can cause our system to enter an error state and where fault-tolerance mechanisms come into play.

## 3.4 Conclusion

In this chapter, we use introduce faults, errors, failures and their relationship in general. Based on this analyze faults, errors and failures in LWB-based networks using fault trees. We introduce the following terms:

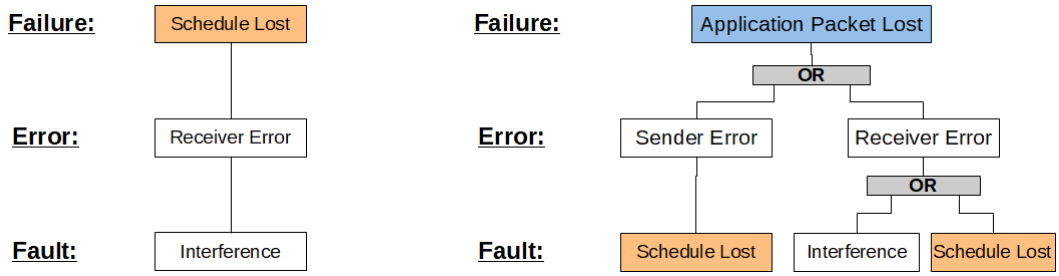
- network failure: A packet is lost and cannot be recovered.
- packet failure: An application packet is lost.
- schedule failure: A schedule is lost.

A network failure is caused by a packet failure, if there is no fault-tolerance mechanism which can recover the packet. a packet failure can be caused by interference or by schedule failure, which again can be caused by interference[2].

Based on the fault-tree analysis and the assumptions on independence made in chapter 2, we present a finite-state machine modeling the 'lifetime' of each packet independently, illustrating at which points interference can cause the system to enter an error state and where fault-tolerance mechanisms come into play.

[1] If the host is sending or receiving the packet, only the remaining node must receive the schedule. Otherwise both nodes need to receive the schedule.

[2] Aside from interference, node failures could cause packet loss, but we argue in section 3.2.3 why we do not consider node failure in this work.




(a) Schedule Packet Failure (b) Application Packet Failure.

Figure 3.3: Simplified fault trees exluding node failures. Otherwise equal to fig. 3.2. (The coloring highlights connections between fault trees.)

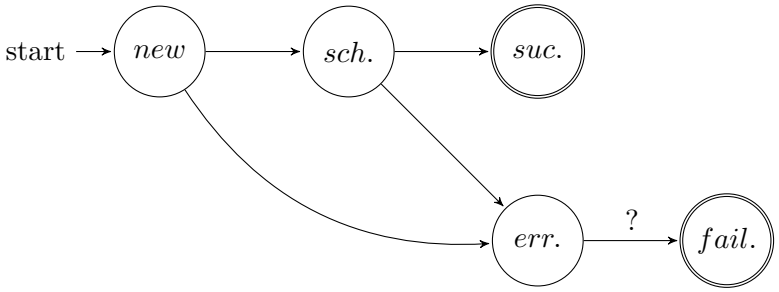



Figure 3.4: Finite State Machine modeling a packet sent from one node (source) to another (sink).
*new*: The source has generated a packet and awaits the schedule.
*sch.*: Schedule received, source sends packet, sink listens for packet.
*suc.*: Packet successfully received.
*err.*: *Error*: Either a schedule or the packet is lost due to interference.
$fail.$: *Failure*: The packet could not be recovered.
The transition from error to failure depends on the fault-tolerance mechanism, indicated by ?.
Note: If one of the nodes is the host, it does not need to receive the schedule.

Chapter 4

# Fault-Tolerance Mechanisms

In chapter 3, faults, errors and failures are introduced. We cannot completely avoid faults, in our case interference, which can happen anywhere and at any time, leading the system to an error state. At this point, fault-tolerance mechanisms come into play, which aim to stop an error from becoming a failure by the use of error correction.

Approaches to error correction can be classified in two categories[6]:

- *Forward Error Correction (FEC)*
- *Automatic Repeat Request (ARQ)*, also known as *Backward Error Correction*

In this chapter, we briefly introduce both FEC and ARQ and introduce packet reception rate and bandwidth overhead factor, the criteria we use to evaluate fault-tolerance mechanisms independent of traffic patters such as number of packets sent at a given point in time.

## 4.1 Error Correction

### 4.1.1 Forward Error Correction (FEC)

With Forward Error Correction, the sender pro-actively adds redundancy by encoding the data before sending in order to recover data even if some packets are lost. The simplest approach is a repetition code, which sends the packet $k$ times. As long as not all $k$ packets are lost, the packet is received successfully.

Glossy flooding itself is a kind of FEC: Each packet is repeated multiple times while it progresses through the network during a flood, since every node repeats every received packet N times (see section 2.1).

### 4.1.2 Automatic Repeat Request (ARQ)

ARQ dynamically retransmits lost packets. Hence, additional information is added to each packet to enable the receiver to detect packet loss and request re-sending of the packet. E.g. TCP uses sequence numbers to detect missing packets and the receiver sends acknowledgments to tell the sender which packets need to be re-sent.
Attempts to re-send packets are usually limited for several reasons:

- Packets have to be stored by the sender until they have been received correctly and storage is limited.
- The available bandwidth is limited as well. If lost packets are re-sent indefinitely, the cumulative number of packets to send grows infinitely.

## 4.2 Evaluation of Mechanisms

In order to compare different fault-tolerant mechanisms, we use the following performance criteria, which relate to the overall reliability, in terms of packet reception, and power consumption.

1. Packet Reception Rate (PRR), i.e. the probability of successfully receiving a packet by the intended recipients taking fault tolerance into account (in contrast to the probability $p$ of receiving a packet during one flood).
2. Bandwidth overhead factor, i.e. how much additional bandwidth is required to send one packet. This closely correlates with additional energy consumption. In LWB, we describe overhead by how many slots are required for a single packet compared to the ideal case, in which a packet requires exactly one slot.

Another performance criterion commonly used is the average time until successful reception, which is important for timing-dependent applications, such as real-time control. In this work, we do not focus on timing-dependent applications and therefore do not consider the average time until success further.

Both performance criteria allow us to evaluate fault-tolerance mechanisms *independent of traffic patterns*, e.g. we do not rely on the number of packets sent at a given time and instead analyze the performance on a *per packet* basis, allowing our result to be more general.

## 4.3 Discussion

In the previous sections, we described forward error correction (FEC) and automatic repeat request (ARQ). Furthermore, we introduced packet reception rate and bandwidth overhead as important performance criteria to evaluate mechanisms based on either FEC or ARQ. Next, we discuss how overhead bandwidth and packet reception rate differ in general in FEC and ARQ.

As the name implies, forward error correction has no feedback: The sender adds redundancy pro-actively to allow room for some errors. The bandwidth overhead is constant: If we repeat a packet $k$ times, it is sent $k$ times, even if it is received after the first attempt since the sender does not know (or care) whether the sending has been successful. While a constant overhead is often wasteful, it is a lot simpler to both implement and analyze.

Feedback makes ARQ-based mechanisms more dynamic, e.g. if a packet is received correctly after the first try, it does not have to be re-sent again, which lowers the bandwidth overhead if no errors happen. But additional acknowledgment packets may be needed, which both increases the required bandwidth and are possible points of failure. Both the dynamic behavior and additional packets make ARQ-based mechanisms harder to model and analyze, but potentially allow better performance, e.g. reducing bandwidth overhead and therefore energy consumption, which is desirable for low-power networks.

