Diss. ETH No. 21474

Mastering Imperfect and Partial Information in Wireless Sensing Systems

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
MATTHIAS KELLER
M.Sc. TUM
born April 22, 1983
citizen of Germany

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Kay Römer, co-examiner
Dr. Jan Beutel, co-examiner

2013

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. 139

Matthias Keller

Mastering Imperfect and Partial Information in Wireless Sensing Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Diss. ETH No. 21474

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Kay Römer, co-examiner
Dr. Jan Beutel, co-examiner

Examination date: September 25, 2013

ISBN 978-3-906031-39-2
DOI 10.3929/ethz-a-009958164

Abstract

Wireless sensor networks (WSNs) are networks of self-organizing, autonomously operating, resource-constrained computing devices. Typical designs of such so called mote-class devices integrate a low-power microcontroller (MCU), a radio chip, external memory and a radio frequency (RF) front-end on an area of a few square centimeters in size. Multi-year operation from small batteries is achieved by utilizing energy-saving low-power states and switching off currently not used components, *e.g.*, the radio chip, whenever possible.

The most prominent use of such networks are so called sense-and-send applications. In a sense-and-send application, nodes are regularly sampling data that is then transmitted to a sink node. To date, the vast majority of published sensor network deployments employs a multi-hop routing protocol for communication. Such protocols create and maintain a dynamic routing tree, packets are forwarded over multiple hops until they are eventually received by a sink.

Wireless communication is not perfect, but subject to well-known phenomena such as multi-path propagation, interference, and loss. Additionally, the resource-scarcity found in systems that are built from mote-class devices limits the amount of capabilities that can be added to a system. For example, in contrast to Internet networks, implementing services such as active network monitoring and network time synchronization can add a considerable, even harmful overhead to a low-power wireless system.

Overall, known properties of wireless communication, common properties of distributed systems, and resource constraints specific to low-power wireless networks altogether negatively impact the quality of data obtained from such a system. Similarly, mentioned characteristics also render debugging low-power wireless networks as a very demanding task. System state first of all being distributed among the network, deployed systems also lack the resources needed for making that state accessible.

In the context of a scientific, multi-year, multi-site permafrost and rock kinematics monitoring effort, this thesis presents algorithms and systems for establishing wireless data collection systems as dependable and precise scientific instruments. Theoretical results obtained are backed by strong empirical evidence, *i.e.*, evaluated in simulation, in testbed experiments on real hardware, and on up to 270 million packets that originate from deployed wireless sensor networks.

The contributions of this thesis are presented in four interconnected parts. First, this thesis contributes to the state-of-the-art by presenting algorithms for the mitigation of data imperfections that are introduced by phenomena common to wireless multi-hop communication. Second, this thesis presents a system for visualizing large sensor network data sets and thus making sensor data more accessible. The third contribution of this thesis tackles the problem of wireless sensing systems on the one hand maintaining vast amounts of distributed state while on the other hand being too resource-scarce for this state being transferred out of the network. Results of the presented network tomography algorithm are then used for building a minimally active system for monitoring network health, the fourth and final contribution of this thesis.

Zusammenfassung

Drahtlose Sensornetzwerke sind Kommunikationssysteme, die aus sich selbst organisierenden, autonom arbeitenden, an Ressourcen beschränkten Rechengeräten bestehen. Typische Sensorknoten bestehen aus einem stromsparenden Mikrokontroller, einem Funkmodul, und einem Speicherchip. Alle Komponenten sind auf einer Fläche von wenigen Quadratzentimetern untergebracht. Ein mehrjähriger Betrieb aus einer einzigen Batterieladung wird durch die Verwendung von energiesparenden Betriebsmodi und insbesondere durch das dynamische Ein- und Ausschalten von Teilkomponenten erreicht.

Die am meisten verbreitete Anwendung von drahtlosen Sensornetzwerken ist die regelmässige Durchführung von Messungen. Die Resultate dieser Messungen werden unmittelbar nach der Aufnahme drahtlos an eine zentrale Senke übertragen. Nahezu alle bekannten Installationen verwenden dabei ein sogenanntes Multi-Hop Routing Protokoll. Hierbei organisieren sich alle Sensorknoten selbstständig in einer Baumstruktur. Pakete von weiter entfernten Sensorknoten werden über mehrere Stationen weitergeleitet bis sie die Senke erreichen.

Bei drahtloser Kommunikation können allgemein bekannte Effekte wie Mehrpfadverbreitung, Interferenz und Paketverlust auftreten. Der Betrieb von funkbetriebenen Sensornetzwerken wird zusätzlich durch den Mangel an Systemressourcen erschwert. So können zum Beispiel im Internet weit verbreitete Dienste wie die aktive Überwachung von Netzwerken als auch Netzwerk-Zeitsynchronisation nicht ohne weiteres in drahtlosen Funknetzwerken implementiert werden. Der in drahtlosen Sensornetzwerken im Vergleich zu den verfügbaren Ressourcen sehr hohe Aufwand für die Implementierung derartiger Dienste kann bis zur Degradierung des gesamten Systems führen.

Die bekannten Eigenschaften von kabelloser Übertragung, bekannte Herausforderungen in verteilten Systemen sowie für funkbetriebene Sensornetzwerke spezifische Ressourcenbeschränkungen können die Qualität der erhaltenen Sensordaten negativ beeinflussen. Gleichzeitig erschweren die genannten Charakteristiken die Untersuchung von Sensornetzwerken. Während der Zustand des Gesamtsystems einerseits über das Netzwerk verteilt ist, so erschwert der Mangel an Systemressourcen ausserdem die Zusammenführung des verteilten Systemzustandes an einem zentralen Ort.

Die geleisteten Beiträge der vorliegenden Arbeit haben das Ziel drahtlose Sensornetzwerke als zuverlässige und präzise Messinstrumente für wissenschaftliche Kampagnen tauglich zu machen. Vorgestellte Algorithmen werden anhand ausführlicher Analysen von Messdaten, die aus Simulationen, aus Testläufen auf realer Hardware und von Produktivsystemen stammen, validiert und bestätigt.

Die Beiträge dieser Dissertation erstrecken sich über vier zusammenhängende Teile. Der erste Teil dieser Arbeit präsentiert Algorithmen für die Entfernung von Datenartefakten, die typisch für drahtlose Multi-Hop Kommunikation sind. Der zweite Teil befasst sich mit einem System zur Visualisierung von großen Datenmengen. Ein Algorithmus zur Durchführung einer Tomographie in einem energiesparenden Sensornetzwerk wird im dritten Teil dieser Arbeit präsentiert. Der vorgestellte Algorithmus verwendet nur bereits vorhandenen Netzwerkverkehr und belastet das ressourcenarme Sensornetzwerk daher nicht. Ein minimal aktives System zur Netzwerküberwachung auf Basis der im vorherigen Kapitel vorgestellten Netzwerktomographie wird im vierten Teil dieser Arbeit vorgestellt.

This thesis was supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322.

The work presented is scientifically evaluated by the SNSF, financed by the Swiss Confederation, and funded by Nano-Tera.ch.

Contents

Abstract i

Zusammenfassung iii

1 Introduction 1
- 1.1 Wireless Sensor Networks 2
- 1.2 Challenges in Wireless Data Collection 3
- 1.3 Exemplary Deployment Project: PermaSense 6
- 1.4 Thesis Outline and Contributions 8

2 Model-based Data Cleaning and Order Reconstruction 11
- 2.1 Introduction 12
- 2.2 Related Work 15
- 2.3 Packet Classification and Order Reconstruction 16
- 2.4 System Model 16
- 2.5 Data Analysis 21
- 2.6 Case Study 29
- 2.7 Broader Applicability and Limitations 37
- 2.8 Conclusions 38

3 Visualization of Large Data Sets in Space and Time with Vizzly 41
- 3.1 Introduction 41
- 3.2 Related Work 43
- 3.3 Visualizing Large Sensor Network Data Sets 44
- 3.4 Vizzly System Design 46
- 3.5 Vizzly Implementation 52
- 3.6 Two Diverse Use Cases 54
- 3.7 Performance Evaluation 56
- 3.8 Integrating and Extending Vizzly 58
- 3.9 Conclusions 60

4 Multi-Hop Network Tomography 61
- 4.1 Introduction 61
- 4.2 Related Work 64
- 4.3 Exploiting Information Implicitly Given 65
- 4.4 Multi-Hop Network Tomography 65
- 4.5 System Model 67
- 4.6 Safe Information Reconstruction 69

4.7 Multi-Protocol Testbed Evaluation 79
4.8 Making Real Network Dynamics Visible 85
4.9 Broader Applicability and Limitations 92
4.10 Conclusions 93

5 Hybrid Network Health Monitoring **95**
5.1 Introduction 95
5.2 Related Work 97
5.3 Inferring State From Minimal Information 99
5.4 Hybrid Monitoring 99
5.5 System Model 101
5.6 From a Single Bit to a Scalar Value 104
5.7 Multi-Testbed Evaluation 109
5.8 Broader Applicability and Limitations 115
5.9 Conclusions 117

6 Conclusions **119**

Bibliography **121**

Curriculum Vitæ **131**

List of Publications **133**

1

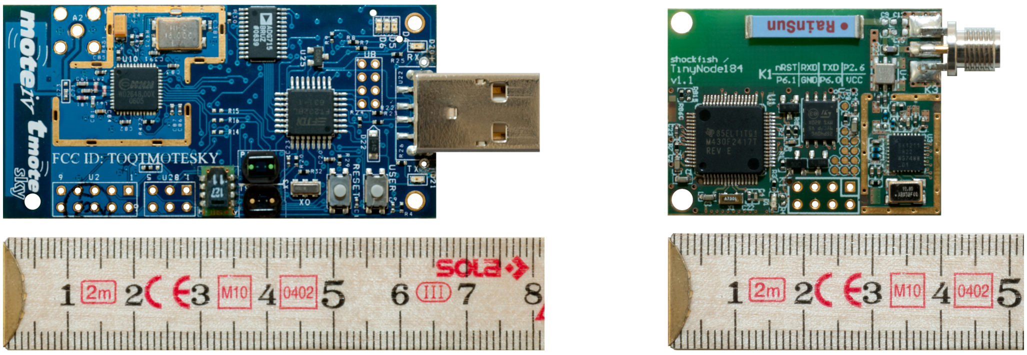
Introduction

Today's quality of living in industrialized countries is also the product of countless, often unnoticed, ubiquitous sensing systems that are permanently taking observations. For instance, being able to determine the arrival of a certain weather condition down to the hour would not be possible without a multitude of sensors on the ground, in the air, and in space. Natural disasters, *e.g.*, floods and landslides, would result in a higher number of fatalities if endangered populations could not be warned prior to an event. Continuous air pollution measurements protect the inhabitants of high traffic zones, *e.g.*, cause road traffic to be restricted when certain thresholds are exceeded.

In the context of a scientific, multi-year, multi-site permafrost and rock kinematics monitoring effort, this thesis presents algorithms and systems for establishing wireless data collection systems as dependable and precise scientific instruments. First, this thesis contributes to the state-of-the-art by presenting algorithms for the mitigation of data imperfections that are introduced by phenomena common to wireless multi-hop communication. Second, this thesis presents a system for visualizing large sensor network data sets and thus making sensor data more accessible. The third contribution of this thesis tackles the problem of wireless sensing systems on the one hand maintaining vast amounts of distributed state while on the other hand being too resource-scarce for this state being transferred out of the network. Results of this network tomography algorithm are used for building a minimally active system for monitoring network health, the fourth and final contribution of this thesis.

1.1 Wireless Sensor Networks

Wireless sensor networks (WSNs) are networks of self-organizing, autonomously operating, resource-constrained computing devices. Typical designs of such so called mote-class devices integrate a low-power micro-controller (MCU), a radio chip, external memory and a radio frequency (RF) front-end on an area of a few square centimeters in size (see Figure 1.1). Multi-year operation from small batteries is achieved by utilizing energy-saving low-power states and switching off currently not used components whenever possible. For example, state-of-the-art low-power communication protocols [BvRW07, FZMT12] reduce the on-time of the radio to less than 1% of the time.



(a) Tmote Sky **(b)** Tinynode 184

Fig. 1.1 Tmote Sky and Tinynode 184 sensor nodes. Both platforms are based on a TI MSP430 micro-controller. The Tmote Sky uses a CC2420 radio at 2.4 GHz, a SX1211 radio that operates at 868 MHz is used in the Tinynode.

The most prominent use of such networks are so called sense-and-send applications. In a sense-and-send application, nodes are regularly sampling data that is then transmitted to a sink node. The number of sink nodes is significantly smaller than the number of nodes that perform sensing tasks, in many situations a single sink node is sufficient. Directly connecting all sensor nodes to a sink node requires no routing layer, but also limits the network coverage to the maximum allowed distance between two devices. In contrast, the coverage of a network w.r.t. energy spent can be extended when a multi-hop routing protocol [BvRW07, GFJ+09] is used. For the price of a higher system complexity, this is achieved by packets being wirelessly forwarded over multiple hops until they are eventually received by a sink node.

Requiring no external wiring for communication and power supply renders networks of small, battery-powered, wirelessly communicating devices as viable for many situations in which practical, esthetic, economical, technical, or regulatory reasons prohibit the installation of extensive cabling. Exemplary applications of this kind are the installation of a wireless sensor network within the habitat of a bird [SMP+04], on a potato field [LBV06], in an high-altitude environment [BGH+09], in an old heritage building [CMP+09], and in a road tunnel [CCD+11].

Furthermore, interfacing with Internet networks requires only the sink nodes to be connected to an external network. Thus only the placement of the sink nodes may be constrained by external factors, *e.g.*, GSM coverage. The majority of nodes can be placed as required by the application whilst nevertheless being connected to the Internet. Permanent Internet connectivity enables quasi real-time access to sensor data, fast access to system health information, an the ability to remotely control the whole network.

1.2 Challenges in Wireless Data Collection

Applications envisioned in the area of environmental monitoring [BBF+11] want to employ wireless sensor networks as new scientific instruments. Ultimately, the goal is to build highly reliable, highly precise, wirelessly communicating systems that are suited for early warning scenarios. Taking responsibility for the inhabitants and infrastructure of regions that are threatened by natural hazards is particularly demanding when resource-constrained, low-power wireless systems are considered. First of all wireless communication itself being subject to a number of well-known effects, *e.g.*, multi-path propagation, interference and signal absorption, designing for a multi-year system lifetime limits the amount of capabilities that can be implemented on a resource-constrained device.

Two important metrics that are significantly influenced by the resource-scarcity found in low-power wireless sensor networks are the quality of data received and the observability of system state. In the following, we will now investigate the influence on each of the two metrics.

1.2.1 Imperfect Data

The first important metric that is influenced by decisions made to save resources is the quality of data received. Data received from wireless sensing systems is not always immediately useable, but may require post-processing. A prominent example for such post-processing is the reconstruction of time information [WALJ+06, GMEST09, LDCE09]. Further problems are the reconstruction of the logical packet ordering and the removal of invalid packets, *e.g.*, packets that carry bogus information and packet duplicates.

Independent of the condition of a physical sensor, *e.g.*, a temperature sensor, that is attached to a sensor node, the following properties of wireless multi-hop networks can negatively influence the quality of data received:

Imprecise, local clocks. Time is a very important information in wireless sensor networks. First of all needed for knowing when a sensor value was taken, time is also critical for scheduling joint wake-ups of duty-cycled sensor nodes. Thus, each sensor node operates a local clock that is continuously updated. While this requires the corresponding crystal oscillator to be always on, common node designs employ a low-frequency crystal for operating the local clock [SDS10]. As any real crystal, the crystal used for operating the clock is not perfectly stable, but running at varying frequencies that are slightly higher and lower than the desired speed. Factors that influence the actual speed of the crystal are the temperature, the stability of the operating voltage, and the condition of the crystal itself, among others. The impact of using a real, low-frequency crystal on the quality of time information obtained is two-fold. First, the resulting drift of the local clock can result in a constantly increasing deviation between the actual and the measured time. Second, using a low-frequency crystal does not only reduce the energy consumption but also the precision of the local clock.

Volatile, local state. The operation of a sensor node requires certain state to be stored in the memory of the micro-controller. Examples are the current local time, the current value of the local packet sequence counter, information on neighboring nodes, and packets that are currently waiting for transmission. Independent of the concrete implementation, *i.e.*, storage in volatile or non-volatile memory, certain events, *e.g.*, a sudden power loss, can cause parts of this local state being corrupted. The volatility of local state can cause packets to carry bogus information or even to be lost.

Lossy wireless channel. Several circumstances, *e.g.*, external interference, can cause packet transmissions to fail. A common strategy for avoiding packet loss is to retransmit a packet until its reception was successfully acknowledged by its immediate receiver. Packet acknowledgements also being subject to transmission failures, a packet is falsely retransmitted and thus duplicated if not the initial transmission itself but the transmission of the acknowledgment failed. If they remain undiscovered, packet duplications can affect a later data analysis.

Routing dynamics in multi-hop networks. Routing dynamics found in multi-hop networks can cause packets of the same source to travel along different paths. As the timing among different paths can vary, a later generated packet may arrive at the sink node before another packet that has been generated earlier. While the mere number of different routing paths generally complicates the tracking of packets inside the network, the possibility of packets being reordered also increases the amount of context that is needed for cleaning tasks, *e.g.*, the detection of packet duplicates.

From an algorithmic perspective, problems mentioned can be alleviated or even completely excluded by adding corresponding capabilities to the network. For example, the effects of clock drift can be mitigated by using an in-band clock synchronization protocol [MKSL04a, LSW09, FZTS11] or by adding receiver hardware for an external clock source [RGR09, CWCT11]. A similar example is the removal of packet duplicates inside the network. It requires only a simple filter that is running on every sensor node for successfully removing the majority of packet duplications inside the network [GFJ$^+$09]. The problem of losing state during node resets can be addressed by letting sensor nodes regularly share their own state with their neighbors [GMEST09].

However, it becomes also apparent that complexity and resource consumption increase when more capabilities are added to a system. For example, in-band clock synchronization requires extra communication which also translates to more energy. Additionally, adding capabilities often also narrows the overall flexibility, *e.g.*, satellite-based time synchronization can only be used in outdoor environments. Lastly, the effectiveness of an added capability is reduced when its requirements, *e.g.*, the need for permanent network connectivity in order to achieve its maximal accuracy, cannot always be satisfied.

In the end, it can only be decided in the context of a concrete sensing scenario if adding certain capabilities would contribute a benefit that is worth the effort. The question arises if and how data nevertheless can be used for demanding tasks despite carrying imperfections that are common to data that originates from low-power wireless sensing systems.

1.2.2 Only Partially Observable State

The second consequence of the resource-scarcity found in wireless low-power systems is the vast majority of system state remaining unobservable from outside the network. Available information is usually limited to a minimal amount, *e.g.*, battery voltage readings and minimal topology information. Though many interactions occur inside a network of autonomously acting sensor nodes until a packet eventually arrives at a sink, it is very difficult to judge the performance of a system from the little piece of the overall system state that reaches the surface. Without further performance data, it often remains unclear if algorithms and parameters used perform as expected in a concrete setting. Ideally, network operators would be able to access detailed information on the operational efficiency such as per-link packet reception rates and the distribution of the packet forwarding load.

However, the amount of state that can be transmitted over the single radio resource in deployed networks is limited by energy and reliability considerations. First of all requiring more energy, additional traffic can also be harmful to a system, *e.g.*, decrease the performance by increasing the number of collisions on the wireless channel. In consequence, carefulness often prefers the least amount of information to be actively transmitted.

In the light of recently proposed passive monitoring systems [LLL10], the question arises if certain state does not necessarily need to be actively transmitted, but can also be estimated or recovered from other, minimal, actively transmitted information.

1.3 Exemplary Deployment Project: PermaSense

The starting point of this thesis is the availability of the first year of data that has been received from the PermaSense Matterhorn deployment. Initially deployed in 2008, the purpose of this installation on the Matterhorn mountain is to collect geophysical measurements for the investigation of permafrost in the Alps [HTB+08]. Today, the still ongoing Matterhorn deployment and the subsequently installed deployment on the Jungfrau mountain are two of the longest lived sensor network deployments known in the research community.

DEPLOYMENT	TIME FRAME	#(SENSOR NODES)	#(SINKS)	MAX. HOPS
Matterhorn	07/2008–today	31	1	3
Jungfraujoch	02/2009–today	29	1	3
Thur	03/2010–05/2012	7	1	1
Dirruhorn	08/2010–today	47	3	5
Aiguille du Midi	07/2012–today	17	1	5

Tab. 1.1 PermaSense deployments in Switzerland and France

A complete deployment inventory is shown in Table 1.1. The currently instrumented locations are all located in alpine, high-altitude environments that are subject to extreme seasonal and daily temperature variations, wind, ice and snow. The technology core of all deployments are Tinynode 184 nodes that run the Dozer [BvRW07] ultra low-power data gathering protocol.

The majority of nodes is installed in conjunction with the so-called *Sensor Interface Board* [BGH+09], a custom-built extension board that integrates an analog-to-digital converter, a SD memory card, a battery, circuitry needed for converting between different serial data interface standards, a buzzer, and a reed switch for resetting a node from outside (see Figure 1.2). Other designs integrate Tinynode 184 nodes with a second computing device to build a base station, to use the sensor node as a voltage monitor and remote power switch [KYB09, BYL+11], or to integrate a complex sensing system into a Dozer network [GBG+12].

Dozer is a cross-layer communication stack that integrates a MAC layer and a multi-hop routing protocol into one holistic piece. An

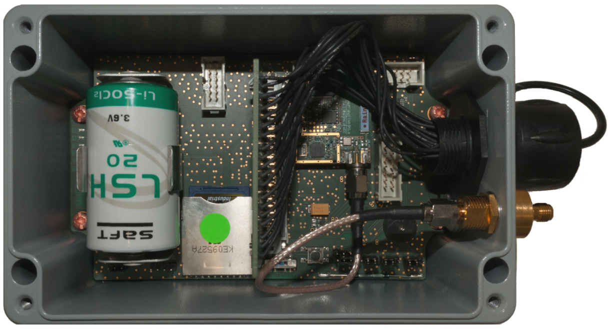


Fig. 1.2 Tinynode 184 sensor node on top of a second-generation Sensor Interface Board (SIB). The 1 GB large SD memory card can store approximately 27 million data packets and is thus well-suited both as a buffer for unsent packets and as a long-term backup medium.

important concept in Dozer is the periodic transmission of beacon messages. Beacon messages are generated at the sink and repeated by all currently connected nodes. As such, beacon messages fulfill the three tasks of announcing the network to yet unconnected nodes, to provide a facility for sending commands into the network, and to act as a synchronization primitive between already connected nodes. Concretely, beacon messages are used to plan the next joint wakeup of a node with each of its children. Coordinated, precisely scheduled wake-ups for data transmission are crucial for achieving the very low radio duty cycles as reported for Dozer [KWL+11].

Temperature-dependent clock variations are locally compensated based on the actual temperature inside the enclosure. Packet generation timestamps are obtained using elapsed time on arrival [KDL+06], the sink is the only component in the network that runs a synchronized clock.

Extreme weather conditions as found in high-altitude environments can cause sensor nodes to not being able to communicate for several days or even weeks. Unsent packets are buffered on the SD memory card until a sensor node is again able to communicate.

1.3.1 Joint Research and Development

Since its initial deployment in 2008, the PermaSense project has grown both in the quantity and in the types of sensors used. Today's system embraces four active deployment sites, the project currently operates more than 110 low-power sensor nodes that are continuously taking measurements. Initially limited to rather simple analog and digital sensors, *e.g.*, temperature and cleft dilatation sensors, low-power sensor nodes are now also in charge of controlling complex, self-contained sensing systems, *e.g.*, multi-sensor weather stations, GPS sensors, and high-resolution imagers.

Growing the system in both the number of devices and in terms of sensing modalities required all layers of its initial design to be either significantly improved or even replaced. Prior to the start of the actual research work presented in this thesis, significant efforts have been put into the ramp-up of the current system generation. The first of these contributions is the design and implementation of the PermaSense IP network infrastructure. While the initial prototype used a GSM modem for all communication between a field site and the university campus, data is nowadays transmitted over dedicated long-haul Wi-Fi communication links and VPN (virtual private network) tunnels on top of leased Internet lines. To date, the PermaSense IP network comprises around 120 hosts, *i.e.*, networking equipment, sensor network gateways, webcams, and experimentation hardware [BYL+11]. The latter is a refinement of the first prototype of MountainView, a high-resolution imager for outdoor application and the second major contribution to the PermaSense system. MountainView [KBT09, KYB09] was the first system within PermaSense in which a powerful embedded PC and a power-hungry sensor were controlled by a low-power sensor node. While commercial off-the-shelf (COTS) components used do not offer low-power sleep modes, the low-power sensor node is in charge of switching off those components whenever possible. In contrast to the switching of micro-controller components within milliseconds, COTS components remain in a steady power state for at least a few minutes and up to several hours. The concept of duty-cycling power-hungry COTS devices has been transferred to several other applications, *e.g.*, GPS sensing systems [BYL+11].

The PermaSense system remained a moving target also during the research that is presented in this thesis. The active usage of available devices and data sets for research purposes has generally been of great help for the early detection of problems of all kind. Furthermore, studies of available data could also deliver valuable input to the design of the current system. First, an analysis of the end-to-end system performance identified that the large majority of data losses was caused by the protocol that has been used for the communication between the field site and the data backend on campus. As such, those results motivated the development of the deployment middleware that is currently in use. Along similar lines, studies had also discovered a severe performance degradation that affected all deployed wireless sensor nodes [KBM+09]. This discovery then led to the software of all deployed sensor nodes to be patched.

1.4 Thesis Outline and Contributions

This thesis presents algorithms and systems for the cleaning, validation, annotation, visualization and augmentation of large sensor network data sets. Results presented are in particular relevant for wireless data collection systems.

The organization of this thesis is as follows (see Figure 1.3): Chapter 2 presents a model-based approach for the cleaning and annotation of data that originates from wireless sensing systems. The purpose of this processing step is to remove data consumers from the burden of individually handling imperfections that have been introduced by the wireless data collection system itself. Data users can thereby fully focus on the processing and analysis that is relevant to their respective domain.

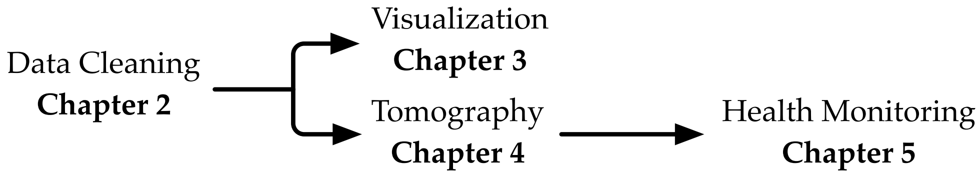


Fig. 1.3 Organization of this thesis

Two concrete examples for the further use of cleaned and annotated data are presented in Chapter 3 and Chapter 4. Chapter 3 presents Vizzly, a system for the visualization of large sensor network data sets in space and time. Exemplary use cases of this system are the visual inspection of data, *e.g.*, manual data verification by a domain expert, and the creation of system management dashboards. Chapter 4 introduces multi-hop network tomography, a method for the reconstruction of the packet path, per-hop ordering and per-hop timing information of individual packets. As a concrete example for the use of that information, Chapter 5 presents hybrid monitoring, a minimally active health monitoring system for wireless sensor networks.

The contributions made by the individual chapters are as follows:

Chapter 2: Data Cleaning and Order Reconstruction

This chapter presents a method for improving data quality by cleaning packets with bogus information and by giving guarantees on accepted packets.

- We present a method that filters packets based on their conformance to a formal model of a data collection system.
- We present a method that reconstructs the proven to be correct order of generation of packets that conform to our model.
- The method is applied to more than 30 million packets that originate from a real system.

Chapter 3: Interactive Browsing of Sensor Data in Space and Time

This chapter presents Vizzly, a system for the interactive browsing of large sensor network data sets in space and time.

- We present a complete system consisting of a cache layer and a front-end application. The proposed system is very flexible and can be attached to arbitrary data repositories.
- The presented cache layer stores pre-computed aggregates of sensor values so that data needed for plotting can be efficiently queried based on a selected time interval and a selected area of interest.
- We evaluate the performance of this system in a production environment. The input of the analyzed instance consists of more than four billion data samples that originate from two large-scale research projects.

Chapter 4: Multi-Hop Network Tomography
This chapter presents a method for uncovering hidden state in wireless sensor networks.

- We present a novel method for the passive reconstruction of the path, per-hop timing and per-hop ordering information of individual packets at runtime.
- The correctness of algorithms presented is formally proven and evaluated in extensive testbed experiments.
- We apply multi-hop network tomography to more than 270 million packets that originate from three real-world sensor network deployments.

Chapter 5: Hybrid Monitoring
This chapter presents a system for the estimation of the number of failure events, *e.g.*, communication failures, that happened inside a wireless low-power network.

- We present a novel health monitoring system that estimates the amount and location of failure events that happened inside a network from the analysis of passively reconstructed packet timing information. Concretely, per-hop timing information that is provided by multi-hop network tomography is complemented with one extra bit of information that is added to every packet inside the network. Setting and transmitting the so called problem bit is the only activity that happens inside the network, all other steps are carried out after packets have been received at the sink.
- The performance of the novel system is evaluated in extensive testbed experiments. In contrast to the sole analysis of passively reconstructed per-hop timing information, we find the accuracy of an exemplary runtime monitoring application to be significantly improved by the single extra bit that is added to every packet.

2

Model-based Data Cleaning and Order Reconstruction

More than a decade ago, researchers envisioned that future sensing systems will consist of massive quantities of cubic-millimeter sized, ubiquitous, wirelessly communicating sensor nodes [KKP99]. Fidelity was not expected to be provided by high-precision components, but by the correlation of many partially redundant, low to medium quality measurements [ECPS02]. Anticipated networks of "smart dust" did not yet become a reality, the design of cubic-millimeter sized devices recently reappeared as an active topic of research [LBL+13].

Though actually existing mote platforms yet miss the anticipated size and energy consumption of "smart dust" devices by orders of magnitude, today's wireless sensor networks have nevertheless proven its applicability and usefulness in many real-world scenarios. Prominent examples are the monitoring of an active volcano [WALJ+06], the monitoring of geophysical processes in the mountains [HTB+08], and the monitoring of old heritage buildings [CMP+09].

In contrast to the original vision, in many sensor network applications the data samples of every single sensor and especially their integrity indeed are of significance. Moreover, data must arrive ordered, sensors are often calibrated, sensor network deployments and their maintenance are labor-intensive and expensive.

For being able to further establish wireless sensor networks as a quality instrument for observation and interaction, we find that the removal of previously mentioned imperfections is not the duty of the final data consumer, but needs to be solved before data is passed over to a user. In this regard, this chapter presents a model-based approach for the filtering of data that originates from multi-hop wireless sensor networks. Packets are filtered based on a formal model of a wireless sensing system, reconstructed sequencing information allows to totally order all packets that passed the formal conformance test.