Chapter 5

# Fault-Tolerant Data Collection

In this chapter, we take a closer look at fault-tolerant data collection. To this end, we provide a context in form of an application case, illustrating the scenario and objective.

Next, we introduce an FEC mechanism using packet repetition for fault-tolerance, which allows us to ignore feedback to simplify analysis in a first step. We motivate why a mechanism based on repetition is suited for our application, and show how it achieves high reliability but comes with high bandwidth overhead.

Then, we consider a mechanism using feedback. As further explained in section 5.2.2, we can leverage the scheduling-based communication of LWB to extend the repetition mechanism to an ARQ mechanism, re-sending packets only if they have been lost (instead of a fixed repetitions). The re-sending mechanism achieves the same packet reception rate as the repetition mechanism and requires less bandwidth on average, which helps to conserve energy.

Furthermore, we show how additional data collecting nodes (later called *sinks*) can be added to further improve fault-tolerance and in particular, how using additional sinks benefits from Glossy-based communication. We show two mechanisms using additional sinks:

1. A passive variant without dedicated feedback from the additional sinks, which benefits from Glossy because the additional sinks overhear all packets.

2. An active variant including acknowledgments to provide feedback from additional sinks, which also benefits from Glossy: We show how shared slots can limit the overhead bandwidth factor.

For each mechanism, we discuss limitations and benefits.

## 5.1 Application and Objective

**Wireless Sensor Networks** One important function of low-power wireless sensor networks is data collection. In such a scenario, one node in the network (the sink) is either connected to the Internet, equipped with big storage capabilities (or similar) while the other nodes (the sources) have neither. Sources generate packets of data and send them to the sink. E.g. in the *PermaSense* project, all nodes are connected in a low-power wireless network, while a single node (the sink) features a GPRS-module to transmit the sensor data to a database via the Internet[7].

**Limitations and Objective** Nodes in such networks, as in *PermaSense*, have limited computational capabilities and must be available over months (or years) and therefore have to consume as little energy as possible. A major factor for energy consumption of each node is the radio on-time[4]. In conclusion, low-power communication protocols must minimize the number of transmissions while achieving high reliability. Fault-tolerance mechanisms for such networks therefore have the following objective in terms of the criteria formulated in chapter 4:

- High packet reception rate by the sink
- Low bandwidth overhead factor

**Considerations for LWB** The sink has more capability in terms of computing power, available energy, Internet access, etc., so it makes sense to let the sink be the network controller, i.e. the LWB host.

Moreover, in a network like *PermaSense*, sources continuously generate data at a constant rate, e.g. one sample every 10 seconds. Therefore, using LWB, a node registers one (or several) data streams with the required inter-packet interval once on startup and no streams are added/removed (Node failure is not considered in this work, see chapter 3). We assume that all data streams have been registered already and do not particularly analyze the startup phase, as described in chapter 2. For a dynamic environment with changing data streams, it might be necessary to adjust the mechanisms introduced in the following, but this is out of scope for this work.

**Multiple Sinks** It can be interesting to add additional sinks, e.g. multiple nodes equipped with a GPRS module. Using multiple sinks, a packet is considered as received if at least one sink has received it. We discuss additional sinks in section 5.3.

## 5.2 Single Sink

We now consider the application case described in section 5.1 with a single sink. First, we propose a simple FEC mechanism using packet repetition for fault tolerance. Second, we present an ARQ mechanism based on the repetition mechanism. Comparative analysis of the two mechanisms shows that both achieve the same packet reception rate, but the ARQ mechanism requires less bandwidth on average.

### 5.2.1 Fault Tolerance using Packet Repetition

Because of its simplicity, an FEC mechanism is an ideal starting point. In particular, we consider packet repetition, which requires no computation on the sources. This is suitable for nodes with low computational capabilities, like in our application.

#### The Mechanism

- The sink schedules every packet in $k$ subsequent rounds.
- If the packet is received in any of the rounds, the transmission was successful.

To differentiate packets and request re-sending, we require a sequence number for every packet of each data stream. The sequence number is added to the schedule, which now contains the following information for every slot:

- The source, i.e. node initiating the flood
- The sequence number of the packet to be sent

Packets themselves do not require to contain the sequence number again, since the slot in which a packet is received is tied to the sequence number of the packet.

Sources need to keep sent packets in memory along with their sequence number. Every time a schedule is received, packets which are no longer scheduled are removed.
Furthermore, we identify limits for $k$:

- The bandwidth must be sufficiently big for $k$ times the number of packets per data stream.
- The sources must be able to keep $k$ packets per data stream in memory.

**Packet Reception Rate** We model the reception of a packet during a single round with a finite state machine as described section 3.3 (fig. 5.1a). Since reception probability is independent of the round, repetitions are essentially separate events.

The probability that the sink receives a packet during a round is $p^2$: The source needs to receive the schedule (and send the packet) and the sink needs to receive the actual packet. Both packets are received independently with probability $p$, resulting in an overall probability $p^2$. We compute the packet reception rate (PRR), i.e. the probability that the sink receives the packet at least once if it is scheduled in $k$ rounds.

$$\text{PRR} = 1 \quad - \quad \underbrace{(1-p^2)^k}_{\text{loose all } k \text{ packets}}$$

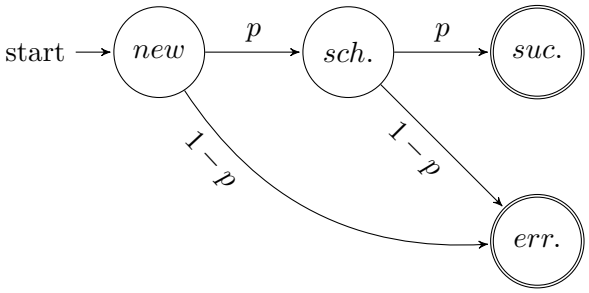
If the sink does not receive the packet at least once after it has been scheduled in $k$ rounds, it is definitely lost.

We note that for increasing $k$, the packet reception rate approaches 1 asymptotically (fig. 5.1b).

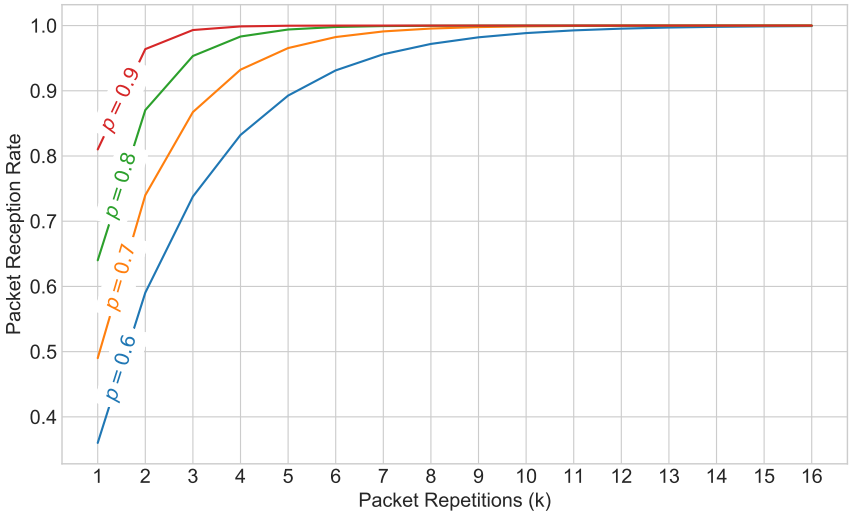
**Bandwidth Overhead Factor** The repetition mechanism requires $k$ times the data slots compared to the ideal case. The slots themselves have the same length as without repetition – while the sequence number is additional information, it is only required in the schedule and does not influence the data packet size.