2.1 Introduction

Looking at a wireless data collection system, there are two major cases that lead to degradation in data quality. First, data generated by sensors and data acquisition equipment may suffer from noise, outliers and inaccuracy due to effects like faulty calibration, stability of power supplies, peculiarities of the system design and others [NRC+09]. Second, artifacts originating from the wireless data transmission system may exist. For example, when analyzing data from a real system [BGH+09] running a highly resource-optimized, energy-efficient data collection protocol, we have observed packet loss, packet duplication, inaccurate timestamps of data generation and wrong packet ordering. Our observations match with the reports of other researchers, *i.e.*, Barrenetxea *et al.* [BISV08, BIS+08] reported an average of 6.5% packet duplicates and up to 20% of lost packets in comparable multi-hop scenarios, others report even worse performance [TPS+05].

While incremental improvements of a system design lead to an improved performance over time, we argue that offline data cleaning, reconstruction and validation are overall valuable and even inevitable for achieving data quality requirements. Firstly, it allows to clean historical artifacts in data derived from, *e.g.*, an initial, yet imperfect system version that are not present in data collected using successive versions of a system. This is very valuable since more sensor data can be utilized despite early imperfections in the realization of a sensor system. Secondly, it will always be the case that a number of situations are not anticipated or accounted for in a system design, leading to sensor data quality degradation or erroneous behavior in certain corner cases. Thirdly, even an “optimal” system design may suffer from fundamental limits. For instance, filtering out all packet duplicates in a streaming network is not realistic due to extensive memory requirements [GFJ+09]. Likewise, out of order packet arrivals can always occur in dynamic multi-hop routing topologies, packet streams must thus be reordered at a higher layer [SRC84]. Lastly, data integrity validation is a valuable tool even if a system is designed and operating correctly.

While there has been extensive research for the first type of problems, *i.e.*, by using statistical methods [EN03], sensor fusion and intense data analysis of the transmitted packet payloads [TPS+05], a systematic approach for the second case described is still missing. Experience has shown that typically users of sensor network data only apply means for filtering of the transmitted packet payloads (*i.e.*, based on data values) and typically do not question the correctness of attributes such as packet header information – as we will argue important indicators of data quality.

We propose to use a two-stage process to improve the quality of data collected in a sensor network. In the first stage, arriving packets are processed by only using the application headers that have been attached and accumulated during packet transmission, *e.g.*, various timestamps, sojourn times throughout the network and various sequence counters. This stage allows to order measurements in the temporal domain, relate measurements to a global notion of time, and to identify packet duplications. In the second stage, data samples are processed using more traditional methods that are typically established in the corresponding application domain and act mainly based on the measurement values themselves, *e.g.*, outlier detection, filtering etc. This chapter covers the first stage.

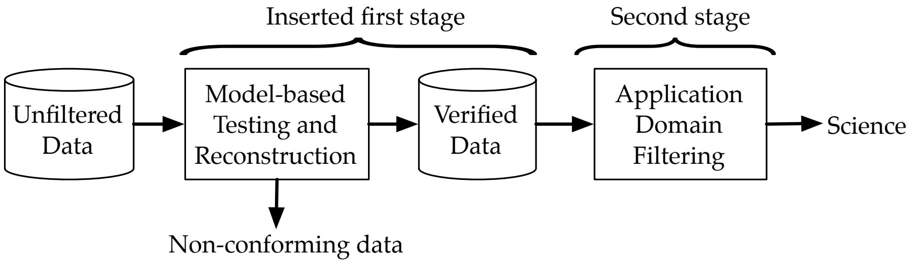


Fig. 2.1 Two-staged process for improving sensor data quality. Arriving packets are first processed by only using application headers before filtering based on sensor data values is applied in the second stage.

In order to provide guarantees on the order of packets received, we propose a model-based approach:

(a) The non-determinism of the overall transmission system including data capture, local clock drifts, reboots, transmission errors, and packet reordering is described in a formal model.
(b) The conformance of packets received with respect to the model is verified. Non-conforming packets are marked as unreliable and excluded from further data analysis.
(c) The correct packet sequence is obtained by using assumptions from our formal model. Conforming packets are annotated with this new sequencing information.
(d) Additional information on the generation of a packet is added to conforming packets inferred from information of temporally adjacent packets.

As an example of this approach, we consider a wireless sensor network application that periodically samples data at a constant rate. Packets can be stored in the network for an arbitrary amount of time. Sensor nodes do not have a global notion of time, local clocks are not synchronized. Generation times of packets are estimated at the sink by subtracting the network sojourn time [KDL$^+$06] from the absolute arrival time. The formal model comprises four scenarios that are common to wireless sensor networks: clock drift, packet duplicates, node reboots, and packet loss.

Based on this formal model, we propose a packet verification and processing approach that provides guarantees on the logical ordering of data. Data that conform to the model are annotated with ordering information and bounds on the time of generation. Thereby, packet duplicates as well as packets that do not conform to the formal system model are marked as such.

The contribution of this chapter can be summarized as follows:

- We introduce an approach for improving data quality by (a) providing a formal system model, (b) verifying conformance of packets received to the model, (c) providing the correct packet sequence, and (d) providing information on the generation of packets inferred from temporally adjacent packets.

- We apply our method to more than 23 months of data from a real-world deployment in an hostile environment. During this time, we collected more than 30 million packets that carry sensor readings and attached application headers.

- A case study is provided validating the usefulness of the proposed intermediate packet processing step. In our validation, we find that our approach successfully reconstructs the correct order of packet data streams. Only a single violation is found when cross-validating a sequence of more than 5 million packets with ground truth from external storage recovered post-deployment. We argue that the subsequent scientific analysis of the environmental data can substantially profit. Here, we especially refer to the problem of not falsely modeling artifacts which have been introduced by the data collection system while designing and calibrating new models of currently only partially understood physical processes.

The remainder of this chapter is organized as follows. An overview of related work is given in Section 2.2. Section 2.3 provides a precise description of the considered problem. In Section 2.4, we present a formal model of a data collection application. Methods for analyzing data originating from systems that conform to this formulated model are presented in Section 2.5. In particular, we consider duplicate filtering, the reconstruction of the generation sequence, and the improvement of timing information of single packets by reasoning with interrelations of multiple packets. For evaluating their performance and practical usefulness, these algorithms are applied to a real data set in Section 2.6. An overview of the broader applicability of our approach is given in Section 2.7.

2.2 Related Work

Data quality and yield have been investigated by many researchers in the community. In particular, literature gives many evidences for approaches in which sensor readings were considered [SMP+04, TPS+05]. The users of data typically remove data that exceeds a threshold given by the sensor specification or identify outliers by applying statistical methods. Orthogonal to work on data cleaning, several data transmission protocols have been evaluated on a very detailed level. But, to the best of our knowledge, this is the first work that approaches data cleaning with a comprehensive formal model of a data collection system that considers a whole set of interacting transmission artifacts. Here, data cleaning is based on application headers gathered during packet generation and forwarding. Our work does not intend to replace the processing of sensor readings. Instead, we propose to add the preceding stage of logical and temporal data filtering before data are finally processed based on the measured physical values.

The problem of reconstructing the temporal order of events has been tackled from different perspectives. First, total message ordering in distributed systems can be achieved using a logical concept of time [Lam78, Mat89]. However, logical time is not sufficient in WSN applications that need to relate events in the physical world [ER03].

Time synchronization protocols such as FTSP [MKSL04b] have been proposed for establishing a global, synchronized time within the sensor network. Ideally, recorded packet generation times immediately represent the temporal order of generation. However, Werner-Allen *et al.* [WALJ+06] reported problems with FTSP in the field. Besides of an reported software bug, especially unstable (wireless) network links caused significant time offsets in the range of hours. More generally, the necessity of stable network links for synchronization is unfortunate for applications that must tolerate high delays and long periods of disconnected operation. For instance, environmental extremes such as ice and snow can force sensor nodes to remain disconnected for several weeks or even months [MOH05].

Non-applicability of network time synchronization protocols has been addressed by the idea of data driven time synchronization. Lukac *et al.* [LDCE09] use microseismics to reconstruct time information, Gupchup *et al.* [GMEST09] developed a similar approach for reconstructing the time from sunlight measurements.

Phoenix [GCME+10] is another recent work dealing with offline time reconstruction. For tackling the problem of sensor nodes losing their local (clock) state due to frequent reboots, the authors propose to exchange time information within the sensor network. An offline algorithm is used to reconstruct global timestamps from this information afterwards.

The work presented in this chapter differs from previous work in two aspects. First, our explicit reconstruction of the generation sequence does not solely rely on either temporal or logical order information, but involves both. This allows us not only to relate events to the physical world, but also to reconstruct causalities in the presence of possibly inaccurate time information. Second, the presented approach does not filter packets based on sensor readings, but on their (non)conformity to a formal model of a real data collection application. For that purpose, we integrated several aspects of data transmission into a single model.

2.3 Packet Classification and Order Reconstruction

Two basic questions are being answered: What are models able to cover the non-deterministic behavior of packet capture and transmission in highly dynamic sensor networks? What are methods that can be used to classify packets received according to their conformance to the model and to reconstruct the correct packet order?

As an example of the overall approach we consider a network of sensor nodes that periodically generate packets. A received packet can be described by the tuple $\{o, s, d, \tilde{t}_s, t_b\}$ consisting of the sender address o, the packet sequence number s, the payload d, the estimated network sojourn time $\tilde{t}_s$, and the absolute time of arrival at the sink t_b. Under a model that covers clock drift, packet duplicates, node reboots and packet loss, packets are classified according to their conformance to the system model. Valid packets are annotated with additional information id_N, t_g^l and t_g^u. Here, id_N represents the temporal order of generation at the source node $o := N$, t_g^l and t_g^u denote upper and lower bounds on the packet generation time.

2.4 System Model

In this section, we introduce a formal model of a sensor network for data collection. Assumptions made are chosen as realistic as possible, but must also contain certain abstractions for providing a solid base needed for deriving correct algorithms in the following Section 2.5. Errors in the assumptions made will lead to a higher amount of data from the real system being non-conforming with respect to the formal model. This is not a particular problem of our model, but a known drawback of modeling in general.

A sensor network for data collection consists of multiple sensor nodes and a sink. For modeling purposes, we abstract a sensor node as a device that offers two services: Packet capturing, *i.e.*, the actual sampling of sensors, and packet forwarding. Packet forwarding is active on all sensor nodes, packet capturing is optional.

Table 2.1 provides a summary of the most important variables that are used in the following. The notation used in this thesis assumes the

Clock and Sensor Node Model	
t	Current time on a perfect clock
t_r	Time on perfect clock at most recent node restart
τ	Locally measured, imperfect time since most recent node restart
$\hat{\rho}$	Worst-case clock drift
t_{reset}	Minimum interarrival time of warm node restarts
i	Internal, unlimited sequence counter
i_{offset}	Offset of internal, unlimited sequence counter
$s_{\max}$	Limit of visible packet sequence counter
T	Sampling period
Packet Application Headers	
$o(k)$	Source node network address
$s(k)$	Packet sequence number
$\tilde{t}_s(k)$	Estimated packet sojourn time
Hidden Packet Information	
$t_g(k)$	Packet generation time
$\rho(k)$	Clock drift during the corresponding sampling interval
$\mathcal{N}_k$	Packet path of packet k
Added on Arrival at the Sink	
$t_b(k)$	Arrival time at the sink
$\tilde{t}_g(k)$	Estimated packet generation time
From Model-based Analysis	
$t_g^{u,l}(k)$	Upper and lower bounds on the unknown packet generation time $t_g(k)$
$id_N(k)$	Packet generation index reflecting the correct order of generation for packets originating from node N

Tab. 2.1 Overview of system model variables

existence of an index that allows the unique identification of every packet that arrived at the sink. One of many practical solutions to achieve this is to simply number all packets that are included in a data set. Most importantly, the identifiers k and l of a packet k and a duplicate l of k are not equal, *i.e.*, $l \not\equiv k$.

2.4.1 Packet Capturing Service

A sensor network contains several sensor nodes that run the packet capturing service. Each instance has the following properties:

- Every node has a source address that is unique in the sensor network. We will use $o(k)$ for referring to the sensor node whose capture service generated a packet k.
- A local clock $\tau := (1+\rho)(t-t_r)$ where ρ denotes the local clock drift, t denotes the absolute time, and t_r denotes the time of the most recent restart of the node. The clock drift is bounded by $\rho \in [-\hat{\rho}, \hat{\rho}]$. Both t and t_r are measured on a perfect clock, none of both is visible to the

sensor node.
- Unplanned warm restarts occur non-deterministically with a minimal interarrival time of t_{reset}.
- Each instance maintains a sequence counter i. The sequence counter is an abstract variable that represents the packet generation sequence. We define the size of i as large enough so that i will never overflow. The sequence counter is initialized exactly once to $i := 0$. After i has been initialized, it supports only reading the current value of i or to increment the value of i by 1.
- Each instance maintains a sequence counter offset i_{offset}. This second abstract variable has the same size as the sequence counter. Once i_{offset} has been initialized to $i_{\text{offset}} := 0$, the value of i_{offset} can only be read or set to $i_{\text{offset}} := i$.
- The sampling period is denoted by T. If $\tau \bmod T \equiv 0$, a packet k of the form $\{o(k), s(k), d(k)\}$ with the source address $o(k)$, a sequence number $s(k)$, and sensor data $d(k)$ is generated. The sequence number is set to

  $$s(k) := (i - i_{\text{offset}}) \bmod s_{\text{max}} \tag{2.1}$$

  where s_{max} bounds the transmitted sequence number. The space for storing and transmitting the sequence number $s(k)$ in a packet is limited, thus the sequence number over-rolls every s_{max}. After the generation of the packet, i is incremented by 1, *i.e.*, $i := i + 1$.
- Due to major faulty behavior, *i.e.*, power failures, cold restarts can occur. Contents stored in volatile memory, *i.e.*, SRAM, are lost after a cold restart. Since certain types of non-volatile memory, *i.e.*, NOR, are not designed for storing frequently changing data, we assume sequence information and packet queues being stored in volatile memory. Thus, a cold restart resets the local clock to $\tau := 0$ and the sequence number of the next packet k to $s(k) := 0$. Our model abstracts the reset of the sequence number $s(k)$ by setting the sequence counter offset i_{offset} to $i_{\text{offset}} := i$. The different behaviors of the model on warm and cold restarts are shown in Table 2.2.

Let us now explain the motivation for the above specification of a packet capturing service. It should generate a packet every T time units, but the time interval T is measured on the local clock and therefore subject to the current clock drift $\rho \in [-\hat{\rho}, \hat{\rho}]$. Using the above model, we can see that for constant reference time t_r and the generation time $t_g(l)$ of the immediate predecessor l the absolute capturing time $t_g(k)$ of a packet k is given as follows:

$$t_g(k) := t_g(l) + \frac{T}{1 + \rho(k)} \tag{2.2}$$

Sensor nodes can restart during operation. This can either be planned, *i.e.*, a reset button push, or unplanned, *i.e.*, the software watchdog

After packet capture:	After warm restart:	After cold restart:
$i := i + 1$	$t_r := t$	$t_r := t$
	$\Rightarrow \tau := 0$	$i_{\text{offset}} := i$
		$\Rightarrow \tau := 0$
		$\Rightarrow s(k) := 0$

Tab. 2.2 The state of the local clock is lost on restarts. Additionally losing SRAM contents in case of a cold restart also causes the sequence number of the next packet k to be reset to $s(k) := 0$.

resetting the sensor node due to an overrun of the task queue [KBM$^+$09]. At this point in time, the clock state is lost and starts again at $\tau := 0$. In the case of a restart, sensor nodes immediately continue sampling after initialization. As a consequence, considering sequence counter value $i-1$ immediately before, and i just after a restart, the time difference between the corresponding packets k and l can be much smaller than the sampling period T. As a restart may occur directly after the generation of a packet, we find

$$0 < t_g(k) - t_g(l) \leq \frac{T}{1 + \rho(k)} \tag{2.3}$$

2.4.2 Forwarding Network

Sensor nodes interact in a forwarding network that transmits packets to a sink using multi-hop routing. The sink S immediately processes arriving packets, it is the only component of the sensor network that has a global notion of time, the clock of the sink is perfect. We model the forwarding network as follows:

- It immediately reads every packet $\{o(k), s(k), d(k)\}$ that has been generated by a sensor node that runs the capturing service.
- A packet k is delivered to the sink S after a sojourn time $t_s(k)$.
- The forwarding network can duplicate packets arbitrarily, *i.e.*, it can generate an arbitrary number of copies from $\{o(k), s(k), d(k)\}$. These packets are forwarded independently from each other.
- It can delete packets arbitrarily, *i.e.*, a packet $\{o(k), s(k), d(k)\}$ is removed and not delivered to the sink S.
- The forwarding network augments captured packets with information about the transmission. It outputs packets of the form $\{o(k), s(k), d(k), \tilde{t}_s(k), t_b(k)\}$ where $\{o(k), s(k), d(k)\}$ was the captured packet, $\tilde{t}_s(k)$ is an estimate of the sojourn time $t_s(k)$ and $t_b(k)$ is the absolute time of arrival at S. The estimated sojourn time $\tilde{t}_s(k)$ satisfies

  $$(1 - \hat{\rho}) \cdot t_s(k) - \hat{h} \cdot \hat{t}_u < \tilde{t}_s(k) \leq (1 + \hat{\rho}) \cdot t_s(k) \tag{2.4}$$

  where $\hat{\rho}$ is the bound on the local clock drifts. With $\hat{t}_u$ as the clock resolution of the local clock, sensor nodes measure time differences with an uncertainty in the interval $(-\hat{t}_u, 0]$. This uncertainty is introduced per hop, the maximum number of hops towards the sink is denoted by $\hat{h}$.

Again, let us now provide the motivation for the above model of the (packet) forwarding network. The sensor nodes are organized in a dynamic multi-hop tree topology where packets are transported over multiple hops until they are finally received by the sink S.

During a one-hop communication, the receiving node sends a receipt to acknowledge the transmission over one hop. A packet is retransmitted as long as the acknowledgement of the next hop did not arrive within an expected time frame. Packet duplicates are generated if an acknowledgement was not received although the transmission of the packet itself was successful.

Furthermore, packet loss is also a well-known problem in the context of real-world applications. For instance, pending packets waiting for transmission are lost if the contents of the local packet queue of a sensor node are (fully or partially) lost due to a cold restart. As another example, packets may also be dropped if the limited local packet queue is full.

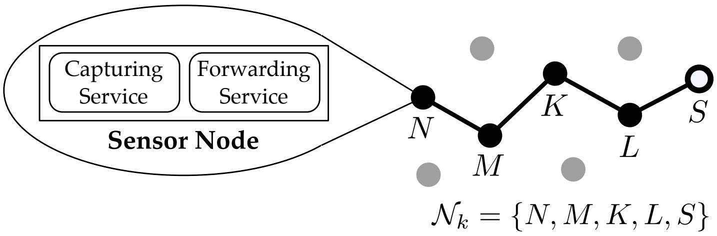


Fig. 2.2 Travel of a packet k that was generated at sensor node N. The packet is processed by all sensor nodes in $\mathcal{N}_k$ before it is finally received by the sink.

The sojourn time $t_s(N, k)$ is the time that packet k spent in the packet queue of some sensor node N of the forwarding network. We define $\mathcal{N}_k$ as the set of nodes that process a packet k during its travel from its source to the sink. Figure 2.2 describes the exemplary travel of a packet from node N to the sink S. The total sojourn time $t_s(k)$ of a packet is thus calculated as

$$t_s(k) := \sum_{N \in \mathcal{N}_k} t_s(N, k) \tag{2.5}$$

During packet transmission, the sensor nodes accumulate the sojourn times between packet reception and transmission. These times are determined using the local clocks which have a bounded drift. Therefore, at the sink there is only an estimate $\tilde{t}_s(k)$ of the sojourn time $t_s(k)$ for each packet available which is

$$\tilde{t}_s(k) := \sum_{N \in \mathcal{N}_k} \tilde{t}_s(N, k) \tag{2.6}$$

where $\tilde{t}_s(N,k)$ is the locally determined estimate of the sojourn time of packet k in node N. Considering uncertainties due to the drift as well as the resolution of the local clock, we have $\tilde{t}_s(N,k) \in ((1-\hat{\rho}) \cdot t_s(N,k) - \hat{t}_u, (1+\hat{\rho}) \cdot t_s(N,k)]$. The resulting interval for $\tilde{t}_s(k)$ is presented in (2.4).

2.5 Data Analysis

The goal of the analysis is 1) to identify and exclude packet duplicates, 2) to test the conformance of packets received to the specified system model, 3) to exclude packets that are not conforming, and 4) to annotate the data set with additional information that provide the correct packet sequence.

The analysis consists of four steps that are explained in more detail in the following sections. The presented sequence is inferred from the order in which the data set must either be annotated with extra information or reduced in its size for fulfilling the assumptions of subsequent steps:

- The time interval in which a packet has been generated is initially calculated for all packets.
- Packet duplicates are removed from the data set. The goal of duplicate filtering is to maximize the number of accepted packets while guaranteeing that all duplicates are removed.
- An epoch assignment algorithm is applied to the filtered data. Assigning packets to so called "epochs" is a method for reconstructing the temporal order of packet generation based on application header information.
- Generation time intervals of ordered and filtered packets are further improved by forward and backward reasoning.

2.5.1 Packet Generation Time Intervals

The estimated packet generation time $\tilde{t}_g(k)$ is given by subtracting the estimated sojourn time of a packet $\tilde{t}_s(k)$ from the arrival timestamp $t_b(k)$.

$$\tilde{t}_g(k) := t_b(k) - \tilde{t}_s(k) \tag{2.7}$$

We have to resort to estimates of the packet generation time as information to reconstruct the exact generation time is missing. Concretely, the value of the local clock τ on packet generation and involved clock drifts $\rho(k)$ while a packet travels through the forwarding network are not known at the sink. From the perspective of the perfect clock at the sink, we get the estimation error

$$t_g(k) - \tilde{t}_g(k) \in \left[-\frac{\tilde{t}_s(k) \cdot \hat{\rho} + \hat{h} \cdot \hat{t}_u}{1-\hat{\rho}}, \frac{\tilde{t}_s(k) \cdot \hat{\rho}}{1+\hat{\rho}} \right] \tag{2.8}$$

This equation firstly addresses the introduced worst-case error of measuring the packet sojourn time $\tilde{t}_s(k)$ on local clocks with drift. Secondly, $\hat{h} \cdot \hat{t}_u$ describes the worst-case error when accumulating time measurements of at most $\hat{h}$ sensor nodes with an uncertainty of $(-\hat{t}_u, 0]$ per hop. Based on these bounds, we can determine the valid range of the packet generation time $t_g(k)$

$$t_g(k) \in [t_g^l(k), t_g^u(k)] \tag{2.9}$$

where

$$t_g^u(k) := t_b(k) - \frac{\tilde{t}_s(k)}{1+\hat{\rho}} \tag{2.10}$$

$$t_g^l(k) := t_b(k) - \frac{\tilde{t}_s(k) + \hat{h} \cdot \hat{t}_u}{1-\hat{\rho}} \tag{2.11}$$

In Section 2.5.4, we will introduce forward and backward reasoning for improving $t_g^u(k)$ and $t_g^l(k)$.

2.5.2 Duplicate Filtering

The goal of the duplicate filtering step is to remove all packet duplicates from a data set. Packet duplicates are packets that are equal with at least one other packet in terms of the following three properties: 1) Packet duplicates have the same source address $o(\dots)$, 2) packet duplicates have the same sequence number $s(\dots)$, and 3) packet duplicates have an equal payload $d(\dots)$. Since packet duplicates travel through the network independently, they may have different estimated sojourn times $\tilde{t}_s(\dots)$, but they will have overlapping generation time intervals $[t_g^l(\dots), t_g^u(\dots)]$.

Based on this definition, we now explain our duplicate filtering mechanism. Here, we consider a subset $\mathcal{D}$ of the whole data set that only includes packets with an equal source address $o(\dots)$, an equal sequence number $s(\dots)$ and equal payloads $d(\dots)$. It becomes apparent, that any possible subset with these properties can be handled independently. The subset is duplicate-free, if all included packets have disjoint generation time intervals $[t_g^l(\dots), t_g^u(\dots)]$.

For finding duplicate-free subsets, we consider the problem of finding the maximum independent set of a graph. We consider a graph $G := (V, E)$ with the set of vertices V and the set of edges E. In short, the maximum independent set I of G is the largest subset $I \subseteq V$ that contains only vertices that are not connected to any other vertex of the subset I. For our application, each packet being member of $\mathcal{D}$ is represented by a vertex $v \in V$. Two vertices v and w are connected by an edge $(v, w) \in E$, if the corresponding packets have overlapping generation time intervals:

$$(v, w) \in E \Leftrightarrow (t_g^u(v) \geq t_g^l(w)) \wedge (t_g^u(w) \geq t_g^l(v)) \tag{2.12}$$

In summary, duplicate filtering starts with separating a data set into subsets of a fixed originator $o(\dots)$, a fixed sequence number $s(\dots)$, and a fixed payload $d(\dots)$. All subsets are analyzed independently. Firstly, the corresponding graph G of the subset is constructed. Then, we employ a standard algorithm [Lub85] for finding a maximum independent set I. Packets that correspond to a vertex $v \in I$ are kept, packets corresponding to a vertex $v \in V \setminus I$ are marked as packet duplicates and not considered in the further analysis.

Without further assumptions on the analyzed data set, it is not possible to avoid packets falsely being marked as duplicates. Firstly, we do not state any restrictions on the payload $d(\dots)$. Thus, the payload can also be constant for an arbitrary number of packets without any packet duplications being involved. Secondly, a power failure can lead to two consecutively generated packets k and l having an equal sequence number $s(k) \equiv s(l) \equiv 0$. Concerning the trade-off between accepting false positives and accepting false negatives, our superior goal of ensuring a duplicate-free data set allows us only to tolerate packets being falsely removed. From now on, we suppose that the packet streams are free of duplicates.

2.5.3 Epoch Assignment

In this section, we present and proof the core foundations of our proposed packet sequence reconstruction step. We propose to assign packets to epochs for reconstructing their temporal order of generation. The following analysis first supposes that there are no cold restarts which re-initialize the sequence number with $i_{\text{offset}} := i$, only warm restarts that reset the timer of the capturing service are allowed. The effect of cold restarts will be discussed at the end of the section.

Considering data from a single sensor node, we are now briefly explaining the concept of separating data into epochs:

- All packets being generated between two consecutive resets of the sequence number $s(k) := (i - i_{\text{offset}}) \bmod s_{\max}$ belong to the same epoch. An epoch embraces up to $s_{\max}$ sequentially generated packets, any two packets belonging to the same epoch have disjoint packet sequence numbers $s(k)$. More precisely, subsequently generated packets $k^{(0)}, k^{(1)}, k^{(2)}, \dots, k^{(L-1)}$ belong to the same epoch if $s(k^{(0)}) \equiv 0$, $s(k^{(j)}) \equiv j$ for all $0 \leq j < L$, and $s(k^{(L)}) \equiv 0$.
- Epochs are labeled with an incrementing index, *i.e.*, $e \in \mathbb{N}$, the index of the corresponding epoch of a packet k is denoted by $e(k)$.

From this definition of an epoch, we can derive the following statement: *The epoch numbers $e(k)$ and $e(l)$ of two packets k and l satisfy $(e(k) < e(l)) \vee ((e(k) \equiv e(l)) \wedge (s(k) < s(l)))$ if and only if k was generated before l, i.e., $t_g(k) < t_g(l)$.*

We will now provide a method to assign packets uniquely to epochs which leads to a total order according to the previous theorem. The main concept is based on the notion of the "epoch center" $T_c(k)$ of a packet k which is computed offline according to

$$T_c(k) := \tilde{t}_g(k) - s(k) \cdot T \tag{2.13}$$

where $\tilde{t}_g(k) := t_b(k) - \tilde{t}_s(k)$ denotes the estimated generation time of packet k. In order to explain the concept, let us first suppose that there are no restarts, no measuring inaccuracies of time differences and no clock drifts. Then, the estimated generation time of a packet equals the actual one, *i.e.*, $\tilde{t}_g(k) \equiv t_g(k)$, and the time difference between subsequent packets is T. Therefore, $T_c(k) \equiv T_c(l)$ holds for all packets of an epoch $e(k) \equiv e(l)$, *i.e.*, all packets of an epoch have the same "epoch center". Using the above assumptions, the time difference between subsequent "epoch centers" is simply $s_{\max} \cdot T$.

Of course, node restarts and clock drifts will change the above scenario as (a) the virtual "epoch centers" of packets belonging to one epoch are not equal and (b) the time differences between subsequent epoch centers are not $s_{\max} \cdot T$ anymore.

The concept of the epoch assignment algorithm can be described as follows: *All packets whose "epoch centers" are close enough are assigned to the same epoch, whereas packets whose "epoch centers" have a large distance are assigned to different epochs.* The following two theorems that allow for warm restarts formalize the above notions.

Theorem 2.1. *All packets k, l that belong to the same epoch, i.e., $e(k) \equiv e(l)$, satisfy*

$$|T_c(k) - T_c(l)| \leq \Delta T_c \tag{2.14}$$

where

$$\Delta T_c := (s_{max} - 1)(\hat{\rho}T + T - T') + T' + 2\hat{\rho}t_s^{max} \tag{2.15}$$

where t_s^{max} is an upper bound on the network sojourn time, i.e., $t_s(k) \leq t_s^{max}$ and

$$T' := \frac{1}{(1+\hat{\rho})/T + 1/t_{reset}} \tag{2.16}$$

Proof. For two packets of the same epoch, we find

$$\begin{aligned} T_c(k) - T_c(l) &= \tilde{t}_g(k) - \tilde{t}_g(l) - s(k) \cdot T + s(l) \cdot T \\ &\leq t_b(k) - t_b(l) - (1-\hat{\rho})t_s(k) \\ &\qquad + (1+\hat{\rho})t_s(l) - s(k) \cdot T + s(l) \cdot T \\ &\leq (t_g(k) - s(k) \cdot T) - (t_g(l) - s(l) \cdot T) + 2\hat{\rho}t_s^{\max} \end{aligned} \tag{2.17}$$

Neglecting second order drift influences, we can upper bound the first term as

$$\begin{aligned} t_g(k) - s(k) \cdot T &= t_g(m_0) + s(k)\frac{T}{1-\hat{\rho}} - s(k) \cdot T \\ &\leq t_g(m_0) + (s_{\max} - 1)\hat{\rho}T \end{aligned} \tag{2.18}$$

where m_0 denotes the first packet of the epoch. The lower bound on the second term is obtained by a packet generation that is as fast as possible. In other words, we first need to determine a lower bound B on time difference between $s_{\max}$ packets. As we know, the minimal interarrival time of unplanned warm restarts is t_{reset} and at each restart, the generation clock is reset and a packet is generated. As a result, B can be determined as the smallest value that satisfies

$$\begin{aligned} B &\geq \left(s_{\max} - 1 - \left\lceil \frac{B}{t_{\text{reset}}} \right\rceil\right) \frac{T}{1+\hat{\rho}} \\ &\geq \left(s_{\max} - 2 - \frac{B}{t_{\text{reset}}}\right) \frac{T}{1+\hat{\rho}} \end{aligned} \tag{2.19}$$

Solving this equation for B and using the abbreviation

$$T' := \frac{1}{(1+\hat{\rho})/T + 1/t_{\text{reset}}} \tag{2.20}$$

yields a lower bound

$$B := (s_{\max} - 2)T' \tag{2.21}$$

Now we can use this bound in order to determine

$$t_g(l) - s(l) \cdot T \geq t_g(m_0) + (s_{\max} - 2)T' - (s_{\max} - 1)T \tag{2.22}$$

As a result, we find now

$$T_c(k) - T_c(l) \leq (s_{\max} - 1)(\hat{\rho}T + T - T') + T' + 2\hat{\rho}t_s^{\max} \tag{2.23}$$

which finishes the proof.

□

Theorem 2.2. *Suppose that the generation period T satisfies*

$$T > 2(1+\hat{\rho})\frac{\Delta T_c}{s_{max}} \tag{2.24}$$

Then all packets k, l that belong to different epochs, e.g., $e(k) < e(l)$, satisfy

$$T_c(l) - T_c(k) > \Delta T_c \tag{2.25}$$

where ΔT_c is defined in Theorem 2.1.

Proof. The proof uses results from the proof of Theorem 2.1. In particular, we know that

$$T_c(k) - T_c(l) \leq (t_g(k) - s(k) \cdot T) - (t_g(l) - s(l) \cdot T) + 2\hat{\rho}t_s^{\max} \tag{2.26}$$

If $e(l) \equiv e(k) + 1$, then we can not use the same reference time $t_g(m_0)$ of the first packet of the common epoch anymore. Instead, the reference points of packets k and l differ by at least $s_{\max} \cdot T/(1+\hat{\rho})$. Therefore, we find

$$T_c(k) - T_c(l) \leq -s_{\max}\frac{T}{1+\hat{\rho}} + \Delta T_c \tag{2.27}$$

By using $T_c(l) - T_c(k) > \Delta T_c$ we finally obtain

$$s_{\max} \frac{T}{1+\hat{\rho}} > 2\Delta T_c \tag{2.28}$$

which leads to the condition in the theorem.

□

The condition on the nominal generation period T in Theorem 2.2 involves the maximal sojourn time of packets $t_s^{\max}$. Therefore, given a generation period T, the minimal restart time interval t_{reset}, the maximal clock drift $\hat{\rho}$, and the maximal sequence number $s_{\max}$, one can determine an upper bound on the sojourn time of packets which would allow for an epoch assignment based on the above theorems:

$$t_s^{\max} < \frac{1}{2\hat{\rho}} \left(\frac{s_{\max} \cdot T}{2(1+\hat{\rho})} - (s_{\max} - 1)(\hat{\rho}T + T - T') - T' \right) \tag{2.29}$$

This bound can be used to mark (or remove) packets that cannot be assigned to epochs due to their sojourn time.

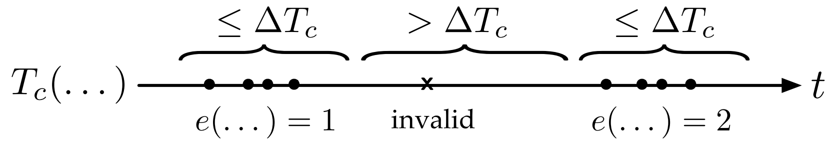


Fig. 2.3 Assignment of packets to epochs. Epoch centers $T_c(k)$ and $T_c(l)$ of packets k and l of the same epoch must lie within ΔT_c, see (2.14). In turn, epoch centers $T_c(k)$ and $T_c(l)$ of two packets k and l that belong to disjoint epochs must be at least ΔT_c apart from each other, see (2.25).

The two theorems also allow to classify packets that do not conform to the system model. In particular, suppose that we look at three packets k, l, and m where $T_c(k) - T_c(l) \le \Delta T_c$ and $T_c(l) - T_c(m) \le \Delta T_c$. In this case, k, l, and m need to belong to the same epoch due to Theorem 2.1. But if $T_c(k) - T_c(m) > \Delta T_c$, then k and m can not belong to the same epoch due to Theorem 2.2 which is a contradiction. The following Algorithm 2.1 uses this fact to mark or remove such packets.

The question arises, why packets could violate Theorems 2.1 and 2.2. As described in the formal model, there may be cold restarts, *i.e.*, a restart that also causes a reset of the sequence number. In this case, epochs contain a smaller number of packets, and therefore, may have a smaller distance in time. Using the proof of Theorem 2.2, we can infer that the minimal distance between two epochs, *i.e.*, the timing distance of a cold restart from the beginning of an epoch, needs to be larger than $2 \cdot \Delta T_c$ in order to guarantee a sufficient separation of epochs.

At first, all packets that do not satisfy the bound on the sojourn time in (2.29) are removed. The estimated epoch centers $T_c(k)$ are calculated for all packets. Algorithm 2.1 firstly checks for packets that violate Theorems 2.1

Algorithm 2.1: Annotation of epoch $e(k)$ and index $id_N(k)$ to all packets k. *pop*() returns the next packet from the ordered data set, or *false* if all packets have been pulled.

```
input  : Packets of a single node $N$, ordered by increasing $T_c(k)$
output: Packets with annotated epoch $e(k)$ and index $id_N(k)$
begin
    $epoch \longleftarrow 0$ ; $l \longleftarrow pop()$ ; $f \longleftarrow l$ ;
    while $k \longleftarrow pop()$ do
        if $T_c(k) - T_c(f) > \Delta T_c$ then
            if $T_c(k) - T_c(l) \leq \Delta T_c$ then
                mark packet $k$ as non-conforming ; continue ;
            else
                $epoch \longleftarrow epoch + 1$ ; $f \longleftarrow k$ ;
            end
        end
        $e(k) \longleftarrow epoch$ ; $id_N(k) \longleftarrow epoch \cdot s_{\max} + s(k)$ ; $l \longleftarrow k$ ;
    end
end
```

and 2.2. Non-violating packets are then annotated with the index of the corresponding epoch $e(k)$ and an index $id_N(k)$ which reflects the ordering of packets, *i.e.*, it satisfies $id_N(k) > id_N(l)$ if $t_g(k) > t_g(l)$. After executing Algorithm 2.1, all packets that have the same index $id_N(\dots)$ and epoch $e(\dots)$ also need to be marked as non-conforming.

2.5.4 Forward and Backward Reasoning

In Section 2.5.1, we introduced $t_g^u(k)$ and $t_g^l(k)$ as upper and lower bounds of the valid range of the unknown, perfect packet generation time $t_g(k)$. The goal of the presented forward and backward reasoning is to refine these time intervals by exploiting the sequence information provided by the index $id_N(\dots)$ computed in the previous section. This way, we take into account sequence and timing information of the whole packet stream.

For the following discussion, we suppose that packets are processed separately based on their equal origin $o(k) := N$. The basis for the algorithm are bounds on the time difference between the generation of packets k and l with $id_N(k) < id_N(l)$

$$0 < t_g(l) - t_g(k) \leq (id_N(l) - id_N(k)) \frac{T}{1 - \hat{\rho}} \tag{2.30}$$

where the lower bound is due to the possibility of node restarts and the upper bound is due to a slow clock at the sensor node. Now, we can tighten the upper and lower bounds by applying the above relation iteratively for all packets.

The tightening algorithm applies (2.30) iteratively starting from the initial upper and lower bounds, see (2.31)-(2.34). Note that the upper and lower bounds are treated independently. In addition, we need only one pass for each iteration. For the lower bounds, the iteration finished if we execute firstly (2.31) in the order of the increasing packet generation index $id_N(\dots)$ and then (2.33) in the order of the decreasing index $id_N(\dots)$. Likewise, the improved upper bounds are determined after firstly executing (2.34) in the order of the decreasing packet generation index $id_N(\dots)$ and then (2.32) in the order of the increasing index $id_N(\dots)$.

(Forward reasoning)

$$t_g^l(k) := \max\left(t_g^l(k), t_g^l(m)\right) \tag{2.31}$$

$$\forall l \in \mathcal{L} : t_g^u(k) := \min\left(t_g^u(k), t_g^u(l) + (id_N(k) - id_N(l))\frac{T}{1-\hat{\rho}}\right) \tag{2.32}$$

$$\textbf{with } \mathcal{L} := \{l \mid id_N(l) < id_N(k)\} \textbf{ and}$$

$$m := \arg\max_x id_N(x) \equiv id_N(k) - 1$$

(Backward reasoning)

$$\forall l \in \mathcal{L} : t_g^l(k) := \max\left(t_g^l(k), t_g^l(l) - (id_N(l) - id_N(k))\frac{T}{1-\hat{\rho}}\right) \tag{2.33}$$

$$t_g^u(k) := \min\left(t_g^u(k), t_g^u(m)\right) \tag{2.34}$$

$$\textbf{with } \mathcal{L} := \{l \mid id_N(l) > id_N(k)\} \textbf{ and}$$

$$m := \arg\max_x id_N(x) \equiv id_N(k) + 1$$

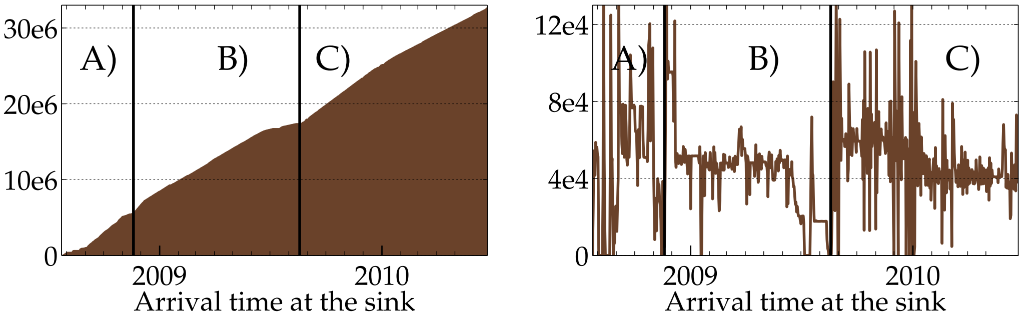
If we use this order of execution, then in (2.31) and (2.34) we only need to take the nearest neighbor into account. In addition, no fixed point iteration is necessary. Let us show this for $t_g^l(\dots)$ only, as the other case can be handled similarly.

Obviously, after the forward phase, we have $t_g^l(k) \leq t_g^l(l)$ for all l with $id_N(l) > id_N(k)$, and after the backward phase, (2.33) holds for all packets k. Therefore, we only have to show that $t_g^l(k) \leq t_g^l(l)$ for all $l : id_N(l) > id_N(k)$ still holds after the backward phase. Suppose that this is not the case, *i.e.*, there exist some k and l with $id_N(l) \equiv id_N(k) + 1$ such that $t_g^l(k) > t_g^l(l)$. Then there must exist a packet m with $id_N(m) > id_N(k)$ which increased the bound for k in the backward phase to the new (larger) value, *i.e.*, $t_g^l(k) = t_g^l(m) - \Delta_1$ for some positive value Δ_1. $id_N(m) \equiv id_N(k) + 1 \equiv id_N(l)$ is not possible as $t_g^l(l) < t_g^l(k)$ and therefore, we have $id_N(m) > id_N(l)$. Therefore, during the backward phase, packet m also may have changed packet l: $t_g^l(l) \geq t_g^l(m) - \Delta_2$ for some positive value Δ_2. As $id_N(m) - id_N(k) > id_N(m) - id_N(l)$ we have $\Delta_1 > \Delta_2$. Now, we can write $t_g^l(k) = t_g^l(m) - \Delta_1 < t_g^l(m) - \Delta_2 \leq t_g^l(l)$ which contradicts the assumption.

If after the execution of the algorithm there are packets k with $t_g^u(k) < t_g^l(k)$, those packets will be marked as non-conforming and removed. After that, the tightening algorithm is applied again. This way, we finally achieve a packet stream that is conforming to the formal model.

2.6 Case Study

The PermaSense project [HTB+08] strives for modeling physical processes related to high-alpine permafrost that have previously only been partly understood by the environmental science community. For verifying new physical models, considerable volumes of highly accurate measurements collected over a multi-year period are necessary. Observations are typically taken at remote locations that offer no existing infrastructure. Furthermore, extremely harsh environmental conditions, especially ice and snow, allow to visit the field locations only within a certain time period of the year. Here, the approach is to deploy highly energy-optimized wireless sensor nodes that are designed for a reliable operation under these conditions. For highest data quality, a purpose-built sensor interface board is used to interface expensive high-precision instruments [BGH+09].



(a) Total number of packets **(b)** Number of packets per day

Fig. 2.4 Number of packets received at the sink of the multi-hop network. Each packet carries a number of sensor readings and application headers. The three analyzed phases of the PermaSense Matterhorn deployment are denoted with A) to C).

For this case study we consider a data set that consists of more than 30 million packets that have been gathered during a 23 month period. Each packet carries a number of sensor readings and packet header data. Within the observed time span, the deployment consisted of up to 19 sensor nodes and a single sink. There are five different packet types that include the same set of application headers, but different types of sensor readings. The system is designed to generate a packet of each type every two minutes. These five packets are generated in immediate succession one after the other, an individual, local timestamp and a unique sequence number are added to every packet immediately after generation. A multi-hop data collection protocol [BvRW07] is used to transport the data from the sensor nodes to the sink where the accumulated sojourn time of each packet and the absolute reference time of the sink are used to calculate the generation time of the packet.

The PermaSense deployment used in this case study has been initially set up in July 2008. The time from July 2008 until May 2010 can be

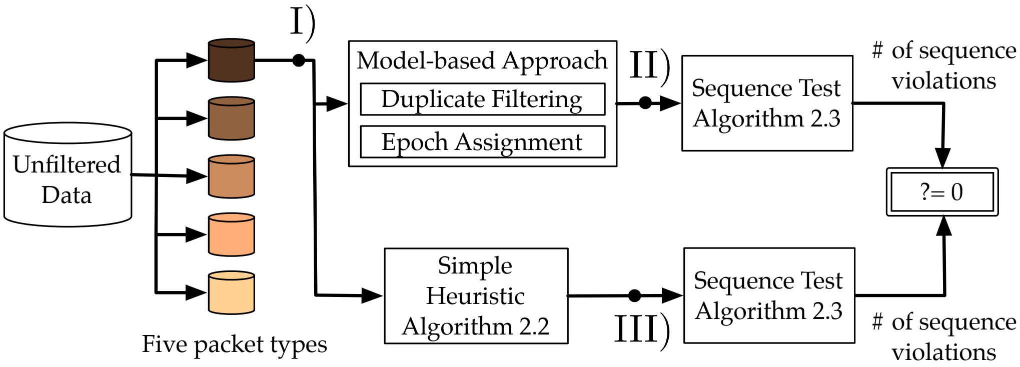


Fig. 2.5 Case study validation strategy. We verify and compare the correctness of the packet sequences resulting from applying the model-based approach and a simple heuristic to the data set. Packet sequences are validated by a model of the behavior of the node uptime measurement. This measurement is included in *health packets*, one out of five packet types. Algorithm 2.3 returns the number of sequence violations based on this model, ideally this should always be zero. We validate results on the whole packet stream with Algorithm 2.3, the results on the deployment phases B) and C) are additionally cross-validated with ground truth from recovered external storage. Annotations I) to III) correspond to probes that we add for reference in the remainder of this chapter.

split into three different deployment phases that can be characterized by different system behaviors. We will exploit this history for evaluating the performance of our model-based approach in the following three scenarios: A) Highly non-conforming system behavior, B) sensor nodes subject to a high frequency of unplanned warm restarts, and C) more than one third of the collected data experiencing transmission delays of several hours to days.

In more detail, the first four months of the deployment were mainly determined by initial tests of new hardware and software. Artifacts of this non-conforming system behavior during phase A) are shown in Figure 2.4(b). The number of packets received per day is varying over time, the total number of packets received during this phase violates system specifications. Learning from problems caused by outages of the sink on site and the database server in the backend, the installed sensor nodes were completely replaced by new sensor nodes with additional external storage to accommodate sensor data backups in November 2008. In March 2009, an initial data analysis identified a severe software problem that caused all sensor nodes to restart up to 40 times per day [KBM+09]. The resulting long-term effect of "dying" sensor nodes is observable in the drop of the packet reception rate at the end of phase B), see Figure 2.4(b). This issue was fixed by installing a new sensor node software image in September 2009. In the following phase C), lack of enough solar power at the sink node often led to a nightly power cut-off at the sink. Sensor data packets were buffered in the network for the duration of the power outage and then flushed in burst mode to the sink upon restoration of the network topology. The resulting bursty behavior is also observable in Figure 2.4(b). While certainly an undesired behavior, the unique system design of Dozer [BvRW07] allowed such long-term operation with only little extra energy cost and no observed data loss.

For clarity and brevity, we limit our case study to the analysis of only one out of five packet types, namely *health packets*. Extending our system model to support five packets being generated in each sampling period is straightforward. However, limiting this case study to only one single packet type facilitates a clear understanding of the core features of our approach.

After giving a short overview of our implementation of the model-based approach used during this case study, we introduce a simple heuristic that is used as a reference for evaluating the performance of our model-based filtering approach. Then, we introduce the number of sequence violations as metric for quantifying the correctness of a packet sequence. We then evaluate the performance in terms of accepted packets and the correctness of the retrieved packet sequence. The entire packet sequence validation strategy is depicted in Figure 2.5. An evaluation of the achieved gain by applying forward and backward reasoning concludes this case study.

2.6.1 Case Study Implementation

The results shown in this case study are based on a MATLAB implementation of the algorithms presented in Section 2.5. For processing our data set with the model-based approach, we use the parameter set which is shown in Table 2.3. The analysis runs on a standard PC system, data is currently fetched from an external MySQL database server. After neglecting the time spent for fetching the data from the database server, the execution time for analyzing the whole data set is less than one hour on a single processing core.

PARAMETER	VALUE
Sampling period T	120 sec
Maximum clock drift $\hat{\rho}$	±60 ppm
Clock resolution $\hat{t}_u$	1 sec
Maximum hop distance $\hat{h}$	4
Restart interarrival time t_{reset}	0.6 hours
Packets per epoch $s_{\max}$	2^{16}

Tab. 2.3 PermaSense system parameters

2.6.2 Comparison to a Simple Heuristic

In this case study, we evaluate the performance of our model-based approach using three metrics: 1) Packet acceptance rate, 2) correctness of retrieved packet sequence and 3) improvement of generation time intervals by applying forward and backward reasoning. We evaluate the first two metrics using a comparison with a simple heuristic for retrieving an ordered packet sequence. The third metric will be evaluated standalone.

Algorithm 2.2: Simple heuristic for finding a packet sequence that is ordered by the packet sequence number. *pop*() returns the next packet from the input data set, or *false* if all packets have been pulled.

```
input  : Packets of a single node, ordered by ascending $\tilde{t}_g$
output: Set of packets $R$, ordered by packet sequence number
begin
    $R \longleftarrow \emptyset$ ; $l \longleftarrow pop()$ ;
    while $k \longleftarrow pop()$ do
        if $s(k) > s(l)$ or $s(l) - s(k) > 0.8 \cdot s_{max}$ then
            $R \longleftarrow R \cup \{k\}$ ; $l \longleftarrow k$ ;
        end
    end
end
```

Algorithm 2.2 implements a simple heuristic for retrieving a packet sequence that is ordered by the packet sequence number. The algorithm uses a best effort approach to detect overflows of the sequence number counter, the sequence number is strictly increasing between two subsequent overflows. A packet sequence number overflow is assumed if the sequence number of a following packet is smaller than the sequence number of its predecessor. Setting the threshold for detecting an overflow to $0.8 \cdot s_{\text{max}}$ allows to detect an overflow in the presence of a considerable amount of packet loss.

2.6.3 Counting Sequence Violations

We now explain how we evaluate the correctness of a packet sequence. For that purpose, we introduce the metric of sequence violations which corresponds to the number of conflicting packets when comparing a packet sequence under test with a baseline.

For the validation of this case study, we recovered external storage cards containing duplicate sensor data packets for obtaining the ground truth of the packet sequence. The data recovered from external storage spans from November 2008 to May 2010 which allows us to evaluate two of three deployment phases with this method. We derive from our system specification that packets are sequentially appended to the external storage in the correct sequence of packet generation. The number of sequence violations is the result of comparing a packet sequence under test with the packet sequence from external storage.

For evaluating the correctness of a packet sequence prior to November 2008, we have to resort to a generated baseline. Here, we evaluate the correctness of a packet sequence by testing how good an extracted,

Algorithm 2.3: Algorithm for counting sequence violations based on a model of the node uptime measurement. *pop*() returns the next packet from the ordered data set, or *false* if all packets have been pulled.

```
   input  : Health packets of a single node, ordered by the sequence under test
   output: Number of sequence violations $v$
 1 begin
 2     $T \longleftarrow \emptyset$ ; $v \longleftarrow 0$ ; $l \longleftarrow pop()$ ;
 3     while $k \longleftarrow pop()$ do
 4         if $u(k) > u(l)$ or $(u(k) < u(l)$ and $u(k) < u_{reset})$ then $T \longleftarrow T \cup k$;
 5         else
 6             if $|T| < c_{min}$ then $v \longleftarrow v + |T|$;
 7             $T \longleftarrow \emptyset$ ;
 8         end
 9         $l \longleftarrow k$ ;
10     end
11 end
```

measured signal conforms with a model of this measurement. As an exemplary measurement, we employ the node uptime for testing the correctness of a packet sequence. The node uptime is a local counter of a sensor node, its value is included in health packets, one out of five packet types. Compared to other transmitted measurements that mainly correspond to observations of complex physical processes, the behavior of the node uptime can be described by a simple model: After a node restart, the node uptime is monotonically increasing until again being reset to zero on the arrival of the next restart.

We define $u(k)$ as the node uptime that has been transmitted within the payload $d(k)$ of a health packet k. Algorithm 2.3 lists the test used for counting sequence violations. It is important to notice, that detecting node restarts is a hard problem in the presence of arbitrary packet loss. While one would normally detect a node restart when receiving a node uptime of a defined minimal value, this is not possible if exactly that packet got lost. It is not possible to safely detect all restarts without extra information, but we use two mechanisms that make the algorithm more robust to packet loss. Firstly, we allow a certain number of health packets that were generated immediately after a node restart to be lost by introducing the parameter u_{reset}. For instance, $u_{\text{reset}} := 6 \cdot T$ allows approximately the first six health packets after a node restart to be lost.

Secondly, we try to distinguish between wrongly inserted packets and a discontinuation of the current measurement due to large holes in the data. We group measurements as long as the signal is monotonically increasing or explained by a node restart (Algorithm 2.3, line 4). On the occurrence of a discontinuity, the size of the current set is added to the number of sequence violations if the size of the set is smaller than c_{min} (line 6). Here, we assume that only short packet sequences are wrongly inserted into the packet stream while long sequences are evidence for large holes.

2.6.4 Packet Acceptance Rate and Correctness of Obtained Sequence

The result of filtering the data set with the model-based approach matches with our expectations: 25.2% of the data from the first deployment phase of non-conforming system operation are discarded. In contrast, 96.6% and 99.0% of packets being generated in the last two deployment phases are accepted. The packet acceptance rates of the model-based approach and the simple heuristic are comparable, see Table 2.4.

COUNTER	A) JUL 08-NOV 08	B) NOV 08-AUG 09	C) SEP 09-MAY 10
	II) Model-based Approach		
Accepted packets	739,319 (74.8%)	2,245,880 (96.6%)	2,958,139 (99.0%)
Sequence viol. (Alg. 2.3)	1 (0.0%)	12 (0.0%)	0 (0.0%)
Sequence viol. (Ext. storage)	*n/a*	1 (0.0%)	0 (0.0%)
	III) Simple Heuristic		
Accepted packets	726,297 (73.5%)	2,227,799 (95.8%)	2,958,192 (99.0%)
Sequence viol. (Alg. 2.3)	5,283 (0.7%)	11 (0.0%)	1 (0.0%)
Sequence viol. (Ext. storage)	*n/a*	0 (0.0%)	11 (0.0%)
	I) Unfiltered Data Set		
Total packets	988,062 (100.0%)	2,325,168 (100.0%)	2,987,901 (100.0%)
Sequence viol. (Alg. 2.3)	131,629 (13.3%)	66,159 (2.8%)	29,133 (1.0%)
Sequence viol. (Ext. storage)	*n/a*	69,025 (3.0%)	91,866 (3.1%)

Tab. 2.4 Both approaches achieve comparable, high packet acceptance rates on good data. Only the model-based approach is able to deliver correct packet sequences in all three scenarios of different system behaviors. While both approaches achieve comparable results in the number of accepted packets, an incorrect packet sequence is retrieved when applying the simple heuristic to data from the first deployment phase of non-conforming system operation.

As we can apply this validation method to the whole packet stream, we firstly start evaluating the correctness of obtained packet sequences based on Algorithm 2.3 ($u_{\text{reset}} := 6 \cdot T$, $c_{\text{min}} := 10$). For the unfiltered data set, the sequence under test is retrieved by ordering all packets by the ascending estimated packet generation time $\tilde{t}_g(k)$. The non-conforming system behavior during the first deployment phase is again confirmed by 13.3% packets being marked as invalid due to a sequence violation. In opposite, only 2.8% and 1.0% are marked as invalid when analyzing the last two deployment phases.

We are now comparing the number of sequence violations after processing the data set. After applying the simple heuristic, the packet sequence under test is again obtained by sorting accepted packets by the ascending estimated packet generation time $\tilde{t}_g(k)$. In opposite, the new property $id_N(k)$ is used to sort the output of the model-based approach. Both filter algorithms produce equal results when being applied to data of the last two deployment phases. After neglecting 11 and 12 errors that might be accounted to non-detected node restarts, both algorithms deliver a correct packet sequence.

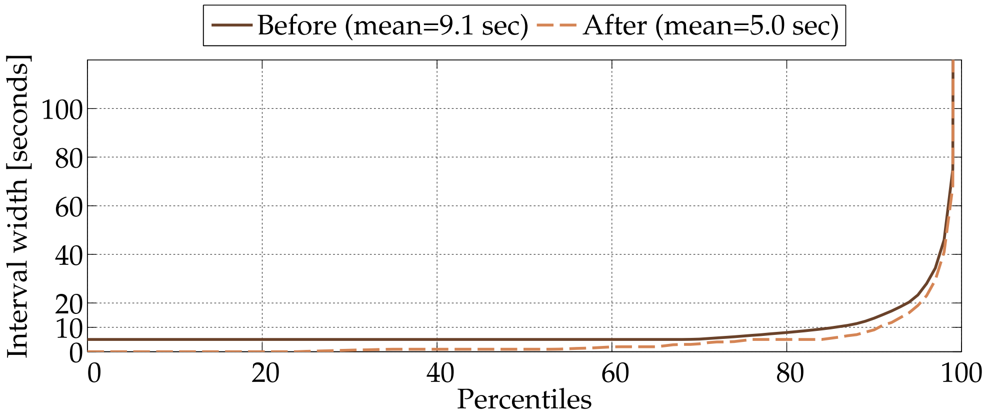
The situation is different for data from the first deployment phase: While 5,283 sequence violations remain after applying the simple heuristic, only one sequence violation can be found in the result of the model-based filter. While both algorithms perform well on good data, only the model-based approach is capable of safely removing erroneous data.

We can make a stronger argument by also including results from validating packet sequences against ground truth from recovered external storage. The observations made from this method generally match with the results from Algorithm 2.3, and also underline our claim that the model-based approach outputs a correct packet sequence. Concretely, comparing the output of the model-based approach with the packet sequence from external storage results in a single sequence violation out of more than five million packets. More detailed results on the performance of the model-based filtering approach are shown in Table 2.5.

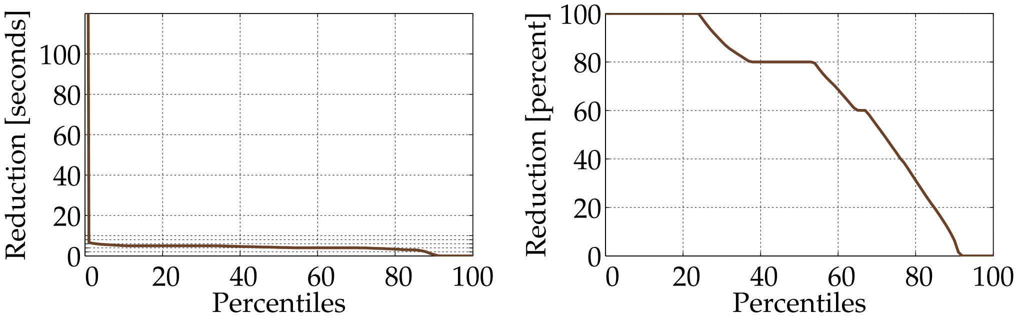
COUNTER	A) JUL 08-NOV 08	B) NOV 08-AUG 09	C) SEP 09-MAY 10
Accepted packets	739,319 (74.8%)	2,245,880 (96.6%)	2,958,139 (99.0%)
Discarded packets	248,743 (25.2%)	79,288 (3.4%)	29,762 (1.0%)
Packet duplicates	3,373 (0.3%)	69,974 (3.0%)	27,909 (0.9%)
$t_s(k) > t_s^{\max}$	0 (0.0%)	0 (0.0%)	0 (0.0%)
Failed epoch assignment	101,246 (10.2%)	7,988 (0.3%)	1,341 (0.0%)
Invalid interval $t_g^{u,l}(k)$	144,124 (14.6%)	1,326 (0.1%)	512 (0.0%)
Total packets	988,062 (100.0%)	2,325,168 (100.0%)	2,987,901 (100.0%)

Tab. 2.5 Results of model-based approach. The achieved packet acceptance rates clearly separate the first deployment phase of non-conforming system operation from the following two phases of good operation. The distribution of discarded packets to categories is also different when comparing the results of the second deployment phase with the results of the third deployment phase. While nodes are resending recovered queue contents after a restart, the amount of packet duplicates is increasing with the number of node restarts. The latter is significantly higher in the second deployment phase.

Before evaluating the performance of forward and backward reasoning in the next section, we can conclude that both the model-based approach and the simple heuristic deliver very good results when being applied to data of the last two deployment phases. However, only the model-based approach is also able to return a correct packet sequence regarding data from the first deployment phase. Thus, the model-based approach was successfully applied to all three initially mentioned scenarios of different system behaviors: A) Highly non-conforming system behavior, B) sensor nodes subject to a high frequency of unplanned warm restarts, and C) more than one third of the collected data experiencing transmission delays of several hours to days.