**Summary & Discussion** We analyze a fault-tolerance mechanism using packet repetition suitable for sources with little computational power, as common in low-power wireless networks. While the packet reception rate approaches 1 asymptotically for increasing $k$, the bandwidth overhead increases linearly with $k$.

The proposed mechanism schedules a packet in $k$ subsequent rounds. Alternatively, a packet could be scheduled $k$ times in the same round, although with less reliability: A source only sends packets in a round, if the schedule for the round has been received. As a result, no matter how often a packet is scheduled in a single round, receiving the schedule is a bottleneck – if the schedule is lost, no packet are sent at all.




(a) FSM based on fig. 3.4 for the reception of a single packet. The source receives the schedule with probability $p$. The sink receives the packet sent by the source with probability $p$ as well. The failure state depends on the FSMs of all $k$ packets and is not included here.




(b) Packet reception rate over number of packet repetitions, plotted for several values of $p$ (probability for receiving a packet in a round). The packet reception rate approaches 1 asymptotically with increasing $k$

Figure 5.1: Fault-tolerance mechanism using packet repetition.

### 5.2.2 Fault Tolerance using Packet Re-Sending

In this section, we analyze an ARQ mechanism, which is saving bandwidth by re-sending a packet only until it has been received by the sink.

We show that this mechanism achieves the same packet reception rate as the repetition mechanism described in section 5.2.1, but uses the available information to only re-schedule a packet if necessary instead of a fixed number of repetitions. We show that for a given probability $p$ to receive a packet in a round, the average bandwidth overhead factor of the re-sending mechanism does not exceed $1/p^2$ while the packet reception rate is equal to the repetition mechanism.

**Mechanism** ARQ requires detection of packet loss and requesting of re-sending. Fortunately, both are easily achieved in LWB with the sink as LWB host.

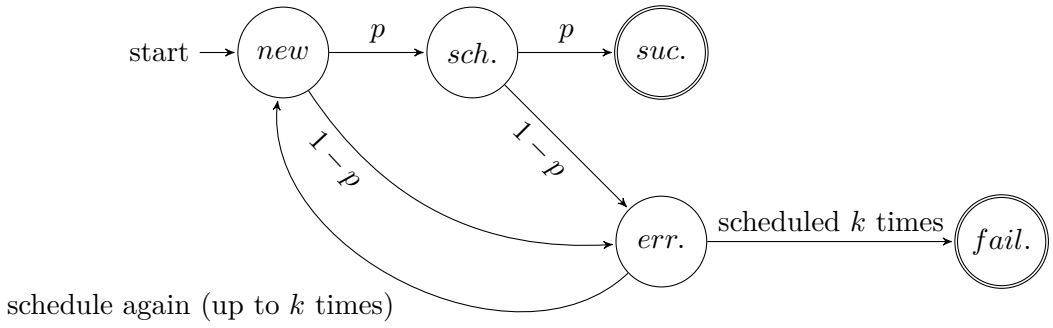
- Packet loss is detected by the sink simply by checking if a packet has been received in a slot or not.
- If a packet is lost, the sink requests re-sending by scheduling the packet again in the next round.
- If a packet has already been scheduled $k$ times, it is not scheduled again and is definitely lost.

In a worst-case scenario, i.e. a packet is only received in the $k$-th round or never, the re-sending mechanism exactly equals the repetition mechanism, but the re-sending mechanism avoids unnecessary re-scheduling if a packet is received earlier.

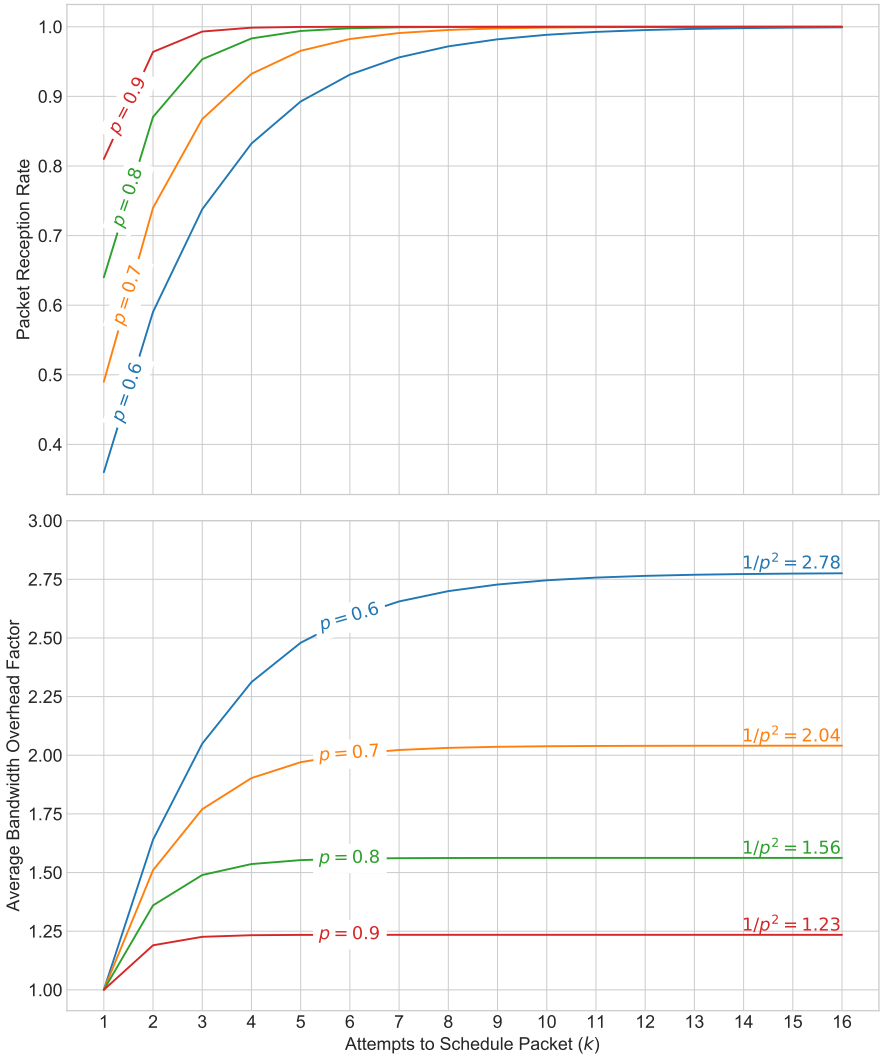
**Packet Reception Rate** Packet reception for the re-sending mechanism is modeled with an FSM. In contrast to the repetition mechanism, subsequent rounds are now connected, in particular, the process is only repeated if the packet is lost (fig. 5.2a).