(a) Generation time interval widths



(b) Absolute improvement

(c) Relative improvement

Fig. 2.6 Improvement of generation time interval width $t_g^u(k) - t_g^l(k)$. The analysis covers 5,204,019 packets of the last two deployment phases. The initial generation time interval width could be shortened in 91% of the cases. The distributions of generation time interval widths and achieved improvements by applying forward and backward reasoning are given in percentiles. For instance, the 70th percentile in Figure 2.6(a) corresponds to 70% of the packets having an interval width of at most 5 seconds before applying forward and backward reasoning. The initial interval width was reduced by at least 4 seconds or 50% of the initial value for 70% of the packets.

2.6.5 Packet Generation Time Intervals

At the beginning of the data analysis, generation time intervals $[t_g^l(k), t_g^u(k)]$ were initially set by only including information from each single packet. In Section 2.5.4, we presented how these initially set intervals can be improved by also including information of temporally adjacent packets. This second step can not be applied before the correct packet sequence is known. Since data from the first deployment phase does not conform to our formulated model, further processing is only evaluated for packets of the last two deployment phases B) and C).

Applying forward and backward reasoning leads to tighter generation time intervals $t_g^u(k) - t_g^l(k)$ for 91% of the packets. The initial generation time interval width is at least reduced by half in 70% of all cases. This corresponds to an absolute reduction of up to 710 seconds. The mean generation time interval width of all processed packets is significantly decreased by a factor of almost two from 9.1 seconds to 5.0 seconds.

Initial and improved generation time interval widths are shown in Figure 2.6(a). Approximately 70% of all packets have an initial generation time interval width of five seconds. Concerning communication over up to four hops with an uncertainty of one second per hop, we must account four seconds of the initial width to the finite resolution of a sensor node clock. In opposite, the initial generation time interval width is definitely dominated by measuring the network sojourn time $t_s(k)$ under clock drift in at least 15% of the cases.

Absolute and relative improvement by applying forward and backward reasoning are shown in Figure 2.6(b) and Figure 2.6(c), respectively. The initial generation time interval is reduced by up to 710 seconds on the absolute scale. On the relative scale, up to 99% of the initial width are subtracted. Large improvements on the absolute scale can only be achieved for packets having a large initial interval width. In contrast, large improvements on the relative scale are in the majority of the cases achieved for packets whose initial generation time interval width is dominated by the uncertainty caused by a finite clock resolution. Initial interval widths remain unchanged for 9% of the packets.

Concluding, significant reductions of the initial generation time intervals could be achieved for a considerable amount of packets. The last processing step of forward and backward reasoning does not only compensate introduced worst-case uncertainties, but also leads to considerable improvements in general.

2.7 Broader Applicability and Limitations

Key assumptions of our formal model are data acquisition at constant rate, the existence of unique packet source addresses, the existence of packet sequence numbers, and the existence of provisions for estimating the generation time of a packet. Hard limits are formally given by Theorems 2.1 and 2.2 which must be satisfied for being able to safely assign packets to epochs.

There is no strict requirement on the accuracy of generation time estimates; the required accuracy can only be seen in the context of a full system model parameter set. While we consider a system design that also supports disconnected operation over long periods and thus resorts to a simple packet generation time estimation, the presented data analysis algorithms for duplicate filtering and epoch assignment are also suited for taking time information retrieved from other mechanisms, *i.e.*, FTSP [MKSL04b], as input. The presented work does not compete with work on clock synchronization, rather, it can be used to enhance data quality also in systems that already offer precise packet generation time information.

Additionally, we also want to stress that there are no strict requirements concerning the data collection protocol used. Regarding the retrieval of packet generation time information as an orthogonal problem that is addressed by another layer, we can currently see no limitations when considering the use of other comparable data collection systems.

Sampling data at a constant rate is a valid scenario in the context of environmental monitoring. Prominent examples are the monitoring of the microclimate of a coastal redwood tree [TPS+05] or environmental monitoring under extreme conditions [BIS+08]. Furthermore, glacier monitoring [MOH05] is an application that does not only sample data at a constant rate, but also allows sensor nodes to be unable to communicate for several days or even weeks. In spite of recent advancements in protocols and platforms, we consider sampling at constant rate as a valid scenario for many current and future environmental monitoring applications.

Extending this work to systems that generate multiple packets, both periodically and sporadically, within a constant sampling period T requires the availability of an additional sampling round counter that is transmitted as part of every periodically generated packet. While the packet sequence number $s(k)$ is equal to such a sampling round counter in the model presented, the packet sequence number is no longer a valid proxy for the sampling round when more than one packet is generated within a sampling period.

2.8 Conclusions

The proposed model-based approach is a viable method for reconstructing the correct sequence of packet generation and validating data integrity at the sink. Only a single violation (see Figure 2.7 and Table 2.4) is found when cross-validating a sequence of more than 5.2 million packets with ground truth from external storage. Forward and backward reasoning clearly tightens packet generation time bounds by employing information of temporally adjacent packets. Overall, we retrieved convincing results for all three evaluated scenarios of A) highly non-conforming system behavior, B) sensor nodes subject to a high frequency of unplanned warm restarts, and C) more than one third of the collected data experiencing transmission delays of several hours to days. We compared the model-based approach to a simple heuristic: Only the model-based approach was able to return correct packet sequences in all three scenarios.

Our approach is not only useful for cleaning historical data. It also enables to learn about the limits of a system design. First, our formal model clearly shows the limitations of certain parameter sets. For instance, it becomes obvious that the length and the management of the packet sequence number are important parameters for removing uncertainties when reconstructing the packet sequence. Second, our new approach for offline data processing also allows to shift complexity

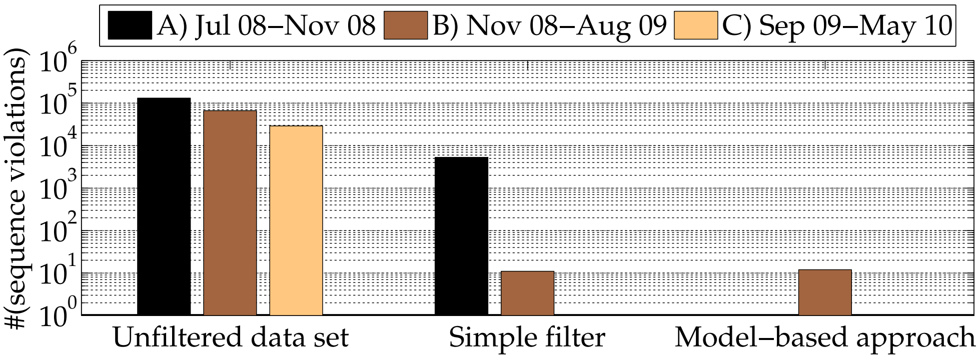


Fig. 2.7 While both approaches succeed on data from the last two deployment phases of good system operation, only the model-based approach can deliver basically no sequence violations when applied to the first deployment phase of non-conforming system operation (Algorithm 2.3).

from resource-scarce sensor nodes to powerful computation devices in the backend. Depending on the deployment scenario, services such as clock synchronization can be intentionally left out without sacrificing data quality.

Multi-hop network tomography (Chapter 4) and hybrid monitoring (Chapter 5) are two exemplary methods that require data to be free of duplicates as well as the order of packet generation to be known.

3

Visualization of Large Data Sets in Space and Time with Vizzly

After data has been cleaned from artifacts that have been introduced by the wireless sensing system (see previous Chapter 2), it is ready for further use. Even though many applications ask for quantitative analysis methods, *e.g.*, statistical analysis, the manual inspection of sensor data is an at least equally important use case. Apart from the creation of all sorts of system supervision and real-time dashboards, being able to visualize data is also an important asset during the development and debugging of data processing algorithms.

With sensing systems reaching maturity and scale, data visualization is no longer only a problem of representation, but also requires algorithmic problems to be addressed. First of all data repositories in total now embracing billions of data points, individual time series are likewise reaching sizes that can no longer be simply displayed. For instance, a single temperature sensor that is sampled every two minutes will already generate one million data points in less than four years. Challenges are first of all to quickly locate data in such large repositories, and secondly to aggregate time series on-the-fly so that they can be displayed on arbitrary devices.

3.1 Introduction

This chapter presents Vizzly, a middleware that enables the interactive browsing of large sensor network data sets that originate from static and mobile scenarios. The back-end infrastructure of Vizzly consists of two main components, namely a cache layer and a web service. The web service component of Vizzly provides a single point of entry for retrieving sensor data that may originate from multiple data repositories. Based on the received request parameters, Vizzly automatically chooses the temporal and spatial levels of detail in which data is sent to a user. The cache layer is significantly reducing data access times. The tasks of the cache layer are (i) to continuously monitor and read from multiple repositories of different kind, *e.g.*, SQL databases, feeds, or CSV files, (ii) to aggregate the received data, and (iii) to store pre-computed results in data structures that are optimized for time-based access.

The Vizzly front-end library enables the integration of interactive map and line plot widgets into existing web pages. A user must only specify the position and size of a new widget, the setup of required displaying components and visual control elements is automatically handled by provided library functions. Additionally, the Vizzly client library handles all client-server communication, *e.g.*, decides when new data must be dynamically loaded.

The contribution of this chapter is as follows:

- We present Vizzly, a middleware that enables the interactive browsing of large sensor network data sets. The presented client-server architecture focuses on the problem of adapting to each user by automatically choosing the temporal and spatial resolution of returned data. For example, highly aggregated data is loaded when a user wants to quickly navigate through multiple years of data. In contrast, raw data points are shown when the selection is narrowed down to a particular point in space and time.

- The effectiveness of our approach is evaluated in the context of two distinct static and mobile sensing scenarios.

- We evaluate the performance of Vizzly based on the instance used in a production environment. The analyzed instance currently handles more than four billion data points that originate from 2,900 different sensing channels.

The structure of this chapter is as follows: Related work and the positioning of Vizzly are discussed in Section 3.2. Section 3.3 presents the challenges found in the visualization of large sensor network data sets. The system design of Vizzly is described in Section 3.4, implementation details of Vizzly can be found in Section 3.5. Section 3.6 presents two case studies that prove the applicability of Vizzly and highlight the conceptual advantages of our approach. The performance of Vizzly is evaluated in Section 3.7. The broader applicability of Vizzly is discussed in Section 3.8. Section 3.9 concludes this chapter.

3.2 Related Work

A. Sensor Data Visualization

A web interface for displaying sensed data is part of many sensing projects. For example, GlacsWeb[1] [MOH05] and LoCal[2] [DHLT+12] provide an interface for visualizing sensed data on a timeline. Based on Microsoft SensorMap, the interface of Life Under Your Feet[3] [TMEC+10] also allows to display the locations of static sensors on a 2D map. A similar solution is provided by Climaps[4], a data visualization interface that is part of the SensorScope [BIS+08] project. PowerTron [KGL10] has been developed for the visualization of power meter data in the PowerNet [KHLK09] project. Here, the locations of available power meters are shown on the floor plans of a building. Pachube/Cosm/Xively is a very popular online platform for sensor data streaming. An exemplary project based on Pachube/Cosm/Xively is the Japan Geigermap[5]. Apart from displaying the raw measurements from static and mobile sensors, *da_sense*[6] [SBS+11] also includes a heat map representation of noise pollution measurements. Within a more general scope, Google Fusion Tables[7] is a web service for visualizing data as maps, timelines and charts.

Vizzly is a middleware for visualizing large sensor network data sets in space and time. Both static and mobile sensing scenarios are supported, measurements of arbitrary length can be visualized in any temporal and spatial level of detail. Independent of a particular front-end component used for eventually displaying loaded data, Vizzly focuses on the problem of making sensor data dynamically and efficiently loadable when request parameters change. For example, less aggregated data is automatically loaded and displayed when the length of the time period of interest is decreased.

From a survey of existing approaches based on publicly available information, *e.g.*, manual analysis of the client-server communication of public data interfaces and consulting of available documentation, we find that Vizzly distinguishes itself from other solutions by dynamically loading data of varying detail. While similar techniques can be found in map visualizations, *e.g.*, in the *da_sense* [SBS+11] project, we could not find other time series visualizations that would allow to display multi-year measurements in any temporal level of detail. We find existing solutions to either limit the presentable time range, or to only offer data on a fixed, reduced level of detail, *i.e.*, data is only loaded once after the initial request, but not refined when the user selects a smaller area of

---

[1] http://env.ecs.soton.ac.uk/glacsweb/iceland/graph/

[2] http://new.openbms.org/

[3] http://dracula.cs.jhu.edu/luyf/en/tools/VZTool/Default.aspx

[4] http://climaps.com/

[5] http://japan.failedrobot.com/

[6] http://www.da-sense.de/

[7] http://www.google.com/fusiontables

interest.

B. Processing and Storage of Time Series Data

The creation of materialized views [GM95] is a common technique for continuously pre-computing and storing aggregates in database systems. Data cubes [GCB+97] allow for the efficient computation of multi-dimensional aggregates. For example, the Life Under Your Feet Project uses data cubes for aggregating data over time intervals and other quantities. RasDaMan [BDF+99] is a database system that is optimized for the storage of multidimensional raster data.

Network monitoring tools, *e.g.*, Zabbix[8] and ntop, often also include the ability to visualize measured performance data. In this context, *tsdb* [DMF12] implements a compressed database for time series. Using a special storage scheme, massive data volumes can be efficiently handled and made available to a user, *e.g.*, for plotting. Probably the closest to this work is the function of the Archiver Daemon (ARD) that is part of *sMAP*[9] [DHJT+10]. The storage layer used by the ARD is highly optimized for the processing and storage of time series data.

sMAP, *tsdb* and Vizzly share the design decision of implementing data processing in an application layer and to use the underlying database system as a key-value store only. Instead of heavily relying on the support of certain features, *e.g.*, data cube analysis, flexibility is gained when basically any database system can be used. While *sMAP* and *tsdb* can potentially also be extended to support this, Vizzly is in particular designed for data originating from both static and mobile sensors, *i.e.*, sensor readings that are annotated with both time and location information.

3.3 Visualizing Large Sensor Network Data Sets

Being able to visually inspect recorded data is advantageous for all stakeholders of a sensing system, *e.g.*, design engineers, system operators, scientists, and even the general public. For small data sets that consist of some hundreds of data points, a simple approach is to just send raw data points to a client for plotting. However, this approach does not scale for large multi-year data sets that consist of millions of data points. The amount of data to be transferred to a user becomes too large, the client would need to perform extensive computation, *e.g.*, sorting and down-sampling, for making the data displayable. Instead, data must already be filtered, sorted and aggregated before it is transferred to a client that only displays the already prepared data.

Needed processing steps for making raw data displayable can be very expensive, *e.g.*, involve large database tables to be fully read and

---

[8] http://www.zabbix.com/

[9] http://code.google.com/p/smap-data/

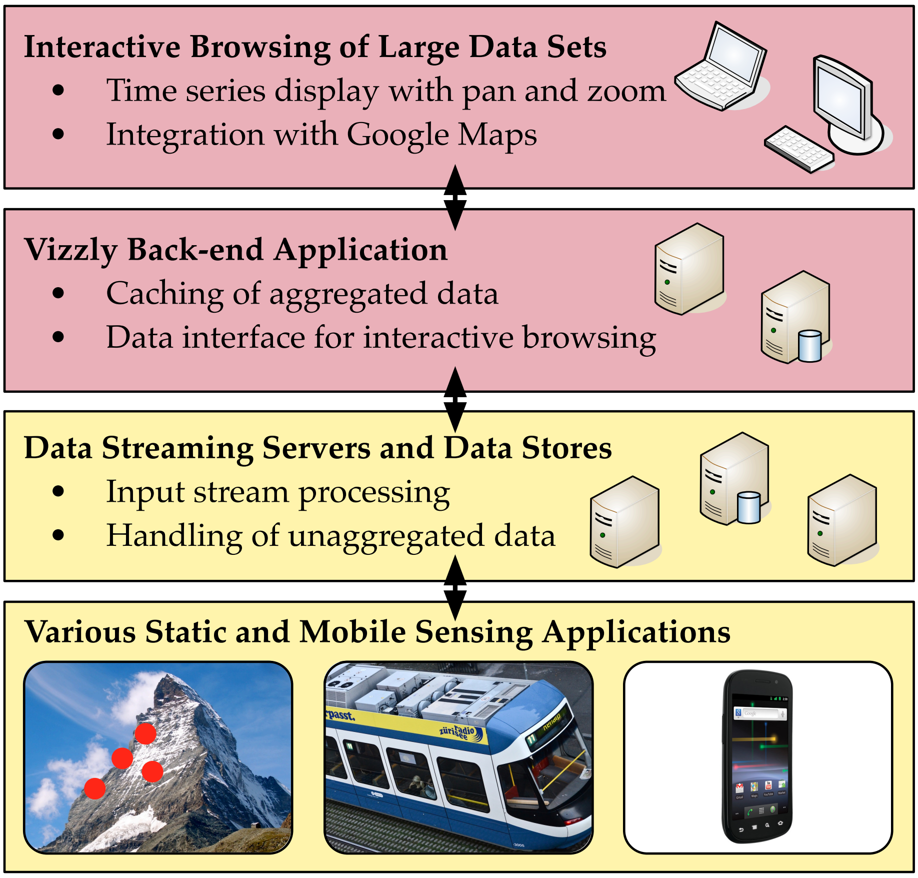

Interactive Browsing of Large Data Sets
Time series display with pan and zoom
Integration with Google Maps
Vizzly Back-end Application
Caching of aggregated data
Data interface for interactive browsing
Data Streaming Servers and Data Stores
Input stream processing
Handling of unaggregated data
Various Static and Mobile Sensing Applications

Fig. 3.1 Exemplary usage scenario for Vizzly. Without the need of modifying existing infrastructure (yellow boxes), powerful plotting capabilities are added by Vizzly (red boxes).

sorted. The response time of such a request can easily reach tens of seconds if the underlying system has not been specifically optimized. This is problematic as the adoption and success of a visualization tool are threatened when interested users perceive that the front-end responds too slow [BBK00].

In the context of visualizing large sensor network data sets, this work wants to address the following questions: Which data structures allow for the efficient retrieval of spatiotemporal, structured data at different scales? Which system design is suited for as many application scenarios as possible while also satisfying the needs of all user groups? What can we learn from standard PC memory architectures when designing a cache application that can choose to either store pre-aggregated data in different back-ends, *i.e.*, in memory (RAM) or in a database, or to further aggregate already pre-aggregated data on-the-fly?

In the following, we present the design and implementation of Vizzly, a middleware for the interactive browsing of large sensor network data sets. The applicability of Vizzly is evaluated in the context of two diverse research projects, the performance of Vizzly is evaluated based on the traces that originate from a production environment.

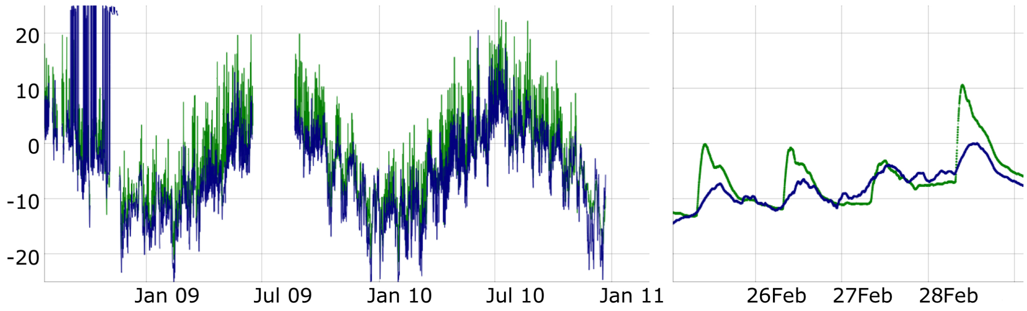


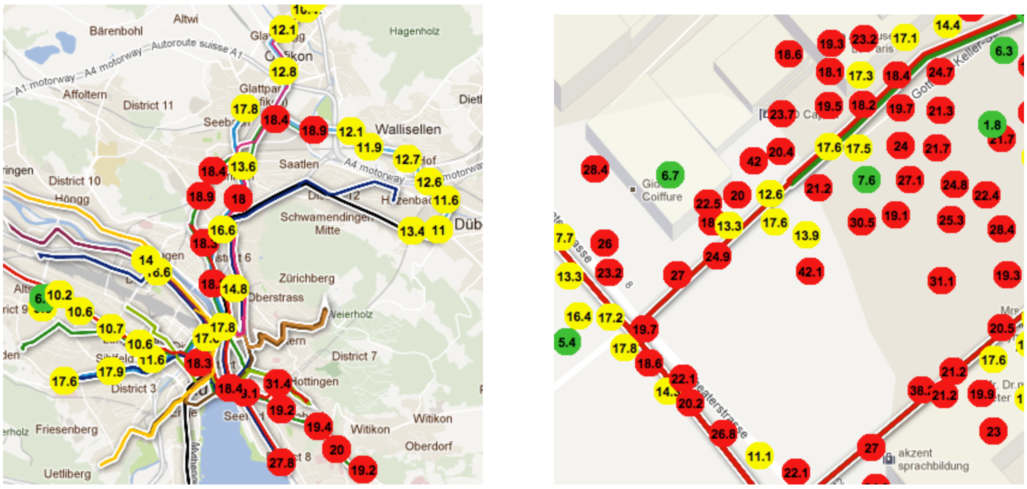
Fig. 3.2 Temperature readings from two sensor nodes. The temporal resolution must be highly reduced for displaying multiple years in a single view. Less aggregated or even raw readings are shown when the time period of interest is lessened.

3.4 Vizzly System Design

An exemplary usage scenario for Vizzly is shown in Figure 3.1. Starting from the bottom, structured data is recorded in various static and mobile sensing applications. Recorded data is then uploaded to a streaming server that annotates, *e.g.*, adds meta-information, and stores the data received. Vizzly continuously monitors and reads from multiple streaming servers, cached aggregates are immediately updated when new data arrives. The Vizzly back-end can handle multiple clients in parallel, requests are either served from pre-computed aggregates, from an on-the-fly aggregation of already aggregated data, or by forwarding the results of raw data requests. The latter are always served by the lower layer, *i.e.*, the corresponding streaming server. On the client side, the Vizzly front-end library implements map data and time series displays. Exemplary screenshots are shown in Figure 3.2 and Figure 3.3, respectively. The time series display can be used standalone, but is also part of the map data display. Only measurements taken within the corresponding map area are shown in a line plot when a user selects a map marker.

Decoupling data visualization from data storage certainly increases the overall system complexity, *e.g.*, requires data to be synchronized between multiple systems. However, adding missing functionality to an existing data store is often risky or even impossible. For instance, adding visualization capabilities may decrease the performance of another important use case. Organizational issues, *e.g.*, license restrictions, may not allow for adding modifications. Vizzly is an optimized and centralized solution that enables the visualization of sensor data. Vizzly is not restricted to a certain system, but can potentially be integrated into many existing platforms (see Sec. 3.8).

Detached from the concrete implementation of Vizzly (see Sec. 3.5), we will now firstly present the overall system design of Vizzly. The interfacing between front-end and back-end is described in Section 3.4.1. Section 3.4.2 presents the mechanism used for automatically determining the temporal and spatial resolutions in which data is sent to a user. The strategy used for aggregating spatiotemporal data in space and time is



(a) 9.6 km x 9.6 km

(b) 148 m x 148 m

Fig. 3.3 Ozone data shown at two different spatial scales. Both excerpts correspond to the same measurement period, more detail is shown when the selected map area gets smaller. A user can change the data source of interest, *e.g.*, select CO measurements instead of ozone measurements, the time period of interest, and the map area of interest. After each interaction, missing data is automatically loaded from the Vizzly back-end. Shown data points are within the accuracy of the GPS receiver used.

presented in Section 3.4.3. Section 3.4.4 presents data structures used for caching and persisting aggregated data. Vizzly is not automatically informed of new sensor data, but actively polls known repositories. The automated updating of cached contents is discussed in Section 3.4.5.

3.4.1 Client-Server Communication

Clients neither store sensor data nor any system state, *e.g.*, a list of available sensors. Instead, all information is dynamically loaded from the Vizzly back-end. Apart from necessary standard request parameters, *e.g.*, the time period of interest, choosing from large collections of sensors requires a precise but also flexible specification format. While mix-ups between similarly named sensors in different repositories need to be avoided, participatory sensing scenarios on the other hand require that sensor readings can be selected both dependent and independent of the recording device, *e.g.*, the particular smartphone used.

To address this problem, we introduce the notion of “virtual signals”. In the context of Vizzly, a virtual signal can be described as a tuple (N, C, R) that consists of a sensor node N, a sensing channel C, and a data repository R. Selecting a particular sensor node can be omitted by setting N to a wildcard value. If the wildcard value is set, data is automatically combined on the equal sensing channel C and the equal data repository R[10]. Exemplary virtual signals are *(node 25, temperature, repository 1)* and

---

[10] Please note that Vizzly expects underlying time series to be normalized with respect to possibly different sensor types or sensor calibrations used.

(node ANY, ADC channel 3, repository 2). Exactly one virtual signal must be specified in a *map data* request, arbitrary combinations of one or more virtual signals can be specified in a *time series* request.

3.4.2 Algorithmic Selection of the Returned Level of Detail

The size of the screen area for displaying map and line plots varies among different clients. In consequence, the optimal level of temporal and spatial detail in which data is to be sent to a client also varies. For example, the doubled number of data points can be displayed when a user views a map in full-screen mode instead of restricting the map to only one half of the screen. The lack of dynamically adapted levels of detail would lead to either unused space on large screens, or to overlapping data points on small screens. To overcome this, the Vizzly back-end first automatically decides if unaggregated data can be displayed or, if not, calculates suitable temporal and spatial target resolutions $\hat{r}_{\text{temp}}$ and $\hat{r}_{\text{spat}}$. Target resolutions $\hat{r}_{\text{temp}}$ and $\hat{r}_{\text{spat}}$, respectively, then define the window length of the aggregation operator used.

For this mechanism to work, clients are requested to specify the dimensions of the displaying widget used. Data included in the response of a *map data* request is always reduced to a dynamically computed $\hat{r}_{\text{spat}}$. Decided separately for each of multiple included time series, the response of a *time series* request can contain both unaggregated data and values that have been aggregated to a dynamically chosen $\hat{r}_{\text{temp}}$.

Based on a defined size of a grid cell in pixels, determining $\hat{r}_{\text{spat}}$ starts with calculating the number of displayable grid rows and columns. The geographical dimensions of a quadratic grid cell are then derived from the map area of interest that is specified in geographic coordinates. The optimal spatial target resolution $\hat{r}_{\text{spat}}$ corresponds to the geographic length of the edges of a grid cell, *e.g.*, $\hat{r}_{\text{spat}} := 1\,\text{km}$.

Determining the temporal level of detail starts with calculating the maximum number of data points per time series that the client can display. This number is obtained by dividing the reported width of the plotting canvas by a defined maximum average number of data points per pixel. With the help of continuously updated estimates of the sampling rate of each virtual signal (see Sec. 3.4.5), Vizzly first decides for each virtual signal if the client can be supplied with unaggregated data. To decide this, the estimated number of unaggregated data points is compared with the maximum displayable number of data points. If the level of temporal detail must be reduced, the optimal temporal target resolution $\hat{r}_{\text{temp}}$, *e.g.*, $\hat{r}_{\text{temp}} := 1\,\text{hour}$, is determined by dividing the time period of interest by the maximum number of data points per time series.

For being able to further aggregate already aggregated data, target resolutions $\hat{r}_{\text{spat}}$ and $\hat{r}_{\text{temp}}$ can not be arbitrarily chosen, but need to be multiples of defined highest spatial and temporal target resolutions.

Intermediate resolution levels $\hat{r}_{\text{spat}}$ and $\hat{r}_{\text{temp}}$ are "rounded" to the next available smaller target resolution.

3.4.3 Aggregation of Spatiotemporal Data

The level of spatial and temporal detail may need to be reduced before data is sent to a client. In this case, the data of higher or equal resolution is loaded from the cache and, if needed, further aggregated on-the-fly. While users can choose arbitrary combinations of temporal and spatial levels of detail, the number of possible combinations is clearly too large for each combination being stored in a cache.

To address this problem, we propose a location-preserving aggregation scheme: Temporal aggregation is carried out separately for each location, the results of the location-preserving aggregation are then cached. While this strategy certainly decreases the efficiency of data reduction, *i.e.*, leads to a smaller reduction in terms of the number of returned samples, one stored aggregate per virtual signal is sufficient for being able to choose any level of detail for any later spatial data aggregation. Additionally, this approach preserves advantages of aggregating and organizing data in the temporal domain. Compared to executing the spatial aggregation step first, this order of aggregating spatiotemporal data in space and time benefits from the structure of the data. Preserving temporal locality is much less complex than organizing data in the two-dimensional spatial domain.

Data series without location information can be described as a set of tuples (t, v). Each tuple describes a data point that consists of a timestamp t and a sensor reading v. For aggregating data in the temporal domain, the first step is to reduce the resolution of the timestamp t to the target resolution $\hat{r}_{\text{temp}}$, *e.g.*, $\hat{r}_{\text{temp}} := 10$ minutes.

$$R(t, \hat{r}_{\text{temp}}) := \left\lfloor \frac{t}{\hat{r}_{\text{temp}}} \right\rfloor \cdot \hat{r}_{\text{temp}} \tag{3.1}$$

Data points are then grouped by their truncated timestamp $t' := R(t, \hat{r}_{\text{temp}})$. Data points with an equal truncated timestamp are put into the same set $\mathcal{G}$, the aggregated sensor value v' of the new tuple (t', v') is obtained by applying the aggregation function $A(\mathcal{G})$. Exemplary aggregation functions are the calculation of the mean, the sum, the number of elements, or the smallest value.

$$v' := A(\mathcal{G}) \text{ for } \mathcal{G} := \{v \mid R(t, \hat{r}_{\text{temp}}) \equiv t'\} \tag{3.2}$$

Already aggregated data, *e.g.*, a set of (t', v') tuples, can be further aggregated, *e.g.*, to obtain (t'', v''). Given that data was reduced to a target resolution $\hat{r}_{\text{temp}}$ in the previous step, the new target resolution $\hat{r}'_{\text{temp}}$ has to be a multiple of $\hat{r}_{\text{temp}}$, *i.e.*, $\hat{r}'_{\text{temp}} := \hat{r}_{\text{temp}} \cdot c$ with $c \in \mathbb{N}, c > 0$.

For measurements from location-aware sensors, the extended tuple $(t, v, l_{\text{lat}}, l_{\text{lng}})$ also includes the geographical latitude l_{lat} and longitude l_{lng} of the measurement location.[11]

To preserve a later aggregation by the location of measurement, location information must remain untouched during temporal aggregation. Data points from location-aware sensors must therefore be grouped by their truncated timestamp t' and by their location of measurement that is specified by l_{lat} and l_{lng}. Data points with an equal truncated timestamp and equal location information are put into the same group, the aggregated sensor value v' of the new tuple $(t', v', \bar{l}_{\text{lat}}, \bar{l}_{\text{lng}})$ is similarly obtained by applying the aggregation function $A(\mathcal{G})$ to the set of data points $\mathcal{G}$.

$$\begin{aligned} v' := A(\mathcal{G}) \text{ for } \mathcal{G} := \{v \mid\ & R(t, \hat{r}_{\text{temp}}) \equiv t' \wedge \\ & l_{\text{lat}} \equiv \bar{l}_{\text{lat}} \wedge l_{\text{lng}} \equiv \bar{l}_{\text{lng}}\} \end{aligned} \tag{3.3}$$

The location of measurement for which v' is computed is defined by $\bar{l}_{\text{lat}}$ and $\bar{l}_{\text{lng}}$. In practice, $\bar{l}_{\text{lat}}$ and $\bar{l}_{\text{lng}}$ are automatically set while iterating over the list of locations.

Aggregating data in the spatial domain starts with reducing the resolution of the location of measurement. Both l_{lat} and l_{lng} are reduced to the same target resolution $\hat{r}_{\text{spat}}$.

$$R(l_{\text{lat}}, \hat{r}_{\text{spat}}) := \left\lfloor \frac{l_{\text{lat}}}{\hat{r}_{\text{spat}}} \right\rfloor \cdot \hat{r}_{\text{spat}} \qquad R(l_{\text{lng}}, \hat{r}_{\text{spat}}) := \left\lfloor \frac{l_{\text{lng}}}{\hat{r}_{\text{spat}}} \right\rfloor \cdot \hat{r}_{\text{spat}}$$

Spatiotemporal data that has been recorded within the time period of interest $[t_s, t_e]$ is grouped by its truncated location information. Based on the length of the time period of interest, Vizzly automatically chooses the level of temporal detail that is used as the input for the spatial aggregation step. Without further specifying the concrete level of detail used, the data is taken from the set of tuples $(t^{(n)}, v^{(n)})$.

$$\begin{aligned} v' := A(\mathcal{G}) \text{ for } \mathcal{G} := \{v^{(n)} \mid\ & t_s \leq t^{(n)} \leq t_e \wedge \\ R(l_{\text{lat}}, \hat{r}_{\text{spat}}) \equiv l'_{\text{lat}} & \wedge R(l_{\text{lng}}, \hat{r}_{\text{spat}}) \equiv l'_{\text{lng}}\} \end{aligned} \tag{3.4}$$

3.4.4 Efficient Storage of Pre-Computed Data

When processing a user request, Vizzly first loads all data that lies within the time period of interest. If location information is relevant, found records are then filtered to only include data from the requested map area of interest. To support this, the caching layer of Vizzly must fulfill the following three requirements. First, data structures used for storing aggregated data must be optimized for time-based access. Second,

---

[11] As the altitude of the location of measurement is not required for 2D map plots, this information is currently omitted.

although data is aggregated before it gets cached, storing aggregated data can still require considerable space for large data sets. Third, the operation of Vizzly requires certain meta-data to be stored, *e.g.*, how often cache contents were loaded, and when the last update took place.

We propose two diverse back-ends for storing aggregated data, *i.e.*, storing aggregated data in memory (RAM) and in a SQL database. While aggregated data that is stored in memory can be accessed significantly faster, the SQL database adds cheaper storage capacity and persistence. Infrequently used data can be offloaded to the slower tier, populating cold memory from the SQL database is faster than again fetching unaggregated data from its source. Since only aggregated data is cached in Vizzly, any time information used in the following description must be seen in the context of a certain temporal target resolution $\hat{r}_{\text{temp}}$.

The memory back-end uses one-dimensional arrays for storing sensor readings and location information. Timestamps are not explicitly stored, but are implicitly given by the index of an array element. The data structure used for storing time series data without location is depicted in Figure 3.4(a). Each instance of this data structure can be described by a tuple $\big((N, C, R), \hat{r}_{\text{temp}}, T_{\text{start}}\big)$ that specifies the corresponding virtual signal, the temporal resolution $\hat{r}_{\text{temp}}$ of the stored data, and the timestamp T_{start} that corresponds to the first array element.

The index of the first element is always 0, the timestamp of any array element is obtained by multiplying its index by $\hat{r}_{\text{temp}}$ and adding the result to T_{start}, *i.e.*, $T := T_{\text{start}} + i \cdot \hat{r}_{\text{temp}}$. Not only reducing the amount of stored data, this linear relationship between array indexes and timestamps allows to access data very efficiently. Each array element covers a time period of length $\hat{r}_{\text{temp}}$. Adding new data only requires to either update the last array element or to append a new element to the end of the array.

An additional lookup table, see Figure 3.4(b), is needed for storing time series data with location information. As location information is preserved during temporal data aggregation, there can be multiple aggregates referring to the same timestamp. The relationship between array indexes and timestamps is again linear in the lookup table. Each lookup table record specifies the start and end of contiguous segment in a second one-dimensional array that stores sensor readings with location information. Both arrays must be updated when new data is added. The scope is again limited to replacing at most the last array element or the last contiguous segment, respectively.

The database back-end organizes aggregated data in several database tables. The contents of each database table can be described by a reduced tuple $\big((N, C, R), \hat{r}_{\text{temp}}\big)$ that specifies the source and the temporal resolution $\hat{r}_{\text{temp}}$ of the table contents. Aggregated data without location information is stored in database tables consisting of two columns, *i.e.*, timestamp and aggregated sensor value. Four columns are necessary for storing aggregated data with location. An index on the time column is used for

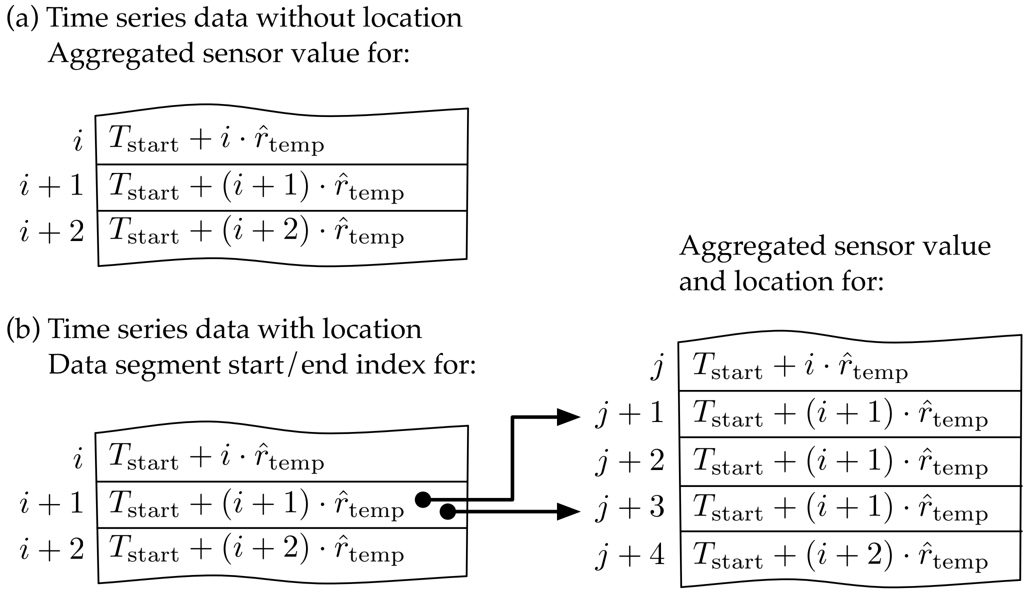

(a) Time series data without location
Aggregated sensor value for:
i $T_{\text{start}} + i \cdot \hat{r}_{\text{temp}}$
$i+1$ $T_{\text{start}} + (i+1) \cdot \hat{r}_{\text{temp}}$
$i+2$ $T_{\text{start}} + (i+2) \cdot \hat{r}_{\text{temp}}$
(b) Time series data with location
Data segment start/end index for:
i $T_{\text{start}} + i \cdot \hat{r}_{\text{temp}}$
$i+1$ $T_{\text{start}} + (i+1) \cdot \hat{r}_{\text{temp}}$
$i+2$ $T_{\text{start}} + (i+2) \cdot \hat{r}_{\text{temp}}$
Aggregated sensor value and location for:
j $T_{\text{start}} + i \cdot \hat{r}_{\text{temp}}$
$j+1$ $T_{\text{start}} + (i+1) \cdot \hat{r}_{\text{temp}}$
$j+2$ $T_{\text{start}} + (i+1) \cdot \hat{r}_{\text{temp}}$
$j+3$ $T_{\text{start}} + (i+1) \cdot \hat{r}_{\text{temp}}$
$j+4$ $T_{\text{start}} + (i+2) \cdot \hat{r}_{\text{temp}}$

Fig. 3.4 Data structures used for storing aggregated data in memory. T_{start} is the timestamp of the first array element. Location-preserving temporal aggregation can yield multiple aggregates for the same time but distinct locations.

faster accessing data based on time information.

3.4.5 Continuous Maintenance of Cached Contents

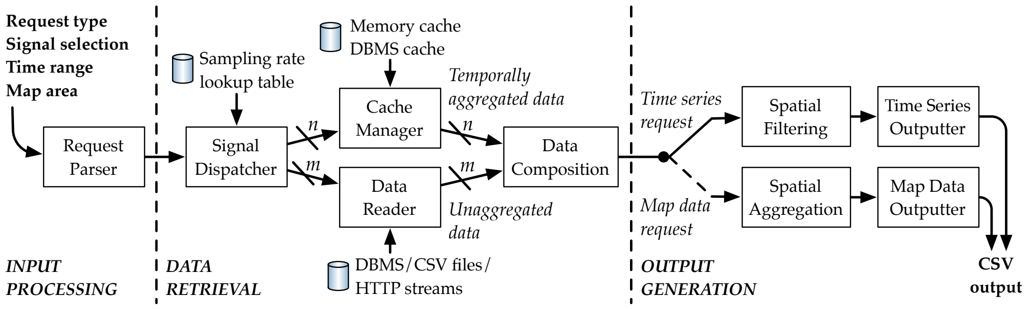
After Vizzly learned about the existence of a certain virtual signal, *e.g.*, from a user requesting certain data, it starts fetching the complete history of this virtual signal from the respective repository. Received data is then aggregated, the results of this aggregation step are stored within Vizzly.

Furthermore, unaggregated data is also used for estimating the sampling rate of sensors used. Vizzly needs this information for deciding if a user request can be served with unaggregated data, or if the response would contain too many entries (see Sec. 3.4.2). A simple solution for estimating the sampling rate is to divide the number of raw samples by the measurement duration. For achieving more robustness against configuration changes that might occur over time, we currently use a window-based approach that maintains monthly sampling rate estimates.

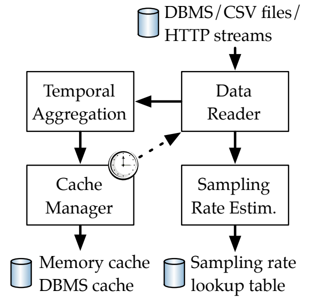
Both aggregated data and sampling rate estimates get outdated when new data is being added to the source repository. Vizzly continuously polls respective repositories and, if necessary, updates cached contents.

3.5 Vizzly Implementation

The Vizzly software package consists of a Java web application and a JavaScript library. After the back-end has been setup once, adding interactive map and line plots to existing web pages is very easy. Setting



(a) Processing of a data request



(b) Background operation

Figure 3.5 The returned temporal level of detail is decided separately for each requested virtual signal. After all data has been collected and put together, generating the final response may require further aggregation or filtering in the spatial domain. Sampling rate estimations and cached contents used for generating CSV outputs are continuously updated by a number of concurrently running background threads.

up a new plot only requires to specify the virtual signals of interest and an empty placeholder object on the web page itself. Size and position of the placeholder object can be freely chosen in the HTML markup of the web page, a fully functional visualization widget is then automatically created by the Vizzly front-end library. The Vizzly front-end library itself integrates *dygraphs*, a line plot library, the *Google Maps JavaScript API*, and the *jQuery UI* user interface library. Event-based communication with the back-end is established using *XMLHttpRequest*, loading new data is triggered by several user interactions, *e.g.*, a change of the time period of interest. Request details are specified in the JSON format, requested data is returned in the CSV (comma-separated values) format.

All back-end functionality is provided by a Java web application that runs within *Jetty*, a light-weight HTTP server. Unaggregated data is fetched from several instances of Global Sensor Networks [AHS06] (GSN). GSN is a middleware for sensor networks that allows to access its data streams over HTTP. Multiple background threads are concurrently updating cached contents, aggregated data is stored in memory (RAM) and in a MySQL database. *DBCP* is used for pooling database connections, *i.e.*, subsequent database accesses are accelerated by connection re-usage. By applying a *gzip* compression filter before sending a response to a client, the amount of transferred data is significantly reduced up to a factor of five and more.

Apart from offering a data access interface, the Vizzly back-end also provides a web-based management console and a web-based performance probe. The management console allows to see which objects are currently stored in the cache, a user can request single cache contents to be removed. The web-based performance probe exposes certain performance indicators, *e.g.*, the current cache size. To support system supervision, this information is periodically sampled by a network monitoring system.

Integral components of the Vizzly back-end and their interplay are shown in Figure 3.5(a) and Figure 3.5(b). The organization of cached contents is encapsulated by the *Cache Manager* component that only exposes an interface for retrieving aggregated sensor data. The *Cache Manager* maintains an internal list of known virtual signals, new virtual signals are learned when a user requests a yet unknown virtual signal. The list of virtual signals is continuously traversed, the *Cache Manager* can decide to update related contents, to remove contents from the cache, or to move contents between different available cache back-ends.

3.6 Two Diverse Use Cases

The requirements for the design of Vizzly originate from the PermaSense [HTB+08, BBF+11] and OpenSense [ASC+10] research projects. For being able to analyze both long-term and short-term dynamics of monitored processes, the PermaSense project requires data to be visualizable on arbitrary time scales. Focusing on mobile sensors, *e.g.*, sensing systems mounted on vehicles, the OpenSense project requires data to also be visualizable at varying spatial scales. To date, more than 1 billion packets have been collected in the PermaSense and X-Sense projects. Packets are carrying multiple sensor values, the number of collected sensor samples is larger than 15.9 billion. Likewise, more than 259 million packets that have been collected in the OpenSense project correspond to more than 5.2 billion samples. The growth of the data set used over the past five years is shown in Figure 3.6. Large amounts of data have been added in the previous two years with mostly newly deployed GPS sensors and air quality monitoring devices being responsible for this growth.

3.6.1 PermaSense: Dynamics of Varying Temporal Horizons

The PermaSense project strives for the observation of geophysical phenomena in high-altitude regions. Since the initial deployment in 2008, we currently operate four long-term sensor network deployments [KWL+11]. More than 110 deployed low-power sensor nodes fulfill different monitoring tasks, *e.g.*, the monitoring of ground temperatures and the monitoring of ice clefts, data is typically sampled every two minutes. New generation sensing devices also monitor deformation processes within rock walls, the movement of rock glaciers is monitored using around 35 online and offline GPS devices [WLB+11].

Apart from scientific data, operating complex, remotely located systems also requires large amounts of additional system information

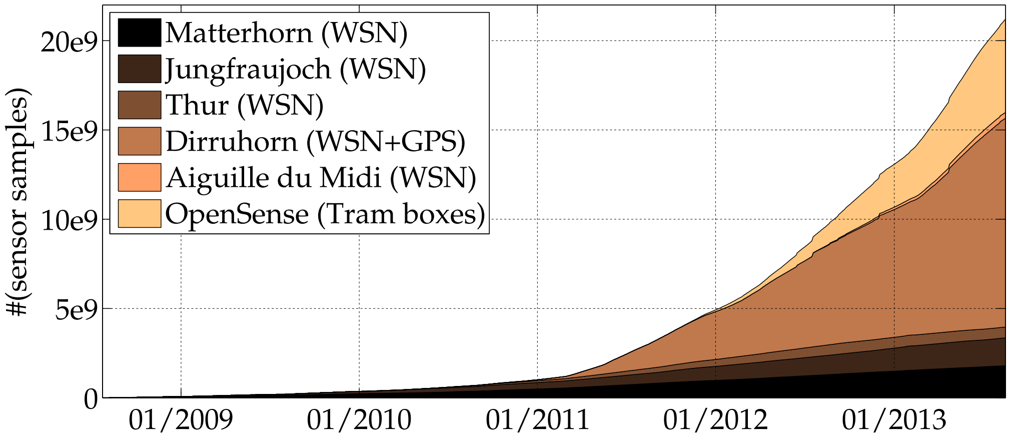


Fig. 3.6 Approximate amount of samples that serve as the input for the Vizzly instance at ETH Zurich. Please notice that a single communication frame, *e.g.*, a TinyOS packet, usually includes multiple samples. For retrieving a realistic estimation, only the final results of multi-step data processing chains are counted. Likewise, auxiliary header information are also not considered as input data for Vizzly. The large growth in the last two years is due to a large amount of new, high-rate sensors, *e.g.*, GPS and fine particle sensors, being deployed.

to be recorded. Health data is continuously sampled at all layers of the system architecture that ranges from low-power sensor nodes up to powerful back-end servers. Sampling intervals used range from 30 to 120 seconds.

Vizzly has been accepted as an essential tool for system supervision already within the first weeks of test operation. Being able to visualize data at arbitrary time scales down to single events allows on the one hand to assess the current system state, but also to detect long-term trends, *e.g.*, slowly degrading components. Apart from this specific application, Vizzly has also proven its usefulness for many other domain experts. Exemplary applications are the visual inspection of data quality, *e.g.*, outliers or data gaps, the visual inspection of signal characteristics, *e.g.*, its value range, and the visual selection of data segments that are the most suited for a particular analysis.

3.6.2 OpenSense: Mobile Sensors of Different Kind

The OpenSense project [ASC+10] investigates the challenge of monitoring urban air quality using (i) mobile sensing stations installed on top of public transport vehicles and (ii) personal sensors such as enhanced smartphones and pocket sensors [DAK+09] in the city of Zurich. The long-term goal is to raise community interest in air pollution data and to foster its involvement in monitoring air quality in urban areas.

To this date, ten trams have been equipped with sensing stations measuring ozone, carbon monoxide (CO) and fine particles (PM). Ozone and CO concentrations are measured every minute, the PM sensor generates one sample every 5 sec. Additionally, the GasMobile system is being used for measuring ozone concentrations with an Android smartphone [HSST12].

Over the past 15 months, one sensing station collected around 319 million data samples[12]. Around 174 million data samples are related to air quality measurements, remaining 145 million samples are dedicated to system management. See [LFS⁺12] for a more detailed description of the OpenSense data set.

Interactive browsing though time- and location-sensitive historical data at any desired level of detail is crucial (i) for fine-grained analysis of raw data by domain experts and (ii) for building various data processing and data access services on top of raw data. In particular, non-expert users are often interested in summaries on the development of air quality in a particular region and whether any limit on the concentration of pollutants was exceeded.

While the success of community-driven sensing highly depends on the interest of individual persons and the possibility to make data easy accessible and understandable for those parties, Vizzly helps OpenSense to solve challenges that arise with the large volumes of gathered data.

3.7 Performance Evaluation

All data visualizations of the PermaSense and OpenSense projects are currently handled by a single instance of Vizzly[13]. Users specify virtual signals of interest independent of a particular device, those user-specified templates are then used for automatically generating a virtual signal for every known device. Resulting 2,900 virtual signals originate from four data repositories, the total input currently consists of more than four billion unaggregated data samples. Most data originates from the PermaSense project and thus does not include location information. More than 118 million data samples that originate from the OpenSense project also include location information.

In the current configuration, unaggregated data points are first downsampled to the temporal resolution of $\hat{r}_{\text{temp}} := 4\,\text{min}$. The aggregation function used is the calculation of the mean value. While the temporal resolution could also be separately chosen for each virtual signal, this static setting is currently derived from the two minute sampling period used in PermaSense. Lower levels of detail are retrieved by further downsampling already aggregated data.

Aggregated data with the temporal resolution $\hat{r}_{\text{temp}}$ of 4 min is stored

---

[12] Note, that the stations are usually turned off over night.

[13] The public data interfaces of both projects can be accessed at http://data.permasense.ch and http://data.opensense.ethz.ch

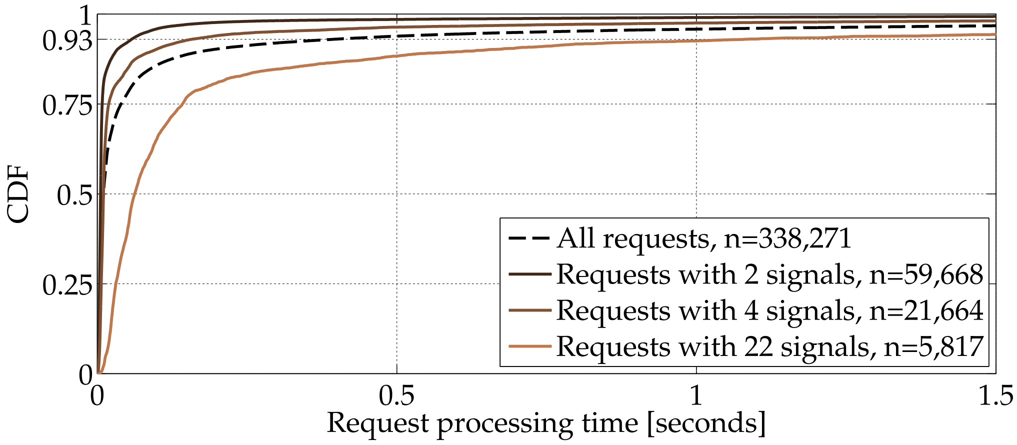


Fig. 3.7 Analysis of 338,271 requests. The number of virtual signals that is included in a request ranges from one to 55 virtual signals. Result data is first collected in sequential cache accesses and then combined to form a single output.

in a MySQL database. The MySQL server occupies 32 GB of disk space for approximately 595 million aggregated data points currently stored in the database. Further down-sampled data is additionally stored in memory, around 217 million aggregated data points with a temporal resolution of 960 sec occupy 2 GB of RAM. The discrepancy between memory usage and database usage per value is due to indexing information that is implicitly given when storing data in memory being explicitly stored in the database. Only two levels of temporal detail are stored, all other temporal resolutions are computed on-the-fly by down-sampling already aggregated data. While currently all virtual signals are treated equally, precious memory will be better utilized when contents are moved based on some metrics, *e.g.*, the number of requests for a particular virtual signal.

For understanding the performance of the Vizzly back-end, we are measuring the execution time of each request. Three separate measurements are made for each of the three phases that are required for serving a request, see Figure 3.5(a). Serving a single request can require the data of several virtual signals to be read in multiple cache accesses, the timing of each data access is measured separately. Apart from the execution times itself, other interesting metrics, *e.g.*, the number of returned data points, are also logged. While directly writing performance data to a database would significantly increase the response time of Vizzly, a background thread is in charge of asynchronously moving collected performance data from the memory to a MySQL database.

The following analysis is based on performance data that has been collected between March 2012 and July 2013. Generating the output usually takes less than a second. Cache accesses are made sequentially until all data that is needed for generating the output has been collected. The total request processing time is therefore larger when the number of virtual signals that are included in a request increases, see Figure 3.7. In

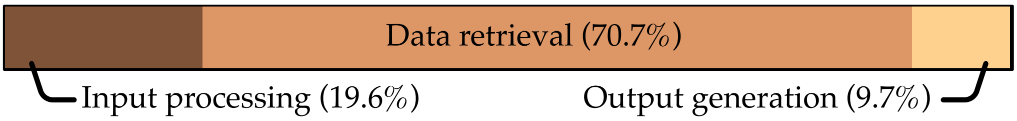


Fig. 3.8 The request execution time is dominated by the data retrieval phase. From 338,271 measured requests, fulfilling the request without transferring the result to the client took 0.4 seconds or less in 95% of the cases. More than 97% of the requests could be fulfilled within at most 1.6 seconds.

addition to the time needed for retrieving needed data from the cache, compressing the result and eventually sending the result to the client also takes a few tenth of a second. The distribution of the request execution time across the three phases needed for processing a request is shown in Figure 3.8. The request execution time is clearly dominated by the data retrieval phase. While initially processing the input data is very simple, generating the output may require the execution of an additional spatial aggregation or filtering.

Figure 3.9 shows the distribution of data fetch times over all recorded requests. An analysis that is limited to virtual signals that include both time and location information is shown in Figure 3.10. Aggregated data that is stored in memory (RAM) can be loaded very fast, the performance of loading aggregated data from the MySQL database is also acceptable. In contrast, loading unaggregated data from a GSN server can take several tens of seconds. Visible variations show no correlation with the size of the fetched result, but are caused by variations in the load of the system and by the changing state of other components, *e.g.*, the state of the database query cache.

As already aggregated data can be further processed on-the-fly without a noticeable delay, it is sufficient to cache only one aggregate of each virtual signal. While the current level of temporal detail can only be further reduced by down-sampling, the highest temporal level of detail in which aggregated data should be available must be cached. If the resulting data is too large for being completely stored in memory, it is reasonable to only keep frequently requested data in memory and to load all other data from the less constrained MySQL database.

3.8 Integrating and Extending Vizzly

Vizzly can be used out-of-the-box by deploying the server application and integrating provided front-end libraries into existing web pages.

However, individual project requirements, *e.g.*, the scheme used for storing data, might require Vizzly to be extended. For allowing us and other parties to easily extend Vizzly, the implementation of Vizzly follows modular design principles.

Regarding the Vizzly server, new data sources and cache back-ends

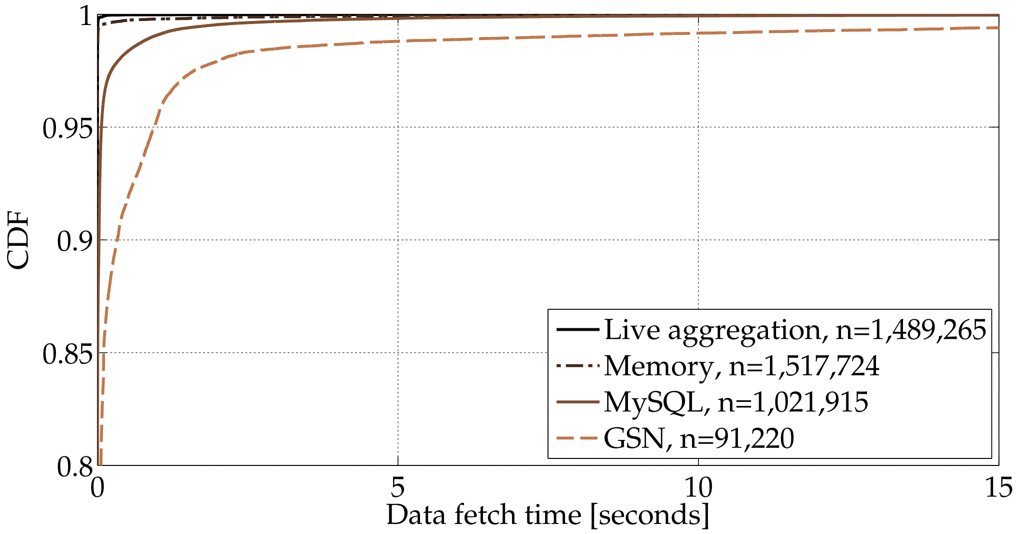


Fig. 3.9 Analysis of more than 4,120,000 data accesses. Retrieving already aggregated and indexed data from memory takes less than 4 milliseconds in 99% of all cases. Further down-sampling data in the temporal domain takes less than 7 milliseconds in 99% of the cases. The value of 99th percentile for loading already aggregated and indexed data from the MySQL database is 913 milliseconds. Loading unaggregated data from GSN takes 7.5 seconds or less in 99% of the cases.

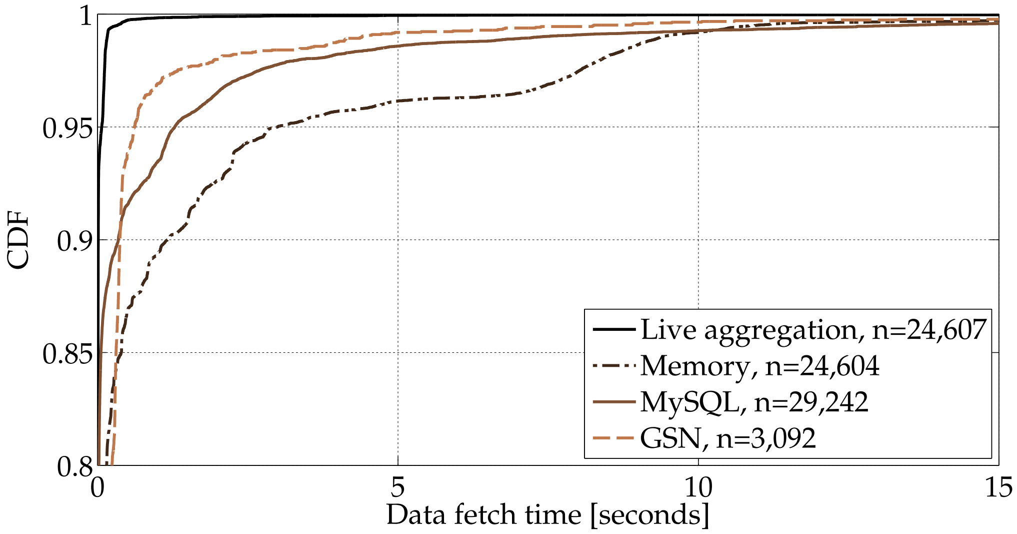


Fig. 3.10 Analysis of 81,500 accesses to virtual signals that include both time and location information. This amount corresponds to 2.0% of the total number of accesses being targeted to virtual signals that also include location information. This number is in line with 15 out of 2,900 virtual signals, thus 0.5% of the total number of virtual signals, containing location information. The shown performance of the memory cache is caused by the routine used for fetching data from the memory cache not fully leveraging the linear memory organization when data with location information is loaded. Likewise, the routine for accessing the MySQL database also needs further optimization. As a result of this analysis, corresponding routines have been improved in the most recent version of Vizzly.

can be implemented by either extending already existing components, *e.g.*, refining an existing MySQL data source, or by adding new components that follow defined abstract interfaces. On the front-end side, we find delivering CSV data as a common denominator that can be used together with many existing visualization libraries. While any client implementation must follow the defined request format of Vizzly, there are no further restrictions concerning platforms or libraries used for building new clients, *e.g.*, a native smartphone application.

Vizzly is open source software and free to use. The Vizzly project repository hosted on Google Code at https://code.google.com/p/vizzly includes all resources needed for installing and extending Vizzly.