The Packet reception rate equals the repetition mechanism since we only stop re-scheduling after $k$ rounds (as before) or in the case of success. The packet reception rate (PRR), i.e. the probability that the sink receives a packet after it has been scheduled in up to $k$ rounds, is:

$$\mathrm{PRR} = 1 - (1 - p^2)^k$$




(a) FSM modeling packet reception a re-sending mechanism based on fig. 3.4. The source receives the schedule with probability $p$. The sink receives the packet sent by the source with probability $p$ as well. If the sink does not receive the packet, it is scheduled again (up to $k$ times). If the packet is not received by the sink after $k$ times, it is definitely lost.




(b) Package reception rate and average bandwidth overhead factor over attempts to schedule a packet plotted for several probabilities to receive a packet in a round $p$. The packet reception rate is equal to fig. 5.1b, the average bandwidth overhead factor approaches $1/p^2$ for increasing $k$.

Figure 5.2: Fault-tolerance mechanism using packet re-sending.

**Bandwidth Overhead Factor** In a worst-case scenario, every packet is scheduled $k$ times, which equals a bandwidth overhead factor of $k$. For the average bandwidth overhead factor, we recall: The sink only re-schedules a packet if it is not received and schedules any packet up to $k$ times.

A packet requires $i$ ($i < k$) slots, if it is received in round $i$, which is equal to the probability of not receiving the packet in $i-1$ rounds and finally receiving it in round $i$. Exactly $k$ slots are used by a packet if it has not been received in the previous $k-1$ rounds, regardless of the reception in round $k$, since the packet won't be scheduled again. The probability to receive a packet in a round is $p^2$, as established in section 5.2.1.
The average bandwidth overhead factor is equal to the expected number of slots per packet: If we expect $j$ slots per packet instead 1 as in the ideal case, the bandwidth overhead factor is exactly $j$:

$$\text{E[slots per packet]} = \sum_{i=1}^{k-1} i\ (1-p^2)^{i-1}\ p^2 + k\ (1-p^2)^{k-1} \tag{5.1}$$

$$= \frac{1-(1-p^2)^k}{p^2} \tag{5.2}$$

How to get from 5.1 to 5.2 is explained in Appendix A, where we also show that for $k \to \infty$, the average bandwidth overhead factor approaches $1/p^2$ (fig. 5.2b).

**Summary & Discussion** We describe a fault-tolerant mechanism based on re-sending that closely relates to the repetition mechanism described in section 5.2.1. In particular, using the sink as LWB host allows requesting the resending of packets with the schedule instead of requiring additional packets. In contrast to the repetition mechanism, a packet is only re-sent until it is received once. This results in a expected bandwidth overhead much lower than the worst-case bandwidth overhead ($k$ times the ideal number of slots). The expected bandwidth overhead does not scale linearly with $k$, but asymptotically approaches $1/p^2$ times the ideal number of packets.

In other applications, it might be interesting consider a bandwidth overhead factor, not per packet, but per round, e.g. scheduling at most one slot for the re-sending of packets per round. Analysis of such a mechanism would depend on the traffic patterns, i.e. the size of rounds over time, which is out of scope for this work. Nevertheless, the approaches to modeling and analysis presented in this section are tools that can be used for such mechanisms as well.

## 5.3 Multiple Sinks

In this section we add additional sinks, e.g. multiple nodes with a GRPS Module in PermaSense. We denote the number of additional sinks by $s$. As described in

section 5.1, a packet needs to be received by at least one sink, no matter which one. One sink still is the LWB host, which we call the *host sink*. All other sinks are called *additonal sinks*.

First we show how additional sinks increase packet reception rate simply by being part of the network, i.e. without any dedicated communication with the host sink, because communication is based on flooding (see section 4.1).[1]

Furthermore, we motivate and analyze a fault-tolerance mechanism for multiple sinks based on sending acknowledgments from the additional sinks to the host sink.

### 5.3.1 Passively Increased Reception by Additional Sinks

All nodes in the network receive (and send) all packets during every Glossy flood, which means that every additional sink overhears all packets just by being part of the network, without any explicit communication with sources or the host sink.

**Packet Reception Rate** The packet reception rate is the probability that a packet is received by at least one sink (host or additional). The probability that the host sink receives a packet is still $p^2$. The probability that an additional sink receives a packet is $p^3$, since this depends on three packet receptions, which we assume to be independent (section 2.3):

1. The source must receive the schedule (to send the packet)
2. The additional sink must receive the schedule (to participate in the round)
3. The additional sink must receive the packet

Therefore, the packet reception rate (PRR) is:

$$\mathrm{PRR} = 1 - \underbrace{\left(1-p^2\right)}_{\text{host}} \underbrace{\left(1-p^3\right)^s}_{\text{additional}}$$

Which is monotonically increasing in $s$, i.e. with every additional sink, the packet reception rate increases.

Combined with the the mechanisms from sections 5.2.1 and 5.2.2, the packet reception rate increases further, without changes to the average and worst-case

[1] We assume the probability to receive a packet by a node to be equal at every point in space and time. In reality, interference might be localized. Multiple sinks in different locations are less affected by localized interference, therefore providing additional benefits not discussed in this work.

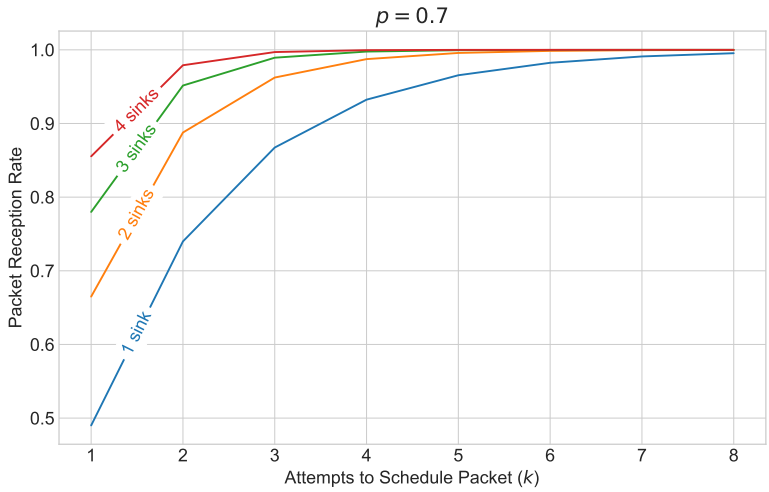



Figure 5.3: Package reception rateover attempts to schedule a packet ($k$) plotted for several sinks with a fixed probability to receive a packet in a round ($p = 0.7$). A desired packet reception rate can be achieved by both increasing $k$ or adding additional sinks.

bandwidth overhead factor. The packet reception rate (PRR) for a re-sending mechanism with additional sinks is:

$$\text{PRR} = 1 - \left(1 - p^2\right)^k \left(1 - p^3\right)^{sk}$$

In conclusion, the packet reception rate is increased by additional sinks and this effect scales with scheduling a packet multiple times.

**Bandwidth Overhead** The bandwidth overhead factor remains unchanged, since no additional slots are required[2] and the additional sinks do not communicate with the host sink, therefore, re-scheduling still depends on the reception by only the host sink.

**Summary & Discussion** Through Glossy flooding, additional sinks in the network increase reliability because they receive all packets. Furthermore, additional sinks do not increase the bandwidth overhead factor and work well together with the re-sending of packets, increasing benefits further.