3.9 Conclusions

We presented Vizzly, a middleware for the interactive browsing of large sensor network data sets. Vizzly can be easily integrated into existing systems, only a web browser is required for accessing map and line plot widgets. Vizzly has been successfully applied to both mobile and static sensing scenarios, its ability to handle very large data sets that consist of hundreds of millions of data points has clearly been proven feasible.

4

Multi-Hop Network Tomography

The previous chapter introduced Vizzly, a system for the visual inspection of large sensor network data sets. Without doubting the usefulness of this and similar systems, it becomes also clear that the manual inspection of data is limited to explicitly stated information such as sensor readings. Furthermore, visually correlating multiple streams is rather limited to the detection of long-term trends. Complex interactions that require considerable amount of history to be known cannot be tracked manually.

This chapter presents multi-hop network tomography, an algorithm for the reconstruction of the path, the per-hop timing and the per-hop ordering of individual packets. Achieving this requires information of all nodes that a packet visited while traveling through a network to be combined. By making interactions that happen inside a wireless sensor network visible, multi-hop network tomography addresses the observability gap that opens up when a network is moved from a controlled development environment to an uncontrolled deployment location.

4.1 Introduction

When preparing a sensor network deployment, a considerable research and development effort is usually spent customizing and optimizing a WSN implementation for a given application scenario. Different methods and tools such as abstractions [GLvB+03], simulators [LLWC03], testbeds [HKWW06] and diagnostics [YSSW07] have been developed for supporting and facilitating all phases of the life-cycle up to long-term operation.

However, appropriate tools are largely missing to support detailed performance analysis of deployed systems. When for example the network size, the sensing modalities or the environment, *e.g.*, due to the addition of new interferers like 4G/LTE equipment in the vicinity, are changing over the years of deployment, it is typically very hard to assess at which point in time the system must undergo minor, *e.g.*, new parametrization, or major, *e.g.*, addition or removal of features, modifications. While commonly available performance metrics, *i.e.*, data yield, end-to-end packet delays, radio duty-cycle, and link quality measurements are well-suited for giving an overview and estimating the general health of a deployed system, the root causes of an observed drop in the system performance remain mostly hidden.

Additional information are usually not available due to the costs and risks attached to their retrieval. For instance, transmitting extra information to aid such analysis in-band would increase the required bandwidth and is not always feasible. Furthermore, if a behavior is only observed after successful deployment, an addition of further data to be collected and transmitted is only possible with an update of all installed systems, either using a manual process, or by in-network reprogramming facilities. The latter of which is often regarded as an extra risk as it can easily render a system unusable [WALJ^{+}06]. Duplication of a similar system in a testbed setting and using this “replica” for analysis purposes in most cases suffers from economic feasibility and an inherent mismatch between production and test environments.

It is therefore desirable to devise comprehensive analysis capabilities based solely on the in-band information transmitted through a wireless sensor network. As a stepping stone for facilitating a passive but accurate health and performance monitoring of WSNs, this chapter presents multi-hop network tomography (MNT), a novel, non-intrusive algorithm for the reconstruction of the travelled path, the per-hop arrival order, and the per-hop arrival times of individual packets at runtime. Information is reconstructed outside the network immediately after a packet has been received at the sink. Concretely, we exploit the fact that packets are transferred through the network in a first-in first-out (FIFO) fashion. We find that the order in which packets arrive at the sink allows us create correspondences between packets that originate from independent sources but travel along similar paths. The apparent challenge lies foremost in the dynamics found in wireless multi-hop networks, *e.g.*, topology changes, packet reordering, and lost packets.

Specifically, multi-hop network tomography addresses the following problems: First, transferring path, per-hop arrival order, and per-hop arrival time information in-band does not scale for large networks as the amount of information grows linearly with the path length. As a result, the overall network performance may decrease due to congestion and higher packet loss rates [SAM03] when sending more and larger packets further aggravating the capability to understand network behavior in detail. Second, active methods for extracting desired information would also require to modify the software that is running on the sensor nodes. Not only introducing the effort of reconfiguring a deployed system, valuable path, per-hop arrival order and per-hop arrival time information from historic data would remain unknown. While this work primarily targets low-power WSNs, the method as such can be applied to any multi-hop network. It is clear however that the most benefit is achieved in resource-constrained scenarios commonly found in WSNs.

Apart from the analysis of deployed systems, MNT offers advantages in potentially any scenario in which no other communication channels, *e.g.*, serial ports, for extracting performance data are usable. For example, adding load on the serial interface for outputting additional performance data is also unfavorable in certain testbed settings, *e.g.*, when the execution timing must not be changed.

The contribution of this chapter is as follows:

- We present the MNT algorithm for reconstructing the packet path, the per-hop arrival order, and the per-hop arrival timing of individual packets.

- Based on a formal model of a real system, we proof the correctness of results obtained, *i.e.*, that extracted path, order and timing information match with ground truth.

- We validate the correctness of our implementation in extensive experiments with two well-known communication protocols, *i.e.*, CTP [GFJ+09], and Dozer [BvRW07], on testbeds of up to 90 nodes size. Here, testbed infrastructure allows us to extract ground truth without the need for congesting the in-band communication resource under investigation. We evaluate the performance of our algorithm in terms of the fraction of packets that can be reconstructed in various settings. The scalability of the approach is discussed using simulation.

- The application of the MNT algorithm to more than 270 million packets from three deployed systems is presented in a case study.

Related work including a discussion of the novelty of this work is presented in Section 4.2. Section 4.3 motivates the underlying problem of reconstructing data from partial information, the core principles used in multi-hop network tomography are presented in Section 4.4. Assumptions made when designing the MNT algorithm are introduced in Section 4.5, the full algorithm is presented in Section 4.6. Section 4.7 presents our validation done on real hardware and in simulation, our case study based on data from real-world deployments is presented in Section 4.8. The broader applicability and limitations of the MNT algorithm are discussed in Section 4.9, Section 4.10 concludes this chapter.

4.2 Related Work

A. Performance of Wireless Sensor Networks

The performance of wireless sensor networks has been intensively studied at several layers. For instance, [ZG03] and [SDTL10] study the link-layer performance in numerous configurations and environments. Various protocol papers, *e.g.*, CTP [GFJ⁺09] and the low-power wireless bus [FZMT12], discuss the performance of the routing layer. The end-to-end application performance is subject of several deployment reports, *e.g.*, [LBV06, BISV08, HSL⁺11]. An extensive cross-layer performance study is provided by [MPC⁺10]. Located on the intersection between the routing and the application layer, the purpose of this work is to passively reconstruct hidden network performance data.

B. Network Health Monitoring

For dealing with the inherent challenges found in the long-term operation of WSNs, *e.g.*, hardware failures, several solutions for the run-time monitoring of deployed systems have been proposed [RCK⁺05, LLL10]. While the problem of how to combine measured information for inferring a root cause is orthogonal to this work, such systems could potentially benefit from per-packet path and timing information that is provided by the MNT algorithm. Ideally, we expect reconstructed per-hop timing information to even facilitate the development of health monitoring systems that are also able to automatically report small variations of the system performance before a major incident happened.

C. Wired Network Tomography

Network tomography is an important tool for network monitoring in wired IP networks. Without the need for cooperation of involved components, *e.g.*, routers, network structure and link-level performance characteristics, *e.g.*, delay and packet loss, are measured based on the travel of actively inserted probes. The problem of network tomography in wired networks has been well-studied, an extensive overview of available methods is given in [CHINY02]. In most cases, the studied problem is either to reconstruct the network structure only, or to determine link-level performance measurements for an a priori known network topology. Regarding our goal of reconstructing both the network structure and link-level characteristics, we found the work of Rabbat *et al.* [RNC04] as the possibly earliest work that already covers both dimensions in the wired scenario.

D. WSN Network Tomography

Network tomography algorithms for wireless sensor networks are restricted by WSNs supporting much less probing traffic than wired networks. Nguyen *et al.* [NT06] propose the application of statistical methods, *i.e.*, Maximum likelihood and Bayesian approaches, for the identification of lossy links. While the network topology is assumed to be known, the method of Nguyen *et al.* also allows for multiple, dynamic topologies by splitting a trace into so called "routing time slots" in which the topology is assumed to be stable. Being interested in learning an unknown network topology, Liu *et al.* [LLL10] propose an active marking scheme for the reconstruction of topology information at the sink. Here, extracted path information is not guaranteed to be correct.

Without adding extra probing traffic to the network, the MNT algorithm extracts all information from already existing application traffic. To the best of our knowledge, this is the first work that aims for the reconstruction of detailed per-packet information while also giving guarantees on the correctness of extracted information.

4.3 Exploiting Information Implicitly Given

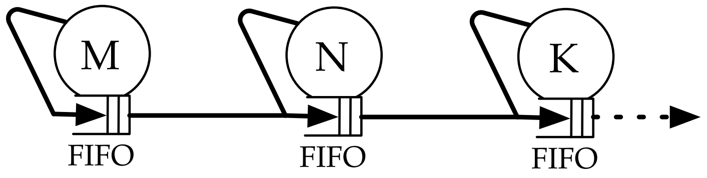
Measurements taken inside a sensor network provide additional detail for the understanding of an observed end-to-end system performance. For example, an observed end-to-end packet delay might have several causes, *e.g.*, transmission failures, back-pressure, or unfair resource allocation. However, the amount of information that can be transferred in-band is limited as additional load potentially threatens the performance of the system under investigation.

For enabling detailed performance analyses of sensor networks in spite of resource constraints this work wants to answer the following questions: Is it sufficient to transmit only partial information in-band in order to reconstruct missing information outside the network? What information is commonly transferred and thus already available in sensor network applications? Which useful information is implicitly given by the structure and behavior a-priori known from a given WSN application, and thus does not need to be transmitted over the precious communication resource? How much information of which detail and accuracy can be reconstructed outside the sensor network? Are corresponding methods applicable to a broader set of applications?

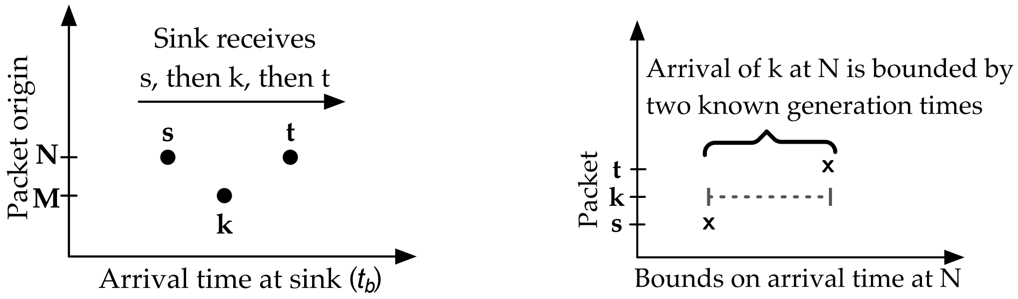
As one concrete example, this chapter presents multi-hop network tomography (MNT) for the reconstruction of the path, the per-hop arrival order, and the per-hop arrival time of individual packets from partial information.

4.4 Multi-Hop Network Tomography

In multi-hop data collection applications, sensor nodes have the dual functionality of (i) generating packets and (ii) forwarding packets of other nodes that are more distant to the sink. Generated packets typically include certain application header information, *e.g.*, the source address and a packet generation timestamp. For reconstructing information at



(a) Topology


Sink receives
s, then k, then t
Packet origin
N
M
s
t
k
Arrival time at sink (t_b)
Arrival of k at N is bounded by
two known generation times
Packet
t
k
s
x
x
Bounds on arrival time at N

(b) Selection of anchor packets

(c) Arrival time estimation

Fig. 4.1 Reconstruction of packet path and arrival time bounds of packet *k* based on information from packets *s* and *t*. Packets *s* and *t* were subsequently generated at node *N*. Packet *k* from node *M* arrived at *N* after the generation of *s*, but before the generation of *t*.

the sink, multi-hop network tomography exploits the fact that state-of-the-art protocols like CTP maintain a single packet queue to which both locally generated packets and forwarded packets are added while traveling through the network. In the common case correspondences created between forwarded and locally generated packets are still visible at the sink, *i.e.*, packets arrive at the sink in the same order as they left a node within the network. The path, the per-hop arrival order and the per-hop arrival timing of individual packets are reconstructed by a per-hop correlation of information from both locally generated packets and forwarded packets.

In the exemplary situation in Figure 4.1, packets *s* and *t* were consecutively generated at a node *N*. In contrast, packet *k* was generated at another node *M*. Minimal topology information is provided by the address of the first-hop receiver, *e.g.*, packet *k* reports node *N*, packets *s* and *t* report node *K*, that is transmitted with every packet. Packet *k* arrived at *N* after the generation of *s*, but before the generation of packet *t*. Given that none of the packets was either lost, duplicated or reordered in this example, the observable order of arrival at the sink, see Figure 4.1(b), matches with the non-observable order of arrival at the intermediate node *N*, see Figure 4.1(c).

Packets *s* and *t* are selected as so called "anchor packets" and used for reconstructing (i) at which time packet *k* arrived at node *N*, and (ii) to which node packet *k* was forwarded after leaving node *N*. Concretely, information is inferred from (i) the known packet generation time of packets *s* and *t*, see Figure 4.1(c), and (ii) the first-hop receiver reported by packets *s* and *t*. For every packet, this procedure is repetitively applied until the packet has been traced up to the sink.

Due to phenomena common to WSNs, *e.g.*, packet loss, packet duplication and packet reordering, observations made at the sink might not match with the reality. Therefore, the MNT algorithm has to assess for each packet if information can safely be reconstructed, or if there is the risk of obtaining incorrect information. Based on the formal model of a system that is presented in the next Section 4.5, the description of the full MNT algorithm is situated in Section 4.6.

4.5 System Model

This section summarizes assumptions made and variables used by the MNT algorithm. Details on the adaptation of this generic model to more complex systems are presented in Section 4.5.1, *i.e.*, multiple sinks and systems that include multi-layered storage architectures.

We assume a multi-hop wireless sensor network that consists of a number of static sensor nodes and a sink. Communication is based on a tree-based routing protocol. The network operation is subject to phenomena common to wireless sensor networks, *i.e.*, topology changes, packet loss, packet duplication and packet reordering. All nodes produce and relay data in the sense of a data collection application. Nodes do not have access to global timing information and rely on a local clock.

FIFO send queue. All sensor nodes have the dual functionality of (i) generating data locally and (ii) forwarding packets received from other nodes. Sensor nodes maintain a finite FIFO queue for all outgoing packets. A packet is immediately added to this queue after generation in the local application or arrival on the radio. If connected, a sensor node transmits the contents of its send queue to the next hop. If single-hop communication fails, a sensor node keeps retransmitting the currently selected packet until this packet was successfully acknowledged by the parent node, or if the maximum number of transmission attempts is reached.

Delay-tolerant data generation. Sensor nodes are generating data, *i.e.*, recording status information, that is transmitted to the sink. The timing of data generation is unspecified. Sensor nodes may also generate data while disconnected from the network. In this case, packets are buffered on the sensor node until connectivity is reestablished.

We further assume that the information listed in Table 4.1 is known for each packet k that has been received at the sink.

Timing information. The arrival time at the sink $t_b(k)$ is measured on a perfect clock and known for any packet k. To allow for packet time-stamping mechanisms with inaccuracies, we assume that only an estimate $\tilde{t}_g(k)$ of the packet generation $t_g(k)$ is accessible for all packets. The error

Packet Application Headers	
$o(k)$	Source node network address
$\tilde{t}_g(k)$	Estimated packet generation time
$p(k)$	Network address of the current parent
Added on Arrival at the Sink	
$t_b(k)$	Arrival time at the sink
From Analysis, Packet Headers, or Post-Processing	
$\Delta^{u,l}(k)$	Upper and lower bounds on the accuracy of the estimated packet generation time $\tilde{t}_g(k)$
$id_N(k)$	Packet index reflecting the correct order of generation for packets originating from a node N

Tab. 4.1 Overview of system model variables

of this estimate is bounded by $\tilde{t}_g(k) - \Delta^l(k) \leq t_g(k) \leq \tilde{t}_g(k) + \Delta^u(k)$. Here, $\Delta^u(k)$ and $\Delta^l(k)$ denote the upper and lower bounds on the error of the time-stamping mechanism used. $\Delta^u(k)$ and $\Delta^l(k)$ are not transmitted as part of a packet k, but assumed to be derived from an analysis, *e.g.*, an analysis on the accuracy of the clock synchronization scheme used. As packets are immediately added to the send queue after generation, $\tilde{t}_g(k)$ is a valid proxy for the arrival time of a packet k at the send queue of its source.

Sequencing information. We assume the existence of a unique index $id_N(k)$ that yields the correct order of packet generation for packets originating from an individual node N: $id_N(u) > id_N(v)$ iff $t_g(u) > t_g(v)$. Packet loss is an artifact that is common to wireless multi-hop networks and therefore cannot be avoided. We assume that $id_N(k)$ also allows us to detect packet loss: Given a packet v generated at N in direct succession of a packet u, it must also hold that $id_N(v) \equiv id_N(u) + 1$. Practically, $id_N(k)$ can be obtained by submitting $id_N(k)$ as part of each packet, or by using post-processing algorithms that can reconstruct $id_N(k)$, *e.g.*, the algorithm that has been presented in Chapter 2 of this thesis.

Piggy-backed topology information. Each packet k carries a source address $o(k)$ and a first-hop receiver address $p(k)$. For retrieving up-to-date information, $p(k)$ is updated immediately before each try of transmitting the packet to the current parent. While transferring parent information is a common best practice in many real applications, *e.g.*, for being able to generate snapshots of the network topology, certain protocols, *e.g.*, [WTC03], even require this information to be transmitted for their operation, *e.g.*, for enabling passive neighbor discovery. For applications that do not yet transfer first-hop receiver information, a single field, we follow the argumentation of Liu *et al.* [LLL10] while considering the introduced traffic overhead to be negligible.

For the MNT algorithm to be able to trace packets based on the topology information described, we need to add an assumption concerning the observability of parent changes: For our further argumentation (see Section 4.6.3.1), it must hold that the parent of N cannot have changed between the successful transmission of consecutively, locally generated packets u and v, if we observe $p(u) \equiv p(v)$. Therefore, we can only allow for up to one parent change between the successful transmission of locally generated packets u and v. While this is satisfied in the common case, properly handling the rare situation of more than one consecutive parent change between the transmission of two locally generated packets would only ask for a small modification of the protocol operation, *e.g.*, letting nodes locally count the number of parent changes since the most recently transmitted topology information and mark packets that were forwarded after the second, consecutive change.

4.5.1 Modeling More Complex Systems

None or only little extra information is required for adapting our model to significantly more complex systems:

Multi-layered storage architecture. For supporting high sampling rates and disconnected operation, recent system designs envision sensor nodes to be equipped with extra hardware for bulk storage, *e.g.*, FRAM [CMP+09] or SD memory cards [BGH+09]. Instead of maintaining a single packet queue only, locally generated packets are firstly added to a second queue that is situated on the added storage. Extra information is needed for supporting multi-layered storage architectures, namely the time spent in other queues before a packet was ultimately added to the send queue.

Multiple sinks. Sensor nodes may concurrently transmit packets to multiple sinks, *e.g.*, [MP11]. Here, sensor nodes are at the same time part of multiple concurrent tree topologies. A concrete multi-sink system conforms to our system model if its operation can be abstracted as the concurrent operation of multiple single-sink data collection trees that individually conform to our system model. Data received at each sink is then analyzed separately.

4.6 Safe Information Reconstruction

The MNT algorithm for reconstructing the travelled path, the per-hop arrival order, and the per-hop arrival times of individual packets is based on three core principles: First, path information is reconstructed by a per-hop correlation of locally generated packets and forwarded packets. Here, we exploit that locally generated packets include the address of the first-hop receiver. Second, the per-hop arrival order of both locally generated packets and forwarded packets is inferred from the observed order of packet arrival at the sink. Third, packet generation time information of locally generated packets is used to bound the per-hop arrival time of forwarded packets that arrived at a respective node immediately before or after packet generation.

The correctness of information inferred is threatened by phenomena common to WSNs, namely topology changes, packet loss, and packet reordering. In the context of the MNT algorithm, we need to address the following two problems: Firstly, observed and real packet paths of individual packets must match. Therefore, we can only argue about packets for which we can guarantee that those packets in any case can only have travelled along exactly one path. Likewise, per-hop order information inferred from the observed order of arrival at the sink must also match with the real packet order at packet queues within the network. Path changes are the single source of packet reordering in multi-hop networks. Thus, we must ensure that a packet can not have been reordered due to a path change before we are allowed to reconstruct information of this packet or to use this packet for reconstructing information of other packets.

Given a trace $\mathcal{P}$ of packets received, the first step of the MNT algorithm is to determine the set $\mathcal{R}$ of so called "reliable" packets. For packets within this set, we introduce the concept of "anchor packets" for reconstructing packet path, per-hop order, and per-hop arrival times of individual packets at the sink: Given a forwarded packet k that was forwarded to a node N, "anchor packets" s and t correspond to locally generated packets that were generated at node N immediately before and after the arrival of k at N. Generation time information and first-hop receiver information of both s and t are used to firstly bound the time of arrival of k at N, and secondly to deduce the next hop to which k travelled after N.

In the following, we will first describe the concept of information reconstruction using "anchor packets" in Section 4.6.1. The correctness of the results obtained is threatened by artifacts of path changes. After specifying the concrete impact on our problem in Section 4.6.2, the following Section 4.6.3 describes the properties of "reliable" packets and how a set $\mathcal{R}$ of "reliable" packets can be determined given a trace of the received packets. Further extensions for improving extracted information using forward and backward reasoning are presented in Section 4.6.4. Reconstructed timing information is often too pessimistic and can be improved by correlating information of multiple packets.

4.6.1 Packet Correlation Using Anchor Packets

This section formally describes the most integral concept of information reconstruction using "anchor packets". For clarity and brevity, the following description assumes that all involved packets are members of the corresponding set of "reliable" packets $\mathcal{R}$, and therefore reconstructed information is correct. The construction of a set of "reliable" packets will be described afterwards in Section 4.6.3.

Given a packet k, we want to reconstruct the following information:

- **Packet path** $\mathcal{N}_k$: Starting at the packet source $o(k)$, the ordered set $\mathcal{N}_k$ contains all nodes that packet k visited until arriving at the sink node S. The order of items in $\mathcal{N}_k$ reflects the order of visited nodes.

- **Queue index** $qid_N(k)$: For all nodes N that k visited, *i.e.*, $\forall N \in \mathcal{N}_k$, we want to build a queue index qid_N so that qid_N reflects the order of packet arrivals at N: The queue index is larger, *i.e.*, $qid_N(m) > qid_N(n)$, iff packet m arrived at N after another packet n, *i.e.*, $t_a(N, m) > t_a(N, n)$. In contrast to the already known packet index $id_N(k)$, the queue index $qid_N(k)$ provides not only the sequence of locally generated packets, but also that of forwarded packets.

- **Bounds on queue arrival time** $t_a^{u,l}(N, k)$: For all nodes N that k visited, *i.e.*, $\forall N \in \mathcal{N}_k$, we want to bound the unknown queue arrival time $t_a(N, k)$ so that $t_a^l(N, k) \leq t_a(N, k) \leq t_a^u(N, k)$.

The reconstruction process for any packet k starts at the source node $o(k)$ where we are immediately able to assign the queue index $qid_{o(k)}(k)$, the arrival time bounds $t_a^u(o(k), k)$ and $t_a^l(o(k), k)$, and the first two entries of the packet path $\mathcal{N}_k$. Concretely, the queue index $qid_{o(k)}(k)$ is initialized with a multiple of the known packet index $id_{o(k)}(k)$, *i.e.*, $qid_{o(k)}(k) := id_{o(k)}(k) \cdot c$ with $c > 1$. By multiplying the packet index $id_N(k)$, we give room for adding forwarded packets that arrived in between locally generated packets. Therefore, the multiplier c must be larger than the maximum number of forwarded packets that can arrive in between two consecutively generated packets.

Next, arrival time bounds are initialized using upper and lower bounds on the packet generation time:

$$t_a^l(o(k), k) := \tilde{t}_g(k) - \Delta^l(k) \tag{4.1}$$

$$t_a^u(o(k), k) := \tilde{t}_g(k) + \Delta^u(k) \tag{4.2}$$

Likewise, the arrival time $t_a^{u,l}(S, k)$ at the sink corresponds to the known time of arrival at the sink $t_b(k)$: $t_a(S, k)^{u,l} := t_b(k)$. The packet path $\mathcal{N}_k$ is initialized with $\mathcal{N}_k := \{o(k)\}$. The next hop corresponds to the known first-hop receiver $p(k)$, we initialize $N^* := p(k)$, and start searching for anchor packets s and t at N^*:

$$s := \arg\max_x t_b(x) \text{ for all } x : o(x) \equiv N^* \wedge t_b(x) < t_b(k) \tag{4.3}$$

$$t := \arg\min_x t_b(x) \text{ for all } x : o(x) \equiv N^* \wedge t_b(x) > t_b(k) \tag{4.4}$$

Regarding all packets that were generated at N^*, s is the packet that arrived at the sink latest before k. Likewise, packet t arrived at the sink earliest after k. While we can only observe the order in which s, t and k arrived at the sink, (4.3) assumes that if $t_b(s) < t_b(k)$, it also holds that $t_a(N, s) < t_a(N, k)$. Likewise, (4.4) assumes that if $t_b(t) > t_b(k)$, it also holds that $t_a(N,t) > t_a(N,k)$. We will show in Section 4.6.3 that this assumption is backed by packets s, k and t being members of the corresponding set $\mathcal{R}$ of "reliable" packets.

The complete packet tracing algorithm is shown in Algorithm 4.1. The anchor packet selection is situated between lines 5 and 8. Tracing must firstly stop, if we cannot find anchor packets s and t (line 9), if found packets s and t were not consecutively generated (line 10), *i.e.*, not all relevant packets are also part of the set of "reliable" packets, or if we cannot safely determine the next hop (line 11). The lowest free queue index $qid_{N^*}(k)$ that is smaller than the queue index $qid_{N^*}(t)$ of the anchor packet t is determined in line 13.

Algorithm 4.1: Reconstruction of the path, the per-hop arrival order, and per-hop arrival times of a packet k

```
   input: Packet $k$ with origin $o(k)$, first-hop receiver $p(k)$ and arrival time at the sink
          $t_b(k)$. $k \in \mathcal{R}$
 1 begin
 2     $t_a^l(o(k),k) \longleftarrow \tilde{t}_g(k) - \Delta^l(k)$ ; $t_a^u(o(k),k) \longleftarrow \tilde{t}_g(k) + \Delta^u(k)$ ;
 3     $\mathcal{N}_k \longleftarrow \{o(k)\}$ ; $N^* \longleftarrow p(k)$ ;
 4     while $N^* \not\equiv S$ do
 5         $s \leftarrow \arg\max_x t_b(x)$
 6             for all $x : x \in \mathcal{R} \wedge o(x) \equiv N^* \wedge t_b(x) < t_b(k)$ ;
 7         $t \leftarrow \arg\min_x t_b(x)$
 8             for all $x : x \in \mathcal{R} \wedge o(x) \equiv N^* \wedge t_b(x) > t_b(k)$ ;
 9         if $s \equiv \{\}$ or $t \equiv \{\}$ then break ;
10         if $id_{N^*}(s) \not\equiv id_{N^*}(t) - 1$ then break ;
11         if $p(s) \not\equiv p(t)$ then break ;
12         $\mathcal{N}_k \longleftarrow \mathcal{N}_k \cup \{N^*\}$ ;
13         $qid_{N^*}(k) \longleftarrow 1 + \max_{qid_{N^*}} qid_{N^*} < qid_{N^*}(t)$ ;
14         $t_a^l(N^*,k) \longleftarrow \tilde{t}_g(s) - \Delta^l(s)$ ; $t_a^u(N^*,k) \longleftarrow \tilde{t}_g(t) + \Delta^u(t)$ ;
15         $N^* \longleftarrow p(s)$ ;
16     end
17     $t_a^l(S,k) \longleftarrow t_b(k)$ ; $t_a^u(S,k) \longleftarrow t_b(k)$ ;
18 end
```

4.6.2 The Problem with Path Changes

Regarding our scheme of inferring information from observations made at the sink, path changes in the network can introduce two kinds of difficulties: (i) Observations may yield more than one possible path along which a packet k may have travelled. In this case, it is not further decidable which of those multiple paths corresponds to the correct path. (ii) Packets can get reordered, and thus arrive at the sink in a different order than they arrived at individual queues within the network. In both cases, inferred information is no longer guaranteed to be correct.

Let us outline those two problems in the following brief example of a parent change: In Figure 4.2, we see that the parent of a node N changes from node K to node L at a time t_x. Let us assume that packets n_{K1}, n_{K2} and

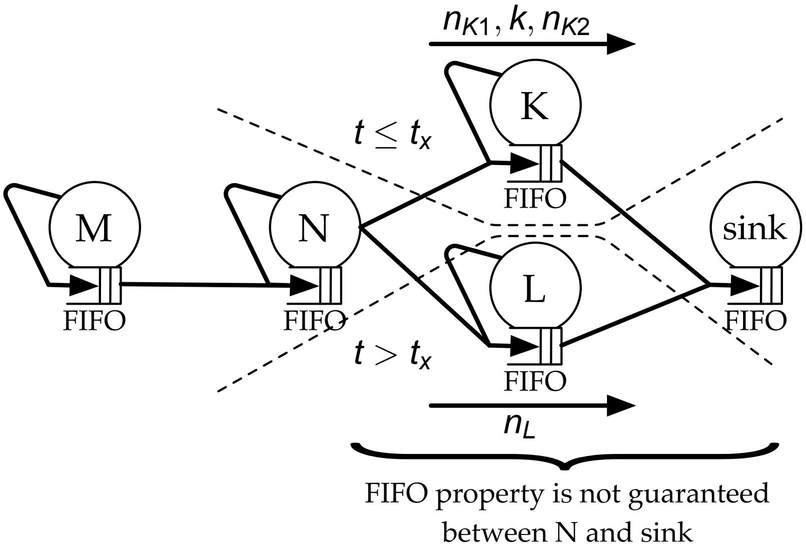


Fig. 4.2 Possible FIFO violation. In this example, the parent of node N changes from node K to node L at time t_x. Since packets on both paths may experience arbitrary delays, packets generated just before and after the topology change might not arrive at the sink in the order of generation anymore. This also affects packets being generated at more distant nodes, *i.e.*, node M.

n_L were generated at node N. Additionally, there is a packet k that was forwarded from a node M to node N in between the generation of n_{K1} and n_{K2}. While packets n_{K1}, k and n_{K2} were still forwarded to node K, packet n_L was the first packet that went to the new parent L. Although both paths individually forward packets in the correct order, a larger delay for packets traveling along the old path can lead to packets arriving out of order when packets from both paths join at the sink. For example, we now assume that packet n_L arrived at the sink before packets k and n_{K2}, thus out of order. This can lead to the following two problems: (i) Observations from the sink no longer suggest that k can only have travelled along its real path $\mathcal{N}_k := \{M, N, K, S\}$, but also along the new path over node L. (ii) Although n_{K1} and n_{K2} are the real "anchor packets" of k, the order of arrival at the sink would suggest to wrongly select n_L and n_{K2}.