The benefit of additional sinks is achieved without dedicated communication between the additional sinks and the host sink. While this is elegant in its

[2]Additional nodes might increase the network diameter, increasing the radio on-time of Glossy [1], which we don't look into further, since we try to find results independent of network topology.

simplicity, a packet might be re-scheduled by the host sink even though an additional sink might have received it. In the next section, we analyze one approach to address this.

Nevertheless, we are able to provide a model combining packet re-sending and additional sinks, which can be used to make a decision in the process of designing a network. If the cost of additional bandwidth and additional sinks or constraints on either bandwidth or number of sinks are given, this model can be used to find an optimal set-up.

### 5.3.2 Additional Sinks Actively Acknowledging Reception

Using a re-sending mechanism without any feedback from the additional sinks can cause packets to be re-scheduled unnecessarily: Regardless of the additional sinks, if the host sink has not received the packet, it is re-scheduled. In order to avoid this, communication between the additional sinks and the host sink is required.

Communication between sinks could happen out-of-band, e.g. over the Internet, but this is out of scope for this work. Instead we focus on a solution using LWB and extend the re-sending mechanism of section 5.2.2 to include acknowledgment packets, *ACK* in the following, sent by the additional sinks to inform the host sink about received packets.

We discuss how Glossy flooding helps limit the number of ACK slots required per packet and analyze the resulting mechanism.

**Acknowledgment Mechanism** We extend the re-sending mechanism of section 5.2.2 as follows:

- For each packet, additional slots for ACKs are scheduled.
- An ACK is a predefined packet, e.g. of one byte length with content 1.
- Additional nodes send an ACK, if they have received the packet, otherwise they don't send anything.
- If the host node has not received the packet or an ACK from at least one of the additional sinks, the packet is re-scheduled (up to $k$ times).

Naively, we need to schedule one ACK slot per packet for each additional sink, resulting in $s$ slots for ACKs per packet. Fortunately, since the network is based on Glossy, we are able to reduce this overhead.

As explained in section 2.1, a flood can be initiated from several nodes without changing the outcome, i.e. a packet is still received with probability $p$. Since

the objective is to receive the packet by at least one sink, no matter which one, all additional sinks which have received a packet can share an ACK slot by initiating the flood together. As a result, one ACK slot is required per packet, regardless of the number of additional sinks.

**Packet Reception Rate** ACKs do not influence the packet reception rate, they only allow re-scheduling to be stopped if one of the additional sinks has received a packet Therefore, the packet reception rate PRR is equal to the re-sending mechanism using additional sinks without ACKs.

$$\text{PRR} = 1 - (1 - p^2)^k (1 - p^3)^{sk}$$

**Bandwidth Overhead Factor** Contrary to the packet reception rate, to compute the bandwidth overhead factor, ACKs have to be considered. The host sink stops to schedule a packet either if it has received the packet itself or if it has received an ACK. If the host receives the packet, the additional sinks are irrelevant. Otherwise, if at least one of the additional sinks initiates ACK transmission, the ACK is received with probability $p$. If the host sink receives the packet or ACK, it is certain that the packet has been received and does not need to be re-scheduled (fig. 5.4a).[3] We call the probability, that the host sink is certain that the packet is received by itself or one of the other sinks, $p_{certain}$ and can use this to compute the average bandwidth overhead factor as in section 5.2.2. We have to consider that every time a packet gets scheduled, we actually require 2 slots, one for the packet and one for the shared ACK.

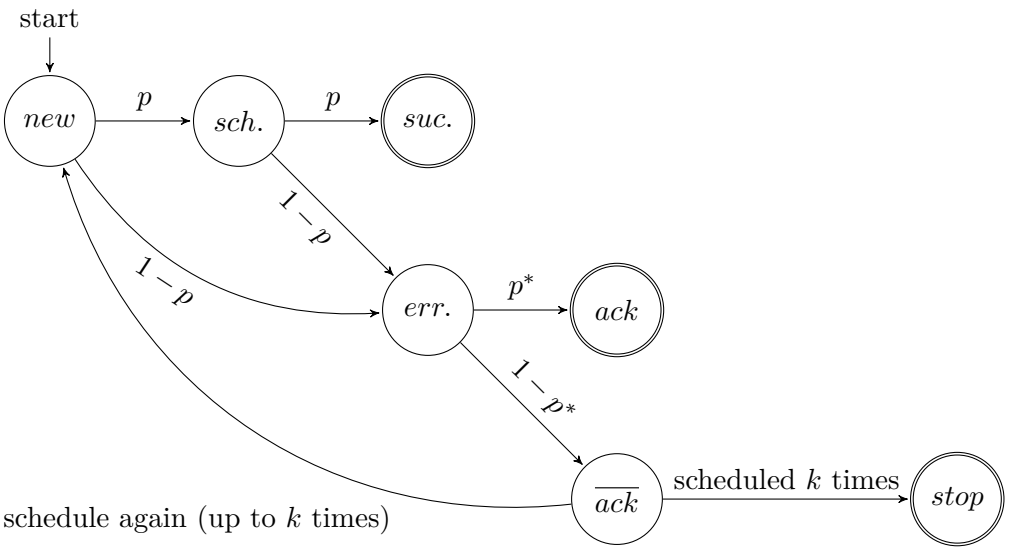
$$p_{certain} = \underbrace{p^2}_{host} + \underbrace{(1 - p^2)(1 - (1 - p^3)^s)\ p}_{additional} \tag{5.3}$$

$$\text{E[slots per packet]} = 2 \sum_{i=1}^{k-1} i\ (1 - p_{certain})^{i-1}\ p_{certain} + k\ (1 - p_{certain})^{k-1} \tag{5.4}$$

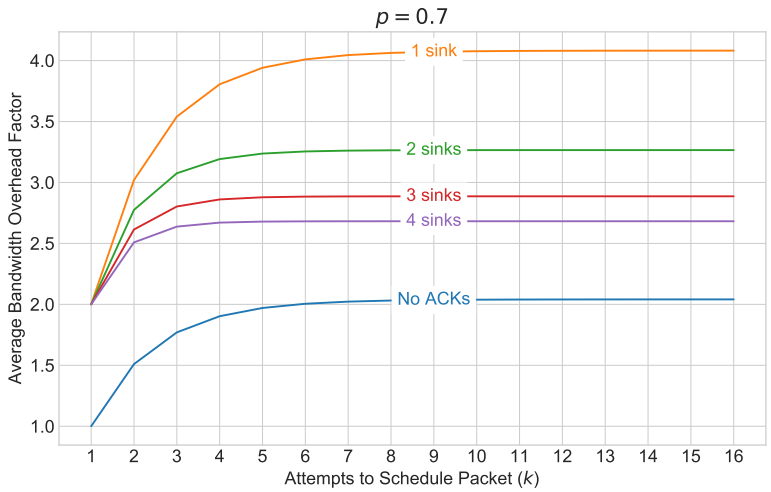
$$= 2\ \frac{1 - (1 - p_{certain})^k}{p_{certain}} \tag{5.5}$$

While the ACKs indeed allow the host sink to stop re-scheduling earlier, the additional slot required for the ACK causes this mechanism to actually have a higher bandwidth overhead factor compared to the mechanism without ACKs (fig. 5.4b). Nevertheless, without a shared ACK slot, the bandwidth overhead factor have a constant multiplier of $s$ instead of 2, i.e. the average bandwidth overhead would increase with the number of additional sinks!

[3] Not receiving an ACK does not mean that no additional node has received the packet, since the ACK could have been simply lost.