4.6.3 Finding a Set of Reliable Packets

With the goal of ensuring that packet correlation using anchor packets is only applied when this procedure is safe, we propose to limit information reconstruction to a subset of the set $\mathcal{P}$ of the received packets, namely to a set $\mathcal{R} \subseteq \mathcal{P}$ of "reliable" packets.

A packet k is reliable, i.e., $k \in \mathcal{R}$, if it fulfills two properties: From our observations at the sink, we can guarantee that (i) packet k can only have travelled along exactly one path $\mathcal{N}_k$, and that (ii) the order relation between packet k and any other packet $m \in \mathcal{R}$ is consistent along all packet queues in the network including the sink.

Regarding the first condition, the underlying problem is that our observations made at the sink can yield multiple, ambiguous paths when it is not decidable if a packet k left a node N before or after node N switching to another parent. Needed timing and order information for deciding this problem are yet unknown. While multiple choices might lead to selecting another, but the real path, we can only reason about packets that can only have travelled along exactly one path.

The second condition allows us to reason about the packet arrival order at packet queues within the network from observations at the sink. While attached packet sequencing information $id_N(k)$ yields the sequence of packet generation at a node N, the condition for reliable packets is stronger as it covers both locally generated packets and forwarded packets at an arbitrary node N. Formally, the second condition asks for the following: *For any packets* $m, n \in \mathcal{R}$, *it holds that* $\forall N \in \mathcal{N}_m \cap \mathcal{N}_n : t_a(N, m) > t_a(N, n)$ *iff* $t_b(m) > t_b(n)$. Here, $t_a(N, m)$ and $t_a(N, n)$ are the unknown arrival times of packets m and n at node N. $\mathcal{N}_m \cap \mathcal{N}_n$ denotes the set of nodes that both m and n visited, and therefore the set of nodes for which an order relation between packets m and n exists.

We will now present how the MNT algorithm consecutively solves those two problems. First, we present a worst-case analysis that decides if a packet can only have travelled along exactly one path, or if there are ambiguities. Second, we will show in Section 4.6.3.2 how available packet sequencing information $id_N(k)$ of locally generated packets can be used for solving the second problem. The complete algorithm for finding a set of "reliable" packets is finally presented in Section 4.6.3.3.

4.6.3.1 Per-Hop Worst-Case Path Analysis

In the following, we will describe how we can test whether a packet k can only have left a node N to a unique next hop, or whether there are ambiguities. Starting at the known first-hop receiver $p(k)$, the following procedure is carried out per-hop until either the sink is reached, or we must conclude that we cannot decide along which path packet k travelled along.

Let us assume a packet k that was forwarded to a node N. We now want to determine whether we can guarantee that k can only have left node N to exactly one next hop, and if yes, to which next hop. In this two-part worst-case analysis, we first determine the set $\mathcal{W}$ of locally generated packets in between which k may have arrived at N. Secondly, we analyze whether all packets in $\mathcal{W}$ were forwarded to the same next hop. Here, we consider both observable parent changes, *i.e.*, packets reporting different first-hop receivers, and potential, hidden parent changes, *i.e.*, a lost packet.

We start with determining the set $\mathcal{W}$ of locally generated packets in between which k may have arrived at N. As the arrival time of packet k at node N is yet unknown, we must resort to bounding the arrival $t_a(N, k)$ of packet k at node N by the lower bound on the generation time, *i.e.*,

$\tilde{t}_g(k) - \Delta^l(k)$, and the arrival time at the sink $t_b(k)$, *i.e.*, $t_g^l(k) \leq t_a(N,k) \leq t_b(k)$. Based on those bounds, we determine packets u and v that were generated at N immediately before and after the earliest and latest arrival of k at N, respectively. Based on the packet indexes $id_N(u)$ and $id_N(v)$, we then determine the set $\mathcal{W}$ of packets in between which k can have arrived at N. Here, $\mathcal{W}$ includes u, v, and all packets received that were generated after and before u and v at node N, respectively.

$$u := \arg\max_x \tilde{t}_g(x) + \Delta^u(x) \text{ for all } x : o(x) \equiv N \qquad (4.5)$$
$$\wedge\ \tilde{t}_g(x) + \Delta^u(x) < \tilde{t}_g(k) - \Delta^l(k)$$
$$v := \arg\min_x \tilde{t}_g(x) - \Delta^l(x) \text{ for all } x : o(x) \equiv N \qquad (4.6)$$
$$\wedge\ \tilde{t}_g(x) - \Delta^l(x) > t_b(k)$$
$$\mathcal{W} := \Big\{ w \mid o(w) \equiv N \qquad (4.7)$$
$$\wedge\ id_N(u) \leq id_N(w) \leq id_N(v) \Big\}$$

Theorem 4.1. *We defined $\mathcal{W}$ as the set of all packets generated at a node N in between which a forwarded packet k must have arrived at N. Given a packet k that arrived at a node N in between two locally generated packets $m, n \in \mathcal{W}$, we can guarantee that k was forwarded to a single possible next hop if (1) all packets $m \in \mathcal{W}$ were received at the sink, i.e., no packet of $\mathcal{W}$ was lost, and (2) all packets $m \in \mathcal{W}$ carry the same first-hop receiver.*

Proof. For the proof, we now go back to our formal model where we require that the first-hop receiver can not change more than once between the successful transmission of two consecutively, locally generated packets. Concretely, for any parent change, there must be at least one transmitted packet m that carries the new parent as its first-hop receiver $p(m)$. Based on this assumption, there can be only one possible next hop if (1) all packets $m \in \mathcal{W}$ were received, and (2) all packets $m, n \in \mathcal{W}$ were forwarded to the same first-hop receiver, *i.e.*, $\forall m, n \in \mathcal{W} : p(m) \equiv p(n)$.

□

Practically, packet loss is detected using the packet index id_N: No packets between u and v were lost, iff the number of elements $|\mathcal{W}|$ of the duplicate-free set $\mathcal{W}$ equals the difference of $id_N(v)$ and $id_N(u)$ plus one, *i.e.*, $|\mathcal{W}| \equiv id_N(v) - id_N(u) + 1$. If packet loss is detected, we cannot add packet k to the set $\mathcal{R}$ of reliable packets.

4.6.3.2 Exclusion of Packet Reordering

If a packet k is guaranteed to have travelled along exactly one path, we can add k to the set $\mathcal{R}$ of "reliable" packets, if we can also guarantee that the order relation between packet k and any other packet $m \in \mathcal{R}$ is consistent along all packet queues in the network including the sink.

Here, our approach is to solve this problem at its source, namely parent changes within the network. Concretely, we want to analyze per hop if packets were reordered due to a parent change at this node. In the following, we present how this is done using sequencing information that is provided for locally generated packets.

We define $\mathcal{C}_N$ as the set of conflict-free packets originating from a node N. Concretely, it holds that $\forall m, n \in \mathcal{C}_N : t_b(m) > t_b(n)$ iff $id_N(m) > id_N(n)$, which essentially means that all packets that are in $\mathcal{C}_N$ arrived at the sink in the same order as they were generated at node N. While the number of possible subsets $\mathcal{C}_N \subseteq \mathcal{P}$ is arbitrarily large, we propose to maximize the size of $\mathcal{C}_N$ by mapping the problem of finding $\mathcal{C}_N$ to solving the maximum independent set problem [Lub85]. Therefore, we construct a graph in which each packet that originates from a node N under investigation is represented by a vertex. While the maximum independent set problem is about finding the largest subset of independent elements, we connect two vertices with an edge, if the requirement $t_b(m) > t_b(n)$ iff $id_N(m) > id_N(n)$ is violated between the corresponding packets m and n.

Theorem 4.2. *We defined $\mathcal{W}$ as the set of all packets generated at a node N in between which a packet k must have arrived at N. For any packet k that arrived at a node N in between two consecutively, locally generated packets $m, n \in \mathcal{W}$, it holds that packet k cannot have been reordered due to a parent change at N, if (1) k can only have left N to a single possible next hop, and (2) all packets m that are part of $\mathcal{W}$ are also part of the set of conflict-free packets $\mathcal{C}_N$, thus $\forall m \in \mathcal{W} : m \in \mathcal{C}_N$.*

Proof. Let us go back to the situation in Figure 4.2 and consider a node N that generated packets n_{K1}, n_{K2} and n_L. While n_{K1} and n_{K2} were forwarded to a node L, packet n_K was forwarded to another parent K. We further assume that all three packets are in $\mathcal{W}$, *i.e.*, $\mathcal{W} := \{n_{K1}, n_{K2}, n_L\}$. In the following, we will now discuss three possible cases concerning a forwarded packet k arriving in between two packets of $\mathcal{W}$.

Case 1: Packet k arrived in between n_{K2} and n_L: As packets n_{K2} and n_L were forwarded to different next hops and therefore must carry distinct first-hop receivers, k is already excluded from being a member of the set $\mathcal{R}$ of reliable packets due to ambiguities in the observable path.

Case 2A: Packet k arrived in between n_{K1} and n_{K2}, all packets n_{K1}, n_{K2} and n_L arrive at the sink in the same order as they arrived at N: Since all locally generated packets arrived in order, all three packets n_1, n_K and n_L are part of the conflict-free set of packets. In fact, packet k has not been reordered due to a parent change at N. Analyzing packet k continues until the sink S is reached.

Case 2B: Packet k arrived in between n_{K1} and n_{K2}, packets arrive at the sink in the inconsistent order $n_{K1} \rightarrow n_L \rightarrow k \rightarrow n_{K2}$: Packet k is no longer arriving at the sink in between its correct "anchor packets" n_{K1} and n_{K2}, but in between n_L and n_{K2}. It becomes apparent, that n_L arriving at the sink before n_{K2} is a conflict, which means that at most one of those two packets can still be in the set of conflict-free packets $\mathcal{C}_N$. Therefore, not all packets that are part of $\mathcal{W}$ are now also part of $\mathcal{C}_N$ anymore. In consequence, k is no longer guaranteed to not have been reordered, and therefore not added to the set of "reliable" packets $\mathcal{R}$.

$\square$

4.6.3.3 Algorithm for Finding a Reliable Set

In the following presentation of the complete algorithm for constructing a set of "reliable" packets $\mathcal{R}$, we will now combine our previous findings. Here, we construct $\mathcal{R}$ by deciding for each packet $k \in \mathcal{P}$, if k is also a member of $\mathcal{R}$. While the corresponding sets of conflict-free packets $\mathcal{C}_N$ is not part of Algorithm 4.2, those sets are determined before executing Algorithm 4.2 by solving the corresponding maximum independent set problem (see previous Section 4.6.3.2).

Algorithm 4.2: Algorithm for deciding if a packet $k \in \mathcal{P}$ is also a member of the set of "reliable" packets $\mathcal{R}$

```
   input: Packet $k$
1  begin
2      if $k \notin \mathcal{C}_{o(k)}$ then
3          return ;
4      end
5      $N^* \longleftarrow p(k)$ ;
6      while $N^* \not\equiv S$ do
7          $u \leftarrow \arg\max_x \tilde{t}_g(x) + \Delta^u(x)$ for all $x : o(x) \equiv N^*$
8              $\wedge\ \tilde{t}_g(x) + \Delta^u(x) < \tilde{t}_g(k) - \Delta^l(k)$ ;
9          $v \leftarrow \arg\min_x \tilde{t}_g(x) - \Delta^l(x)$ for all $x : o(x) \equiv N^*$
10             $\wedge\ \tilde{t}_g(x) - \Delta^l(x) > t_b(k)$ ;
11         if $u \equiv \{\}$ or $v \equiv \{\}$ then break ;
12         $\mathcal{W} \longleftarrow \{w \mid o(w) \equiv N^*$
13             $\wedge id_{N^*}(u) \leq id_{N^*}(w) \leq id_{N^*}(v)\}$
14         $\mathcal{I} \longleftarrow \{id_{N^*}(w) \mid w \in \mathcal{W}\}$ ;
15         if $|\mathcal{I}| \not\equiv \max \mathcal{I} - \min \mathcal{I} + 1$ then break ;
16         $\mathcal{P} \longleftarrow \{p(w) \mid w \in \mathcal{W}\}$ ;
17         if $|\mathcal{P}| > 1$ then break ;
18         if $\exists m \in \mathcal{W} : m \notin \mathcal{C}_{N^*}$ then break ;
19         $N^* \longleftarrow p(u)$ ;
20     end
21     if $N^* \equiv S$ then $\mathcal{R} \longleftarrow \mathcal{R} \cup \{k\}$ ;
22 end
```

We start with firstly determining if packet k arrived at the sink out of order w.r.t. packets originating from the same source (line 2). Then, we start with our analysis at the first-hop receiver $N^* := p(k)$. We can safely assume that k must have arrived at N^* later than it was generated, but earlier than it arrived at the sink, *i.e.*, $\tilde{t}_g(k) - \Delta^l(k) < t_a(N^*, k) < t_b(k)$. Thus, k can have arrived at N^* in between any of two locally generated packets

$m, n \in \mathcal{W}$. Here, Equations (4.5) to (4.7) for determining $\mathcal{W}$ are reflected by lines 7 to 13 in Algorithm 4.2.

This analysis is a worst-case analysis, because we must prematurely stop for safety reasons, if one of the following conditions is met: First, we must stop, if k might have been forwarded to not only one, but alternative other nodes. Evidence for this is found if we find evidence for packet loss in $\mathcal{W}$ (lines 14 to 15), as well as when not all packets in $\mathcal{W}$ were forwarded to the same first-hop receiver (lines 16 to 17). Second, we must also stop, if we find evidence that a packet $w \in \mathcal{W}$ arrived at the sink out of order w.r.t. another locally generated packet v (line 18). In the good case, we continue analyzing k in the context of the next hop, *i.e.*, $N^* := p(u)$ (line 19). Packet k is added to $\mathcal{R}$, if we can safely trace the packet until the sink is reached (line 21). At the end of this algorithm, we constructed a set $\mathcal{R} \subseteq \mathcal{P}$ so that packet correlation using anchor packets is safe for all packets $k \in \mathcal{R}$.

4.6.4 Forward and Backward Reasoning

Initially set bounds on packet arrival times $t_a(N, k)$ are often pessimistic. As a first improvement, a packet can apparently not have arrived earlier than it was generated:

$$\forall N \in \mathcal{N}_k : t_a^l(N, k) := \max\left(\tilde{t}_g(k) - \Delta^l(k), t_a^l(N, k)\right) \tag{4.8}$$

Likewise, the arrival time at the sink $t_b(k)$ of a packet k marks the largest upper bound:

$$\forall N \in \mathcal{N}_k : t_a^u(N, k) := \min\left(t_b(k), t_a^u(N, k)\right) \tag{4.9}$$

4.6.4.1 Forward and Backward Queue Traversal

For achieving further improvements of the arrival time of a packet k at a node N, the timing information of packet k can be correlated with information of other packets that also arrived at node N.

Given a packet k that was arriving at the queue of a node N, thus $N \in \mathcal{N}_k$, the following equations state, that the observable order of arrival at node N must also be reflected by the upper and lower bounds on the arrival time $t_a(N, k)$:

$$\forall u, N \in \mathcal{N}_u, t_b(u) < t_b(k) : t_a^l(N, k) := \max\left(t_a^l\,(N, k)\,, t_a^l\,(N, u)\right) \tag{4.10}$$

$$\forall v, N \in \mathcal{N}_v, t_b(v) > t_b(k) : t_a^u(N, k) := \min\left(t_a^u\,(N, k)\,, t_a^u\,(N, v)\right) \tag{4.11}$$

As we are restricting us to packets $k \in \mathcal{R}$, the order of packet arrivals at the packet queue of node N matches with the observed order of packet arrivals at the sink. Similarly, bounds can also be improved by correlating information from different nodes that a particular packet visited.

4.7 Multi-Protocol Testbed Evaluation

In this section, we validate and evaluate our implementation of the MNT algorithm based on experiments with two well-known state-of-the-art communication stacks used for data collection, namely CTP [GFJ+09] and Dozer [BvRW07]. After describing the experimental setup, *i.e.*, the protocol selection, the packet time-stamping scheme used, and the infrastructure used for extracting ground-truth information, results obtained are validated by comparing them to ground truth. The performance of the implementation used is evaluated in terms of the fraction of packets for which information reconstruction succeeded, the time passed until results are available, and the accuracy of calculated arrival time bounds.

4.7.1 Experimental Setup

The results presented originate from three test environments: CTP Noe is executed on top of the standard low-power listening (LPL) MAC that is provided with version 2.1 of TinyOS. Those tests are carried out on up to 92 Tmote Sky (TI MSP430, CC2420 radio) nodes that are part of the TWIST testbed [HKWW06]. Twenty-five Tinynode 184 (TI MSP430, Semtech SX1211 radio) nodes of the also public FlockLab testbed [LFZ+13] are used for measurements that involve Dozer. For larger scaling tests on a 100-hop line topology, we are furthermore running experiments with an implementation of CTP [CS11] in the Castalia/OMNeT++[1] network simulator.

The purpose of the protocol selection and configuration used is to cover the following design aspect of routing protocols: While nodes are configured to turn on their duty-cycled radios every 62 ms and 250 ms, respectively, when running CTP Noe on top of the standard low-power listening [PHC04] MAC found in TinyOS 2.1, communication only takes place every 15 or 30 seconds, respectively, when running Dozer with a corresponding set of parameters. While the design space for routing protocols is definitely broader, *e.g.*, includes energy considerations, the tradeoff between reactivity and latency is of highest relevance in the context of this work. All protocols have only been modified to transmit per-packet information assumed by our formal system model. This requires only changes on the application-level, other layers remain untouched.

4.7.1.1 Packet Time-stamping

Providing simple integration, we decided to obtain packet generation timestamps using elapsed time on arrival [KDL+06]. We define $t_s(k)$ as the accumulated sojourn time that a packet k spent within the network.

---

[1] http://castalia.npc.nicta.com.au/

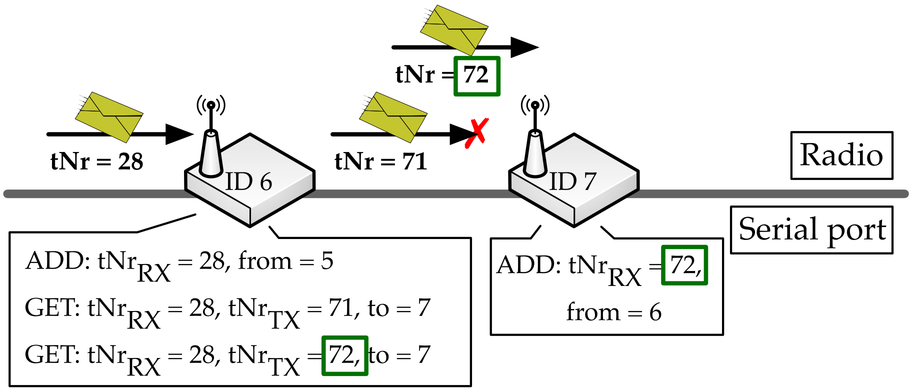

tNr = 72
tNr = 28
tNr = 71
ID 6
ID 7
Radio
Serial port
ADD: tNr$_{RX}$ = 28, from = 5
GET: tNr$_{RX}$ = 28, tNr$_{TX}$ = 71, to = 7
GET: tNr$_{RX}$ = 28, tNr$_{TX}$ = 72, to = 7
ADD: tNr$_{RX}$ = 72, from = 6

Fig. 4.3 Extraction of ground truth. For our validation tests, we instrumented code to make the travel of individual packets observable. This information is not transferred in-band, but over the serial port of the sensor node. Sensor nodes firstly generate a log message when a packet is added to the send queue (ADD), and secondly immediately before a packet is handed over to the radio for transmission (GET). The value of the local transmission counter *tNr* is incremented before any transmission attempt, and logged on both sender and receiver sides.

Ideally, the packet generation time $t_g(k)$ of a packet k is retrieved by subtracting the packet sojourn time $t_s(k)$ from the arrival time of the packet at the sink, thus $t_g(k) := t_b(k) - t_s(k)$. Here, it is assumed that $t_b(k)$ and $t_s(k)$ are measured on perfect clocks.

In the context of real clocks, we cannot measure $t_s(k)$, but the estimated packet sojourn time $\tilde{t}_s(k)$ that includes artifacts caused by measuring time on clocks with a low resolution and drift. After being initialized with $\tilde{t}_s(k) := 0$ on packet generation, this additional packet header is successively updated while a packet travels through the network. The inaccuracy of the resulting packet generation time estimate $\tilde{t}_g(k) := t_b(k) - \tilde{t}_s(k)$ of a packet k is as follows:

$$\Delta^u(k) := \frac{\tilde{t}_s(k)}{1+\hat{\rho}}, \quad \Delta^l(k) := \frac{\tilde{t}_s(k) + |\mathcal{N}_k| \cdot \hat{t}_u}{1-\hat{\rho}} \tag{4.12}$$

Here, we assume a bounded clock drift $\rho \in [-\hat{\rho}; \hat{\rho}]$, *e.g.*, $\hat{\rho} := \pm 60$ ppm, and a clock resolution of $\hat{t}_u$, *e.g.*, $\hat{t}_u := 1$ sec. As an error of $[0, +\hat{t}_u)$ can be introduced by each separate measurement, we must multiply $\hat{t}_u$ with the length $|\mathcal{N}_k|$ of the packet path $\mathcal{N}_k$.

4.7.1.2 Extraction of Ground Truth

For the validation of the MNT algorithm, we are interested in the ground truth w.r.t. the path, the per-hop arrival order, and the per-hop arrival time of individual packets. Therefore, we instrumented existing protocol code for outputting this information over the serial port of the sensor node. An observer device, *e.g.*, a more powerful, networked PC, is connected to the serial port of any sensor node, the received messages are logged to a file. Compared to sending ground truth information in-band over the radio, transmitting ground truth information over the serial port is very reliable, does not influence the packet stream under investigation, and also scales well for larger networks. Additionally, messages can be timestamped using the accurate, synchronized clock of the observer device. Ground truth is eventually reconstructed by correlating information from individual logs. For avoiding interference between serial port communication and radio communication, we took as much care as possible when integrating our instrumentation code into the existing software.

In more detail, sensor nodes log the following two events: Firstly, a log message is generated when a packet is added to the packet queue (ADD). This can be triggered by the application generating a new message, by the reception of a forwarded packet, or when a packet is copied from a secondary node storage, *e.g.*, a SD card. Secondly, a log message is also generated immediately before every attempt of transmitting a message over the radio (GET). Therefore, there can be multiple occurrences of GET events for a single packet. The occurrences of both events are timestamped on the local node clocks, the time difference between corresponding ADD and GET events yields the local packet sojourn time of a packet on a particular node. The sink is simply forwarding any received packet to the serial port.

Packet duplications can lead to multiple copies of a single packet being simultaneously traveling through the network. For being able to distinguish between multiple instances of a single packet, each packet transmission is made uniquely detectable by adding a transmission counter. Each sensor node is individually counting its local transmission attempts, the current counter value is added to each packet just before the packet is passed over to the radio. Sending the counter value in-band allows to log the respective value at both sides and finally to match the traces of the sending and the receiving node. An illustrating example of this mechanism is shown in Figure 4.3.

4.7.2 Validation and Evaluation Results

Validation and evaluation results from nine different test configurations are shown in Table 4.2. Results from simulation are in line with results obtained from experiments on real hardware. Varying test durations and network sizes are a result of varying availabilities of testbed resources. Apart from available time slots, tests are also limited to sensor nodes for which serial logging turned out to be successful in a pre-test.

For evaluating the sensitivity of the MNT algorithm w.r.t. the inter-packet interval (IPI), tests are repeated using varying packet generation rates. While using a periodic packet generation scheme allows us to make the sensitivity w.r.t. the IPI more visible, the MNT algorithm neither requires packet generation to be periodic, nor asks for all sensor nodes

	N	D	IPI	H	PC	DY	PACKETS RECEIVED
			CTP Noe/LPL				
A)	92	10 hour	15 sec	4	274	99.0%	217,078
B)	85	9 hour	30 sec	3	122	98.3%	88,541
C)	91	9 hour	120 sec	4	416	99.2%	54,143
			CTP (Simulation)				
D)	100	5 hour	30 sec	100	0	99.9%	60,150
E)	100	20 hour	120 sec	100	0	99.9%	60,112
			Dozer				
F)	25	12 hour	15 sec	3	548	99.8%	48,321
G)	25	16 hour	30 sec	4	193	99.9%	35,220
H)	25	24 hour	120 sec	4	196	99.9%	18,216
I)	10	60 hour	*120 sec	3	37	99.5%	84,563

(a) Characterization of test cases A) to I)

	PACKETS			PROC. DELAY		UNCERTAINTY	
	RELIABLE	FULL PATH	CORRECT	$p_{0.9}$	$p_{0.98}$	$p_{0.9}$	$p_{0.98}$
			CTP Noe/LPL				
A)	99.0%	98.9%	100.0%	17 sec	20 sec	<1 sec	<1 sec
B)	98.9%	98.9%	100.0%	32 sec	40 sec	<1 sec	1 sec
C)	98.4%	98.2%	100.0%	62 sec	75 sec	1 sec	1 sec
			CTP (Simulation)				
D)	96.5%	93.2%	100.0%	30 sec	31 sec	<1 sec	<1 sec
E)	97.0%	94.3%	100.0%	120 sec	121 sec	<1 sec	<1 sec
			Dozer				
F)	91.3%	91.2%	100.0%	537 sec	1,363 sec	15 sec	90 sec
G)	92.4%	92.3%	100.0%	1,024 sec	2,201 sec	30 sec	120 sec
H)	98.5%	98.4%	100.0%	120 sec	180 sec	84 sec	114 sec
I)	97.7%	97.7%	100.0%	122 sec	212 sec	74 sec	118 sec

(b) Results of multi-hop network tomography

Tab. 4.2 Validation and evaluation based on testbed experiments and simulation. Shown are the number of sensor nodes (N), the duration of the test (D), the inter-packet interval (IPI), the height of the data collection tree in hops (H), the number of observed parent changes (PC), and the data yield (DY). The fractions of packets that were part of the "reliable" set and the fractions of packets that could be fully reconstructed up to the sink are specified based on the number of packets received. The distribution of the processing delay is given by the 90th and 98th percentiles. Similar, the per-hop arrival time bounds uncertainty, *i.e.*, the difference of upper and lower bounds, is also given using percentiles. (*) Deployment configuration in which five packets are generated every 120 sec.

being using the same scheme (see Section 4.5). For being able to evaluate the performance for both very high and artificially lowered data yields, nodes are programmed to only generate new packets when there is free space in the local send queue. While this avoids packets being dropped due to queue overflows, this can result in the effective IPI being lower than the configured IPI.

4.7.2.1 Evaluation Methodology

The number of parent changes is computed from collected ground truth information. The number of missing packets, and thus the data yield, is calculated from included packet sequencing information $id_N(k)$. Here, a packet is not only missing if it was not received at the sink, but also when ground truth information is missing. As we generally find serial port communication very reliable, the amount of packets missing due to missing ground truth information is not significant.

Reconstructed information of a packet is only counted as correct if all three reconstructed components, *i.e.*, packet path, per-hop arrival order, and per-hop arrival times, are correct. The reconstructed path $\mathcal{N}_k$ of a packet k is correct if found nodes including their order matches with ground truth. The reconstructed path can prematurely end if packet k could not be fully reconstructed, but cannot contain additional elements or gaps. Reconstructed arrival order information is correct if the order of the queue index $qid_N(k)$ matches with ground truth for any node $N \in \mathcal{N}_k$. Lastly, extracted arrival time bounds are correct if it holds for all packets $k \in \mathcal{R}$ that the real arrival time $t_a(N, k)$ is within the reconstructed bounds, *i.e.*, $\forall N \in \mathcal{N}_k : t_a^l(N, k) \leq t_a(N, k) \leq t_a^u(N, k)$. As both reconstructed bounds and ground truth are measured on clocks of varying resolution and drift, this comparison allows for a bounded measurement error.

As presented in Section 4.6, the MNT algorithm also requires information of packets that arrived at the sink later than a packet k under investigation. We define the processing delay as the time distance between the arrival time $t_b(k)$ of packet k at the sink and the arrival time $t_b(v)$ of packet v at the sink, *i.e.*, $t_b(v) - t_b(v)$. Here, v is the latest arriving packet that is required for the analysis of packet k.

The uncertainty of reconstructed per-hop arrival time bounds is defined as the difference between upper and lower bounds, *i.e.*, $t_a^u(N, k) - t_a^l(N, k)$.

4.7.2.2 Discussion

This section details on how the IPI, the reactivity of the communication protocol used, the length of the routing paths, the time packets spent in the network, and the amount of lost packets influence the performance of the MNT algorithm. After firstly discussing the fraction of reconstructed packets, we will also detail on the processing delay and the uncertainty

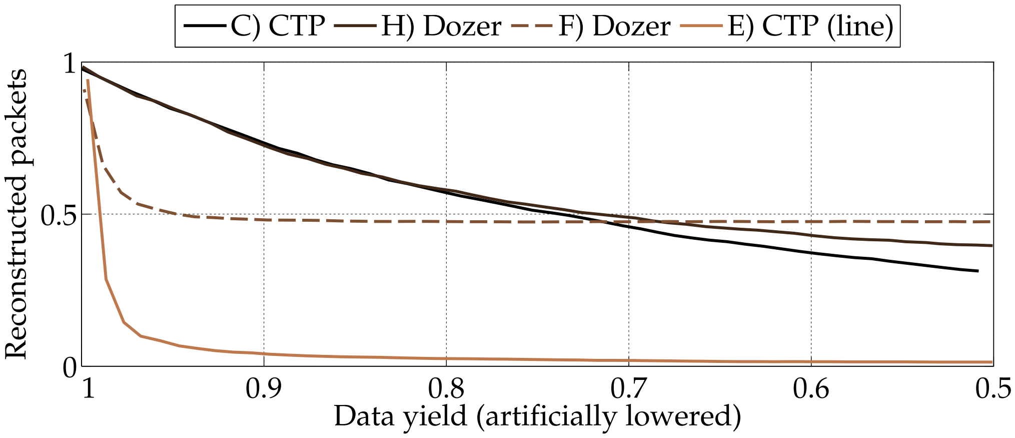


Fig. 4.4 Sensitivity of the reconstruction performance to a decreasing data yield. The fraction of packets received for which information can be safely reconstructed converges towards the fraction of packets that originate from single-hop neighbors of the sink.

of reconstructed per-hop arrival time bounds.