(a) FSM based on fig. 5.2a describing whether a packet is re-scheduled or not using a re-sending mechanism with additional sinks sending ACKs. The additional states are:
*ack*: The host sink has received an ACK.
$\overline{ack}$: The host sink has not received an ACK.
*stop*: The host sinks stops to re-schedule a packet.
The probability $p^*$ to receive an ack is the probability that at least one additional sink receives the packet times the probability that the ACK is received by the sink ($p$ as for all packets), i.e. $p^* = \left(1 - \left(1 - p^3\right)^s\right)\ p$.




(b) Average bandwidth overhead factor over attempts to schedule a packet ($k$), plotted for several numbers of sinks. The probability to receive a packet in a round is fixed to $p = 0.7$. Additionally the average bandwidth overhead factor without using ACKs as comparison. While, using ACKs, additional sinks lower the average bandwidth overhead factor, the slot required for ACKS causes the average bandwidth overhead factor to be higher compared to not using ACKs at all.

Figure 5.4: Additional sinks sending ACKs

**Summary & Discussion** We show how ACKs can be used for communication between additional sinks and the host sink to avoid unnecessarily re-scheduling a packet if it has already been received by an additional sink. Furthermore, by sharing the ACK slot, which is possible because of Glossy flooding, the number of ACK slots is limited to one per packet. Nevertheless, evaluation shows that ultimately the bandwidth overhead factor is higher – although the slots per packet can be reduced, the additional ACK slot counteracts this gain.

Furthermore it has to be considered that the simple ACK mechanism we analyze can not avoid unnecessary re-sending completely. Even if an additional sinks receive a packet, if the ACK is lost, the packet is re-scheduled.

It might be possible to address the overhead of ACKs by scheduling one ACK per additional sink per round containing information about all packets of the round. But analysis of this approach would depend on traffic patterns, i.e. how many slot are scheduled per round, which is out of scope of this work. Additionally, as mentioned above, the sinks could also use a common back-end like the Internet to exchange information, which might provide greater flexibility, although entwining LWB with another protocol like TCP can provide a challenge for modeling and analysis.

In consideration of these limitations, shared slots might be an interesting approach which is made possible by the use of Glossy floods – and indeed, in section 6.4, we briefly discuss another possible use case of shared slots for shared negative-acknowledgments.

## 5.4 Discussion

In this Chapter we describe low-power data collection applications like *PermaSense* and formulate the objectives of fault-tolerance mechanisms for such applications.

Based on this, we describe a mechanism based on re-sending packets and show that this mechanism essentially is an extension of a mechanism using packet repetition and differs by using the available information and the scheduling of LWB to avoid unnecessary repetition of packets without additional communication required.

The re-sending mechanism is limited by available bandwidth in the network and storage capacity of sources, but otherwise requires little computational overhead. Furthermore we show that the expected bandwidth overhead upper bounded by $1/p^2$ ($p$ is the probability to successfully receive a packet) and the packet reception rate is $1-(1-p^2)^k$ ($k$ is the maximum number of attempts to send a packet).

Next, we show how using additional sinks works well in Glossy-based net-

works, since all nodes in the network (including the additional sinks) receive all packets. This increases the packet reception rate without increasing bandwidth overhead and without any dedicated communication between the additional sink and the host sinks. As a result, the packet reception rate can both be increased by adding additional sinks or allowing each packet to be re-scheduled more often and we provide models for the influence of each. Given information such as cost to increase bandwidth versus cost of additional nodes, this can be used to find the best solution for a given situation.

Finally we describe a mechanism using both re-sending and additional nodes in combination with acknowledgment packets to allow the additional nodes to provide feedback. With this feedback, the host node can stop re-scheduling a packet if an additional nodes receives it. Ultimately, we find that the overhead of ACKs outweighs the benefits of avoiding unnecessary re-scheduling. Nevertheless, we show how the *sharing* of slots by jointly initiating a flood by several nodes allows to limit bandwidth overhead. While it is not useful in this particular mechanism, it might be an interesting approach in other situations. Another mechanism making use of this is briefly described in section 6.4.

All results must however be taken with the assumptions made in section 2.3 in mind. In particular, since nodes are sending multiple packets per round, the discussion in section 2.3.4 is relevant. In short, we underestimate our probability of success by assuming independence. Further work analyzing the impact of this assumption might be an interesting next step.

Chapter 6

# Fault-Tolerant Data Dissemination

In chapter 5, we introduce a data-collection scenario in low-power wireless networks. Such a network features multiple nodes, e.g. sensors, which collect and send data to a node with additional capabilities, such as an Internet connection. This sink also assumes the role of LWB host.

Often, the data flow is not one-directional. The host might not only collect data but distribute data packets itself, e.g. sensor settings, control commands or software updates, called *host packets* in the following. We assume host packets can be identified by sequence numbers without getting into more detail for the sake of simplicity. In this chapter, we take a look at data dissemination. As before, the objective is to achieve a high packet reception rate, i.e. as many of the distributed packets as possible should arrive, while keeping the bandwidth overhead factor, i.e. the required additional energy, low. Furthermore, it is interesting to consider the information available to the host, in particular, if the host knows which nodes in the network have received data, and which have not, e.g. to be aware that some sensors might operate with old settings and react accordingly.

W Unfortunately, analyzing data dissemination is more complicated than data collection. All nodes react to the sending of packets by the host, which in turn, depending on the mechanism in use, might schedule the dissemination of host packets based on all nodes in the network, e.g. based on the number of nodes which have received a previous host packet. This can cause nodes to depend on all other nodes in the network, which is difficult to model.

Nevertheless, we roughly describe three mechanisms for fault-tolerant data dissemination in LWB and comment on their benefits and challenges to provide a starting point for future work:

- ACKs for each node in dedicated slots
- ACKs added to existing data streams (*piggy-backing*)

- Shared negative-acknowledgments (NACKs)

## 6.1 Application and Objective

## 6.2 Dedicated Acknowledgments

A first and straightforward mechanism is to schedule ACKs for every node and host packet, similar to section 5.3.2. Unfortunately, we cannot share the acknowledgment slot. The only information provided by receiving a shared ACK is whether at least one node initiated the flood, i.e. at least one node acknowledges reception, which does not help since all nodes need to acknowledge reception and differentiating which node has acknowledged is important.

#### Example Mechanism

- For each host packet, the host schedules one ACK slot per node in the same round.
- If at least one ACK is not received, the packet is re-scheduled.
- Every time a packet is re-scheduled, ACK slots are scheduled again for nodes of which no ACK is received yet.
- Every packet is scheduled up to $k$ times.

This mechanism attempts to save at least some bandwidth by not scheduling ACKs for nodes which have already acknowledged reception of a packet.

**Benefits** Using ACKs by each node, the host receives precise information about which nodes have received a packet. Furthermore, the host receives feedback in the same round in which the packet is disseminated.

**Challenges** A considerable bandwidth overhead is required: If the network consists of $B$ nodes, at least $B$ additional slots are required for ACKs, without considering re-sending and further ACKs.

The issue of many slots could be addressed by scheduling one ACK slot per node, containing the data of the next expected packet, as in TCP. On the downside, several host packets can depend on the same ACK, increasing modeling difficulty.

The challenge of modeling could be approached by analyzing the number of nodes for which no ACK is received for a given round, i.e. the number of

unresponsive nodes. If there are unresponsive nodes, the packet must be rescheduled with as many ACK slots as unresponsive nodes. Depending on the number of unresponsive nodes in the previous round, the number of unresponsive nodes for the next round could be computed. The distribution of the number of unresponsive nodes could provide an estimate for the bandwidth overhead factor, since it determines the number slots needed in the next round, while the expected number of unresponsive nodes after $k$ rounds might be used to determine packet reception rate.

## 6.3 Piggy-Backing Acknowledgments onto Data Streams

**Example Mechanism** Opposed to scheduling dedicated slots to provide acknowledgments, the ACK could be attached to existing data streams, therefore increasing the size of packets sent, but not requiring additional slots.

**Benefits** As before, using acknowledgments provides the host with precise information whether a node has received a packet. Furthermore, no additional packets need to be scheduled.

**Challenges** The ACK is tightly coupled to the existing data streams in this case. If we do not have data streams for some nodes, we need to compensate this with something like acknowledgments as described in section 6.2, which brings the challenges described there. Furthermore, the difficulty of modeling the data streams, which depends on the scheduler in use, has to be considered.

## 6.4 Using Negative-Acknowledgment to Indicate Packet Loss

We have discussed two different ways to send ACKs, either in dedicated packets or by piggy-backing on existing data streams. In this section, we consider how using NACKS enables enables slot sharing, similarly to the shared ACK in section 5.3.2. While slot sharing saves bandwidth, the shared NACK does not provide the information of which nodes received a packet.

**Example Mechanism**

- For every host packet, a single NACK slot is scheduled in the same round.
- Every node which does not receive the host packet sends a pre-determined packet in the NACK slot.

- If a NACK packet is received by the host node, the host packet is re-sent.

Leveraging Glossy to share a slot is possible with NACKs, since it does not matter how many nodes have not received a packet – if one node has not received it, the packet needs to be re-sent.

**Benefits** By using shared NACKs, the number of additional slots required is limited, preserving bandwidth. Furthermore feedback is provided in the same round in which the host packet is scheduled.

**Challenges** Contrary to the previous mechanism, the NACK-based mechanism does not provide the information which nodes have received a packet: Not receiving anything during a NACK slot does not equal reception of the host packet by all nodes – the NACK packet could have been lost.

Nevertheless, a mechanism based on negative-acknowledgment can increase the probability to receive a packet with limited bandwidth overhead, if it is not important to know which node has received each packet.

## 6.5 Discussion

In this chapter, we introduced three mechanisms for data-dissemination scenarios: Sending ACKs in dedicated slots, piggy-backing ACKs onto existing data streams and shared NACKs.

In conclusion, the first two – acknowledgment-based – mechanisms we discuss suffer from high bandwidth overhead. Based on the work we did, it is not obvious if Glossy and LWB can be leveraged for acknowledgments in data dissemination applications.

Following a different approach, we present a mechanism based on negative-acknowledgments, which sacrifices information, i.e. which node has received what, but benefits from network flooding to conserve bandwidth.

All presented mechanisms are not analyzed as extensively as the mechanisms in chapter 5 because their dependencies provided modeling challenges out of scope for this work. Nevertheless, the mechanisms might provide a starting point and inspiration for future work.

Chapter 7

# Related Work

## 7.1 Protocols

**eLWB** In our data-collection application, we focus on periodically generated data. The Event-based Low-power Wireless Bus (eLWB) [8] targets data-collection in applications with additional random and sporadic events. It features a mechanism to register events. Although it is not the focus of eLWB, this mechanism could be an interesting approach to request retransmission of lost packets.

**Splash** Splash [9] is a Glossy-based protocol for data dissemination. It uses both FEC and ARQ for fault-tolerance. When distributing data, Splash adds 500 packets which are XOR-combinations of 20 randomly chosen data packets to allow nodes in the network to recover from single packet losses. After the data distribution is over, nodes which are still missing packets request them from their neighboring nodes.

**Pando** Similar to Splash, Pando [10] is a data dissemination protocol based on Glossy. It uses fountain coding to send a continuous stream of data to nodes in the network and implements an ARQ mechanism with *silence-based feedback* for fault-tolerance. The feedback works as follows: Pando organizes the network in a tree and parent nodes keep forwarding data until their child nodes stop transmitting. As soon as a leaf node has received all the data, it stops transmitting. This is detected by the parent nodes and their parents, ..., until the whole network is silent and the dissemination is complete. Therefore nodes can request missing packets by not going silent.

**Chaos** Chaos [11] takes a different approach to network flooding: In Chaos, every node adds its own data to the packet which is currently flooded, instead of repeating the packet of the initiating node. To realize this, Chaos is based on *merge operators*, which define how a node can combine its own data with the

packet it has received, e.g. aggregation functions like max, min or *sum*. This way, Chaos combines data collection and processing. Chaos implements an ARQ-like mechanism to terminate the flood: A status flag for each node in the network is added to every packet. If a node has participated in the flood, it sets its status to 1. This way node receive feedback from other nodes and can terminate the flood if every node has participated or restart the flood if some nodes are still missing.

## 7.2 Related Work on Modeling and Fault-Tolerance

**On Modeling Low-Power Wireless Protocols Based on Synchronous Packet Transmissions** The authors of [4] model the energy consumption of a re-sending mechanism as described in chapter 5 – indeed, our mechanism is based on their analysis. But their focus lies not on the fault-tolerance mechanism, but on energy consumption based on the reception of schedule packets, in particular modeling the response to lost schedules with a finite-state machine. They create an accurate model of LWB and are able to predict the energy consumption of LWB with an error of less than 1%.

**End-to-End Real-time Guarantees in Wireless Cyber-physical Systems** While we focus on packet reception rate and bandwidth overhead factor, for many applications timing is an important performance criterion. In particular, guaranteed end-to-end deadlines. One approach to guarantee such deadlines is to establish *real-time contracts* between applications and the network [12]. The authors use a protocol built on LWB and add admission tests to the registering of data streams. A stream is only registered if all parts of the chain from application to application have the capability to deliver the packets of the data stream. This way, guarantees can be given for all admitted streams.

**A Survey on Data Dissemination in Wireless Sensor Networks** In a survey about fault-tolerant data dissemination, many approaches to fault-tolerance in wireless networks based on routing are analyzed [13]. The report confirms that up-to-date, little work (and analysis) has been done on fault-tolerance in Glossy-based networks, but states that Glossy-based communication is a promising technique for data dissemination.

Chapter 8

# Conclusion

We provide a classification of problems affecting LWB-based networks and model and analyze several fault-tolerance mechanisms for data collection and briefly discuss possible approaches to fault-tolerant data dissemination. Preliminarily, we summarize the core concepts of Glossy and LWB in chapter 2, describe statistical properties of packet reception in Glossy and motivate and discuss further assumptions we make. In chapter 3, we classify faults, errors and failures in an LWB-based network and model their relationship with fault-trees. We briefly argue why we only consider interference as a fault for our analysis. Based on our assumptions, we are able to model the 'lifetime' of each packet independently with a finite-state machine. In chapter 4 we introduce Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) as the two basic approaches to fault-tolerance. Additionally, we describe packet reception rate and bandwidth overhead factor(related to energy consumption), the two main criteria we use for evaluation of fault-tolerance mechanisms. In chapter 5 we introduce a data collection application based on wireless sensor networks like PermaSense and present several fault-tolerance mechanisms:

- We first focus on a single data collecting sink, present an FEC mechanism on packet repetition and an ARQ mechanism based on re-sending of lost packets, which achieves the same packet reception rate at much lower average bandwidth overhead factor compared to the simple repetition mechanism.
- We also analyze several (redundant) sinks and analyze how they can benefit from Glossy without communication between each other. Furthermore, we present a simple acknowledgment-based mechanism for communication between sinks and explain how Glossy can be leveraged to save bandwidth required for acknowledgments.

Finally in chapter 6 we briefly discuss several approaches to data dissemination.

In conclusion, the statistical properties of Glossy allow us to create a simple model for packet reception which allows us to analyze the proposed fault-

tolerance mechanisms for data collection. We are able to present several ways to leverage Glossy and LWB for the design of fault-tolerance mechanisms:

- LWB scheduling is used to request re-sending of lost packets without any acknowledgment packets.
- The flooding-based communication of Glossy allows every node in the network to overhear every packet, allowing additional sinks in the network to collect data (and increase reliability) without any changes to the way the network operates.
- By initiating a Glossy flood with the same packet by multiple nodes, the slot for an acknowledgment packet is shared to preserve bandwidth.

On the other hand, based on our discussion on data dissemination it is not obvious whether Glossy and LWB-based communication can be leveraged for data dissemination as well.

While some statistical properties of Glossy have been validated, we rely on some additional assumptions for our analysis. Further analysis could be interesting work to gain insight in which situations our assumptions are valid or to be able to find models which do not require on these assumptions. Aside from this, our results aim to be as general as possible, independent of different traffic patterns and network topologies. Our work can be taken as a foundation for analysis in more specific use-cases, e.g. specific network topologies and known traffic patterns. Finally, the mechanisms we presented are only a subset of all possibilities. While we are able to present fault-tolerance mechanisms leveraging Glossy and LWB, there might be other equally – or more – interesting mechanisms in Glossy-based wireless communication networks worth investigating.

Appendix A

# Closed Forms of Sums

For any $0 < q < 1$:

$$\sum_{i=1}^{k-1} i\ (1-q)^{i-1}\ q + k\ (1-q)^{k-1} \tag{A.1}$$

$$= \sum_{i=1}^{k-1} i\ (1-q)^{i-1}\ q + k\ (1-q)^{k-1}(q + (1-q)) \tag{A.2}$$

$$= \left(\sum_{i=1}^{k-1} i\ (1-q)^{i-1}\ q + k\ (1-q)^{k-1} q\right) + k\ (1-q)^{k} \tag{A.3}$$

$$= \sum_{i=1}^{k} i\ (1-q)^{i-1}\ q + k\ (1-q)^{k} \tag{A.4}$$

$$= \frac{q}{1-q}\left(\sum_{i=1}^{k} i\ (1-q)^{i}\right) + k\ (1-q)^{k} \tag{A.5}$$

$$= \frac{q}{1-q}\left(\frac{k(1-q)^{k+2} - (k+1)(1-q)^{k+1} + (1-q)}{(1-(1-q))^2}\right) + k\ (1-q)^{k} \tag{A.6}$$

$$= \frac{k(1-q)^{k+1} - (k+1)(1-q)^{k} + 1}{q} + k\ (1-q)^{k} \tag{A.7}$$

$$= \frac{k(1-q)^{k}(1-q) - k(1-q)^{k} - (1-q)^{k} + 1 + k(1-q)^{k} q)}{q} \tag{A.8}$$

$$= \frac{1 - (1-q)^{k} + k(1-q)^{k}\left[(1-q) - 1 + q\right]}{q} \tag{A.9}$$

$$= \frac{1-(1-q)^{k}}{q} \tag{A.10}$$

$$\tag{A.11}$$

Where we use the elementwise differentiation to go from step A.5 to A.6:

$$\sum_{i=1}^{N} kw^k = \sum_{i=1}^{N} w\frac{\partial}{\partial w}w^k \tag{A.12}$$

$$= w\frac{\partial}{\partial w} \underbrace{\sum_{i=1}^{N} w^k}_{\text{geometric series}} \tag{A.13}$$

$$= w\frac{\partial}{\partial w}\frac{1-w^{n+1}}{1-w} \tag{A.14}$$

$$= \frac{kw^{k+2} - (k+1)w^{k+1} + w}{(1-w)^2} \tag{A.15}$$

For $k \to \infty$, we do not have a final term, but otherwise the idea is the same. Since $1 - q$ is less then 1, the infinite sum converges:

$$\sum_{i=1}^{\infty} \; i\;(1-q)^{i-1}\; q = \frac{q}{1-q}\left(\sum_{i=1}^{\infty} \; i\;(1-q)^{i}\right) \tag{A.16}$$

$$= \frac{q}{1-q}\cdot\frac{1-q}{(1-(1-q))^2} \tag{A.17}$$

$$= \frac{1}{q} \tag{A.18}$$

We use the same trick with elementwise differentiation to get a closed representation for the sum in A.16, but have an infinite geometric series now:

$$\sum_{i=1}^{\infty} kw^k = w\frac{\partial}{\partial w}\sum_{i=1}^{\infty} w^k \tag{A.19}$$

$$= w\frac{\partial}{\partial w}\frac{1}{1-w} \tag{A.20}$$

$$= \frac{w}{(1-w)^2} \tag{A.21}$$

# Bibliography


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

[2] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *Journal of Network and Systems Management*, vol. 15, no. 2, pp. 171–190, 2007.

[3] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pp. 1–14, ACM, 2012.

[4] M. Zimmerling, F. Ferrari, L. Mottola, and L. Thiele, "On modeling low-power wireless protocols based on synchronous packet transmissions," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, pp. 546–555, IEEE, 2013.

[5] L. M. S. De Souza, H. Vogt, and M. Beigl, "A survey on fault tolerance in wireless sensor networks," *Interner Bericht. Fakultät für Informatik, Universität Karlsruhe*, 2007.

[6] Wikipedia, "Error detection and correction — wikipedia, the free encyclopedia," 2017. [Online; accessed 18-June-2017].

[7] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin, "Permasense: Investigating permafrost with a wsn in the swiss alps," in *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, (New York, NY, USA), pp. 8–12, ACM, 2007.

[8] F. Sutton, D. Forno, G. Reto, T. David, Gsell, R. Lim, J. Beutel, and L. Thiele, "The design of a responsive and energy-efficient event-triggered wireless sensing system,"

[9] M. Doddavenkatappa, M. C. Chan, B. Leong, *et al.*, "Splash: Fast data dissemination with constructive interference in wireless sensor networks.," in *NSDI*, pp. 269–282, 2013.

[10] W. Du, J. C. Liando, H. Zhang, and M. Li, "Pando: Fountain-enabled fast data dissemination with constructive interference," *IEEE/ACM Trans. Netw.*, vol. 25, pp. 820–833, Apr. 2017.

[11] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, p. 1, ACM, 2013.

[12] R. Jacob, M. Zimmerling, P. Huang, J. Beutel, and L. Thiele, "End-to-end real-time guarantees in wireless cyber-physical systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, pp. 167–178, Nov 2016.

[13] X.-L. Zheng and M. Wan, "A survey on data dissemination in wireless sensor networks," *Journal of Computer Science and Technology*, vol. 29, no. 3, pp. 470–486, 2014.