While the core principle of the MNT algorithm is to reconstruct information from correspondences created between multiple packets of different sources, an integral metric for understanding the reconstruction performance is the number of correspondences that are needed for being able to safely reconstruct the information of a packet. The number of packets that need to be considered during a worst-case path analysis (see Section 4.6.3.1) grows with the length of the routing path and the time a packet spent in the network. Similarly, the number of packets whose reconstruction relies on a particular packet also grows for longer routing paths and larger packet sojourn times. In consequence, the number of packets whose reconstruction might be affected by a particular parent change or a lost packet is also increasing.

Recent deployment reports, *e.g.*, [CMP+09, KWL+11], confirm that a data yield of ≥ 99.5% is achievable even in very challenging environments. A problem within the network must not necessarily be reflected by a large portion of packets being lost, but can also cause packets arriving at the sink with a larger delay. For instance, the Dozer implementation used in the PermaSense project does not deliberately drop packets, but retransmits each single packet until its reception was eventually acknowledged by the next hop.

Nevertheless, artificially lowering the data yield down to 50% allows us to study the fundamental limits of the MNT algorithm. Starting with the original traces obtained from tests C), E), F) and H), the data yield is consecutively lowered by randomly removing packets. The resulting reconstruction performance is shown in Figure 4.4. Here, we see the fraction of fully reconstructed packets converging towards a stable value which is the fraction of packets originating from one-hop neighbors of the sink. For example, only 1% of the packets in test E) originate from the single one-hop neighbor in the simulated 100-hop line topology. The decrease is almost linear for cases C) and H), the curves for the tests E) and F) show a steep decay at the beginning of the curve. Large routing paths in test E) and large packet sojourn times of up to four hours in test F) result in large numbers of correspondences needed for deciding if information can be reconstructed safely. In consequence, already a small number of lost packets can cause the information needed for the reconstruction of multiple other packets being lost in those extreme situations.

Regarding the time needed until all related packets also arrived at the sink, the results of tests A) to E) show a correlation between the processing delay and the chosen IPI. While almost all packets reached the sink as fast as possible in those five tests, the results presented for tests F) and G) include the effects of a fraction of packets staying in the network for several hours.

The uncertainty of reconstructed per-hop arrival time bounds is upper bound by the largest IPI used along the routing path. Uncertainties are significantly reduced when a reactive communication stack, *i.e.*, CTP Noe/LPL, is used.

Overall, the MNT algorithm has proven to achieve high reconstruction rates ≥ 91.2% in various configurations based on well-known CTP and Dozer protocols. During normal operation without congestion, information can be reconstructed quickly after a packet has been received at the sink. The uncertainty of reconstructed per-hop arrival time bounds is not affected by large packet sojourn times. Comparison with ground truth showed reconstructed information to be correct in all cases.

4.8 Making Real Network Dynamics Visible

This case study presents the application of the MNT algorithm to large data sets that originate from three real-world production deployments of the PermaSense project. After the initial deployment of the first system in 2008, the principle operation of the Dozer protocol used and the definition of packet application headers transmitted have not been modified ever since. This enables a coherent analysis of the complete data sets using the MNT algorithm.

Section 4.8.1 presents the reconstruction of information required by the MNT algorithm. The purpose of this step is to map the output of the specific system implementation to the inputs of the implementation-independent, generic system model (see Section 4.5). The accuracy of this transformation step is evaluated in testbed experiments. Results obtained from applying the MNT algorithm to the resulting traces are presented and discussed in Section 4.8.3. A simple usage example of reconstructed data is given in Section 4.8.4. Section 4.8.5 describes a more complex use case in which results from the MNT algorithm are used to evaluate fairness inside the network.

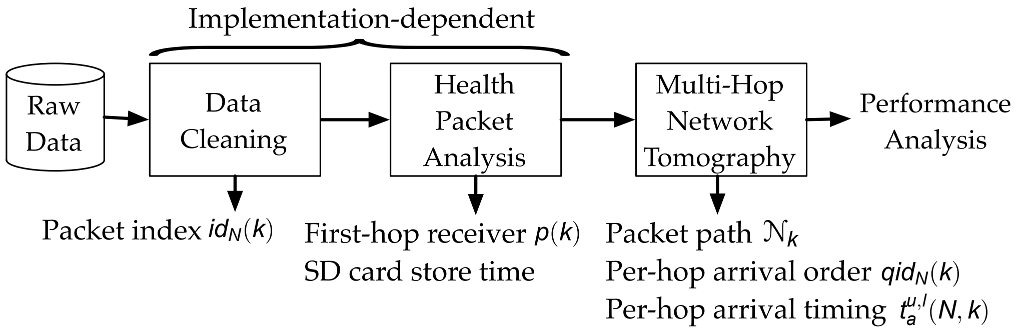

Implementation-dependent
Raw Data
Data Cleaning
Health Packet Analysis
Multi-Hop Network Tomography
Performance Analysis
Packet index $id_N(k)$
First-hop receiver $p(k)$
SD card store time
Packet path $\mathcal{N}_k$
Per-hop arrival order $qid_N(k)$
Per-hop arrival timing $t_a^{u,l}(N,k)$

Fig. 4.5 Implementation-specific data preparation. In order to conform with the inputs required by multi-hop network tomography, historic data from PermaSense deployments must be prepared using algorithms that are specific to the PermaSense system implementation.

4.8.1 Data Preparation Methodology

Input data required by the MNT algorithm is not implicitly included in PermaSense data sets and must therefore be reconstructed from other information during a multi-stage pre-processing step, see Figure 4.5:

- **Packet index** $id_N(k)$: Sequencing information provided by a sequence number transmitted that is reset to zero every 20 days in average must be converted to a monotonically increasing packet index $id_N(k)$. The algorithm used is able to address both intended, *i.e.*, the maximum counter value is reached, and unintended resets, *e.g.*, a node failure, of the sequence number.

- **SD card store time**: Most sensor nodes used in the PermaSense project are equipped with an SD card that buffers locally generated packets when the node is disconnected. Independent of the connection state, all locally generated packets are looped through this extra queue, see Figure 4.6. Received packet sequencing and periodically sampled queue size information are used for replaying queue operations and eventually reconstructing the SD card store time of individual packets.

- **First-hop receiver** $p(k)$: Nodes do not transmit the first-hop receiver of individual packets, but periodically sample the address of the current parent node. This requires the reconstruction of the time when a locally generated packet left the send queue, and thus was successfully transmitted over the radio. The estimation of this information is also based on known properties of the queue implementations used.

An heuristic is used for solving the problem of unknown forwarding traffic: Given that the send queue can only hold up to 20 locally generated packets, a locally generated packet must have departed latest when the next 20 locally generated packets have been added to the queue. For stable links, the first-hop receiver can be reconstructed with a high

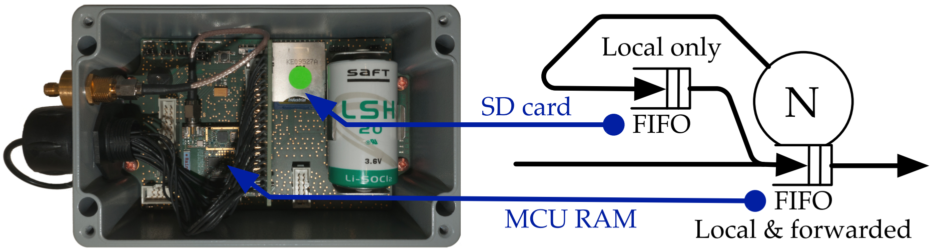


Fig. 4.6 PermaSense sensor node with two packet queues. Locally generated packets are looped through a second queue that is situated on a SD memory card. The packet generation time is no longer a valid proxy for the send queue arrival time, the MNT algorithm thus requires the duration for which a packet was stored on the SD card to be known.

confidence when combining the estimated send queue departure time with corresponding parent information.

The accuracy of the deployment data preparation test is verified in two testbed experiments of 110 hours duration in total. Similar to a deployed setting, these experiments involve six sensor nodes with attached SD cards, four sensor nodes that can only buffer messages in the RAM, and one sink node. Sensor nodes generate five packets every 2 minutes. Queue size counters and parent information are transmitted with every fifth packet. In contrast to the deployed setting, ground truth SD card store time and first-hop receiver are also transmitted with every packet. For forcing sensor nodes to buffer messages, the sink node is switched off eight times for a duration of 3 hours each. From 157,645 unique packets received, 99.4% of the packets pass the first stage of assigning a unique packet index $id_N(k)$. SD card store time and first-hop receiver information are reconstructed for 154,870 packets, remaining packets cannot be reconstructed due to missing context at the end of the trace. Reconstructed first-hop receiver information is correct in 99.96% of the cases. Subject to actually buffered packets only, the absolute mean error of the estimated SD card store time is 50 sec. This is to be expected given that queue sizes are only sampled every two minutes. Please notice that the logical ordering in which packets arrive is unaffected from this uncertainty, and that this uncertainty is specific to historic data originating from PermaSense deployments only.

4.8.2 Multi-Year Deployment Data Preparation

Multi-year deployment data originating from different hardware and software releases is not perfect, *e.g.*, still contains artifacts of problems that have been fixed over time. Therefore, the analysis of such data requires highest attention and care. Results of each intermediate step are separately verified by automated and manual sanity checks. The results of the implementation-specific data preparation for more than 270 million packets that originate from three sensor network deployments are shown in Table 4.3. More than 95.4% of the data from Matterhorn and

	MATTERHORN	JUNGFRAUJOCH	DIRRUHORN
Deployment Characteristics			
First deployment	July 2008	February 2009	August 2010
Current network size	31 nodes, 1 sink	29 nodes, 2 sinks	47 nodes, 3 sinks
Packets Received			
Years of operation	4.7	4.5	3.0
Unique	124,355,896	88,989,345	60,277,830
Duplicates	1,063,303	1,025,562	404,997
Packets After Data Cleaning			
Unique	98.8%	95.0%	96.9%
Packets After Health Packet Analysis			
Total unique	98.0%	83.1%	96.4%
Per node, min	94.7%	43.5%	89.8%
Per node, max	99.6%	99.7%	99.2%
Input of Multi-Hop Network Tomography			
Total	121,885,862	73,965,545	58,117,639
% of unique packets	96.0%	86.4%	95.4%

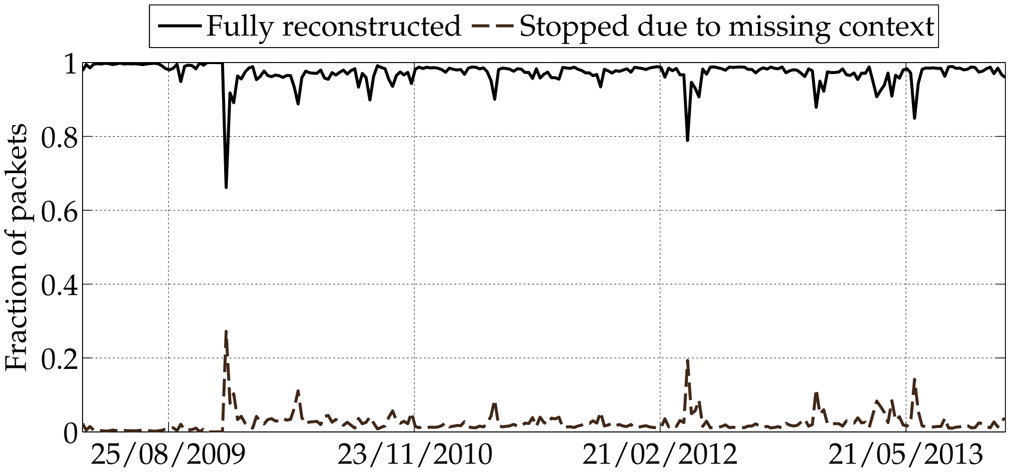
Tab. 4.3 Results of PermaSense-specific data preparation. Large portions of the input data have been reconstructed and are therefore ready for the multi-hop network tomography. Results of the Jungfraujoch deployment vary due to gaps in the traces used.

Dirruhorn deployments have been reconstructed and are therefore ready for multi-hop network tomography. The results for the Jungfraujoch deployment originate from truncated traces that are possibly caused by unintended manual deletion in the data repository. While the ends of the three incomplete traces show ca. 100,000 buffered packets each after a long phase of no connectivity, corresponding health packets for reconstructing how queues were flushed are missing.

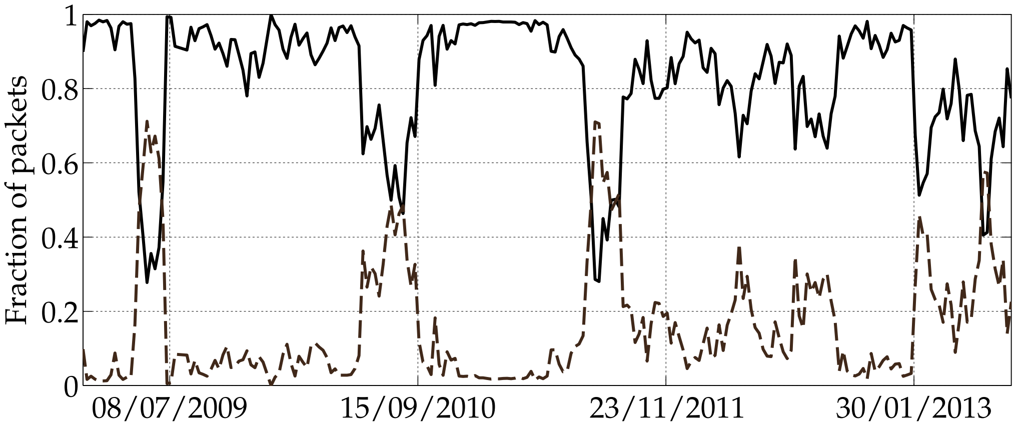
4.8.3 Multi-Deployment Network Tomography

The execution of the MNT algorithm requires the combined processing of the traces of all sensor nodes. To deal with the amounts of data found in this case study, data sets are split into week-long slices with approximately 300,000 packets per slice. Each slice contains packets that were generated during the corresponding week. Depending on the time packets spent in the network also packets of following weeks must be loaded for providing required context to the worst-case path analysis that is part of the MNT algorithm (see Section 4.6.3.1). For example, up to 10 times more data must be loaded when processing packets that were buffered in the network for multiple months at the Jungfraujoch site.

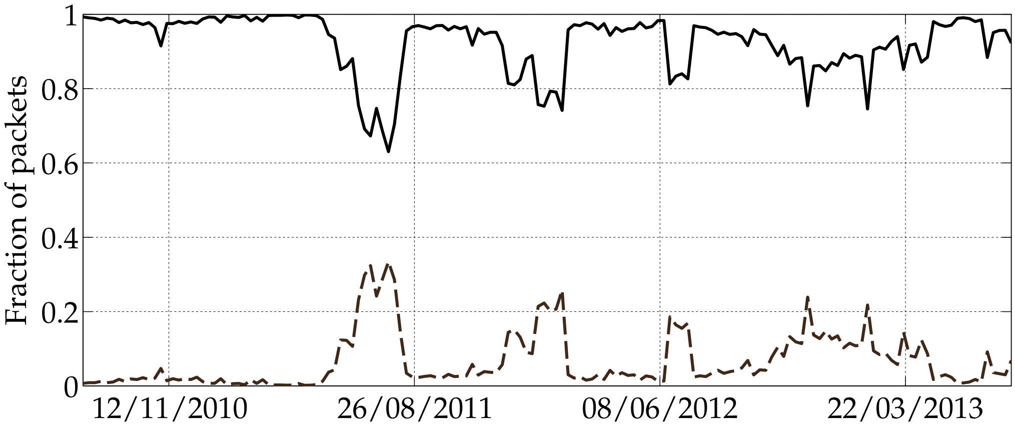
Multi-hop network tomography results are presented in Table 4.4 and Figure 4.7. More than 85.1% of the packets passed to the MNT algorithm



(a) Matterhorn



(b) Jungfraujoch



(c) Dirruhorn

Fig. 4.7 Weekly reconstruction performance when applying multi-hop network tomography to three deployment data sets. Lower performance is mostly caused by gaps in the data repository that were unintentionally introduced during the handling of data received. Additionally, a large number of packets and thus context needed for obtaining more complete results for the last weeks of the Jungfraujoch deployment are still buffered inside the network at the time of writing this thesis.

	MATTERHORN	JUNGFRAUJOCH	DIRRUHORN
	Total Packets		
Input	121,885,862	73,965,545	58,117,639
Reliable	97.4%	88.2%	96.0%
Full path	97.2%	85.1%	92.0%
	Fully Reconstructed Packets per Week-long Slice		
Min	66.1%	27.7%	63.1%
Max	100.0%	100.0%	99.8%

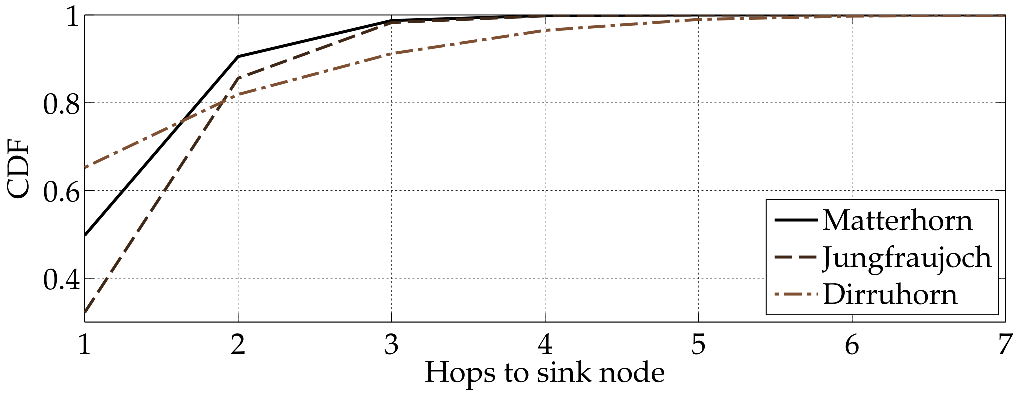
Tab. 4.4 Application of multi-hop network tomography to deployment data. The MNT algorithm is able to trace both original packets and independently traveling packet duplicates. Since it is not known at which hop a packet was actually duplicated, all packet instances are assumed to originate from the source. Duplicates are generally flagged and removed if this inaccuracy might harm the result of a particular analysis.

were fully reconstructed. Compared to the amount of total packets received from the network before the data preparation, this accounts to 95.3%, 70.7%, and 88.7% of packets being fully reconstructed.

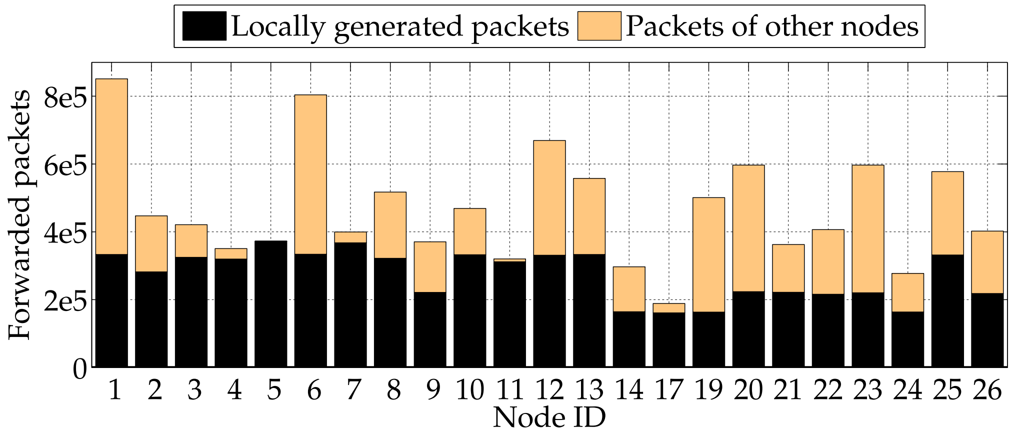
Lower results for individual weeks are caused by the MNT algorithm not being able to reconstruct all packets due to missing context, *i.e.*, human errors while handling data causing significant amounts of packets to be missing in the data repository. The number of errors virtually mirrors the number of packets that could not be reconstructed. Apart from this problem, the results at the end of Figure 4.7(b) are additionally rather low because packets and thus context needed for achieving better results are still buffered inside the network at the time of writing this thesis. Communication problems at the Jungfraujoch site cause packets to remain inside the network for weeks or even months [KWL+11]. Other potential problems, *e.g.*, sporadically lost packets and packet reordering, are only causing insignificant amounts of packets to be not reconstructable.

4.8.4 A First Glance Into the Network

Results of multi-hop network tomography are immediately usable for characterizing network operation. For instance, Figure 4.8(a) shows the distribution of the path length. Using all reconstructed packets of all three case study deployments as the input, this distribution is simply obtained by analyzing the per-packet path information that is provided by the MNT algorithm. Likewise, the traffic distribution as shown in Figure 4.8(b) can also be obtained from per-packet path information. Here, differences in the amount of locally generated packets are caused by varying sensor configurations. The minimal configuration of every node is to report its system health, *e.g.*, the battery voltage level, every two minutes.



(a) Path length distribution



(b) Per-node traffic of 10 weeks at Matterhorn deployment

Fig. 4.8 Detailed network characterization data can be easily obtained from reconstructed per-packet path, ordering and timing information.

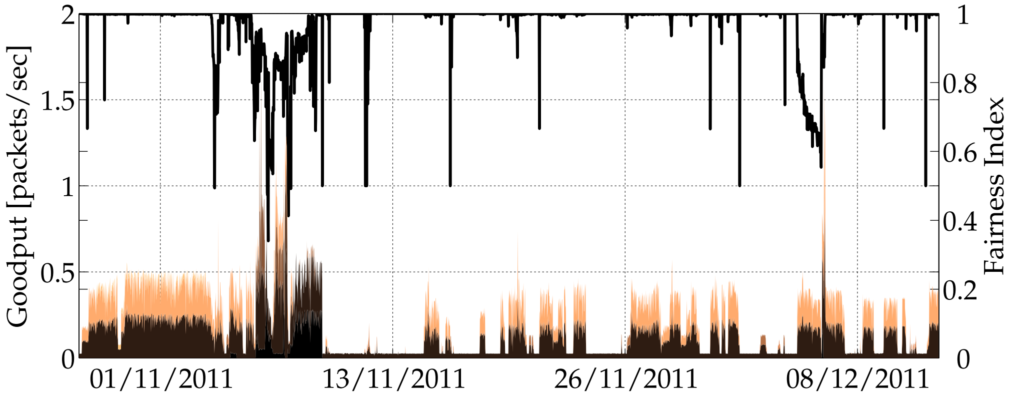


Fig. 4.9 Goodput (left-hand side axis) and fairness index (right-hand side axis) at node position 3 of the Dirruhorn deployment. This node is on average two hops away from the sink. It generates own data and receives packets from seven other nodes during the time period shown. Different packet sources are shown using distinct colors. The allocation of slots in the packet queue is no longer fair when the goodput is higher than 0.5 packets per second.

4.8.5 Performance Analysis Inside the Network

Apart from rather simple network characterization tasks, reconstructed data can also help to answer more complex questions on the performance of a network. The probably most powerful application of reconstructed data is to combine reconstructed packet path and per-hop arrival times for deriving bounds on per-hop packet sojourn times. Per-hop packet sojourn times are particularly interesting as an increase in the packet delay serves as an important proxy for detecting problems inside a network, *e.g.*, problems with the wireless channel or no free buffer capacity at the next hop towards the sink. A complete health monitoring system that is based on reconstructed per-hop packet sojourn times is presented in Chapter 5.

A smaller, yet not less interesting example for performance analysis inside a network is the evaluation of fairness. As missing fairness leads to large amounts of packets being delayed by other traffic, fairness is a valuable indicator for improving the network beyond the scope of protocol self-recovery mechanisms, *e.g.*, by modifying statically set protocol parameters.

Following the definition of Jain *et al.* [JCH84], we calculate the Fairness Index that is defined as the fraction of users being treated fair. An allocation is fair, if all users receive a share that is fair in terms of their own demand and the demand of other users. Results obtained for 20 minutes long slices are shown in Figure 4.9. While the demand is defined as the number of packets that are currently buffered in both queues of a sensor node, the allocated share is defined as the number of packets that a particular child node transmitted during a slice of 20 minutes length.

In our analysis, we find the sink node to always be fair to its children. This is to be expected as the sink node is not duty-cycled and also does not suffer from queue size constraints when data is immediately forwarded to the base station PC. However, fairness is no longer given at intermediate nodes when the amount of packets received is higher than 0.5 packets per second over a longer time period. It is to investigate if this behavior needs to be improved, *e.g.*, by modifying the current queue allocation scheme.

4.9 Broader Applicability and Limitations

With tree-based routing protocols being very popular in real-world deployments, the MAC-independent MNT algorithm can potentially be used in many scenarios. Still, the assumption of all nodes generating data might not fit to some applications. While sensor nodes that only forward packets could be modified to cooperate, *i.e.*, modify passing first-hop receiver information to hide themselves, the existence of sole forwarders will unavoidably introduce inaccuracies. Additionally, the assumption of routing paths being rather stable might render the MNT algorithm not applicable for systems in which path changes occur very frequent, *e.g.*, because of nodes selecting the next hop in a round-robin fashion.

4.10 Conclusions

This chapter presented multi-hop network tomography (MNT), a novel, non-intrusive algorithm for the reconstruction of the path, the per-hop arrival order and the per-hop arrival time of individual packets. The results of extensive testbed runs of two state-of-the-art data collection protocols, *i.e.*, CTP and Dozer, verified that information can be reconstructed with a great confidence. The application in a deployment context has been proven to be feasible in a case study that involved more than 270 million packets from three real-world sensor network deployments.

As one concrete example for further applications of multi-hop network tomography, the following Chapter 5 presents *Hybrid Monitoring*, a system for monitoring network health. Here, information reconstructed and provided by MNT significantly reduces the amount of extra information that is needed to fulfill this task.

5

Hybrid Network Health Monitoring

Network health monitoring is a fundamental, yet well-studied problem in wireless sensing systems. Starting with systems that periodically transmit health information [RCK+05], researchers soon tried to reduce the amount of additional traffic by applying in-network aggregation techniques [RB06, LMZL11] or even moving the failure detection process inside the network [MLML12]. Though even completely passive systems such as PAD [LLL10] have been proposed, this sole analysis of unmodified application traffic suffers from a low detection accuracy and limited coverage. For example, PAD is not able to distinguish if a sensor node is disconnected and thus not trying to send packets, or actually trying but not able to get packets successfully transmitted.

Multi-hop network tomography (see previous Chapter 4), itself a passive method, enables the creation of *Hybrid Monitoring*, a minimally active system that is able to deliver detailed insights from inside the network while at the same time only adding the minimal overhead of a single extra bit to every packet.

5.1 Introduction

Looking at recent projects, we can currently observe a continuous growth in network size [MHL+09] and in the usage horizon of WSN deployments. Furthermore, anticipated future applications of unprecedented societal and economical interest, *e.g.*, the monitoring of natural hazards, will require those networks to fulfill highest reliability requirements. Networks of growing size and importance can no longer be monitored manually, but require automated systems that ideally warn a network operator even before a yet undetected, lingering misbehavior [KBM+09] would eventually result in a fatal incident, *e.g.*, a node completely stopping its service.

In this chapter, we propose *Hybrid Monitoring*, a novel WSN health monitoring system that utilizes passively reconstructed packet information while only adding minimal extra information required for mitigating the aforementioned problem to every packet. The design of this system is based on the observation that a number of failure events, *e.g.*, failed packet transmission attempts, a congested channel, or full packet queues towards the sink, share the common characteristic of affected packets to wait longer inside the network. Resulting delays range from a few seconds to several minutes. Initially being interested in inferring the actual number of failure events occurred solely from passively reconstructed per-hop timing information (see previous Chapter 4), we found that accuracy and coverage are significantly improved when asking for minimal collaboration from inside the network. Concretely, our measurements prove that adding one bit per packet is sufficient for resolving the majority of situations in which a completely passive system would heavily over or underestimate the metric analyzed.

Implementing the semantics of the so called "problem bit" requires only minimal modifications on the sensor nodes. *Hybrid Monitoring* does not inject additional packets into the network, but needs only one extra bit to be attached to every packet that is generated by the existing sensor network application. A sensor node sets the problem bit of a locally generated packet if at least one failure event has occurred since the generation of the previous packet. Here, a failure event refers to any behavior that requires a packet transmission to be repeated or postponed despite of the node being connected to the network. Subject to further refinement w.r.t. the concrete protocol used, failure events are unacknowledged packet transmissions, the radio not being ready for transmission, and the queue at the next hop being full.

Problem bit information from inside the network is then combined with passively reconstructed path and per-hop timing information after packets have been received at the sink. Here, the main principle is to analyze the times that individual packets spent at a node. The resulting health metric is the actual number of failure events that occurred between the generation of two subsequent data packets. Exemplary scenarios that contribute to a rise in this metric are a lossy channel, an unbalanced routing tree, timing problems on the node, and even hardware issues. The number of failure events is an early diagnosis metric, *i.e.*, failures can be detected even before a protocol eventually starts to drop packets.

The contribution of this chapter is as follows:

- We propose *Hybrid Monitoring*, a novel health monitoring system that complements a passive health monitoring method, *i.e.*, the analysis of passively reconstructed timing information, with a minimally active component. Sensor nodes are only required to set the problem bit according to very simple semantics.

Retrieved information are useful for identifying lossy links and traffic bottlenecks, and to generally monitor the performance of a system over time. Our approach minimizes communication and computation overheads. As it can be completely integrated into an existing application, *Hybrid Monitoring* is suited for continuous operation. If needed, *Hybrid Monitoring* can also be combined with heavier debugging components that may be automatically activated based on the number of detected failure events.

- The algorithmic framework of *Hybrid Monitoring* is presented based on a formal model of a data collection application.
- We implemented our system on real nodes on top of the Dozer [BvRW07] ultra low-power data collection protocol. Extensive experiments on testbeds of up to 96 nodes in size show that the number of occurred failure events can be estimated with a high confidence. More than 90% of the decisions are done correctly when the number of occurred failure events is used as the input of a runtime monitoring application that eventually triggers further action, *e.g.*, notifies a network operator.

The remainder of this chapter is structured as follows: After presenting related work in Section 5.2, the general problem that this work is trying to tackle is presented in Section 5.3. Section 5.4 gives a high-level description of the proposed system. Assumptions made are summarized in a formal model of a data collection system that is shown in Section 5.5, a detailed description of algorithms used is presented in Section 5.6. The results of our evaluation on real hardware are shown in Section 5.7, Section 5.8 discusses the broader applicability and the limitations of the presented system. Section 5.9 concludes this chapter.

5.2 Related Work

After initial lessons learned [SMP+04, LBV06], a growing tool support has helped to make WSNs nowadays being ready for long-term missions at scale. The debugging of a system whose state is distributed over many devices being a very challenging task, many efforts have especially focused on this particular problem. As a result, various methods, *e.g.*, simulation [LLWC03, ODE+06], testbeds [HKWW06, LFZ+13], co-located sniffing hardware [CPMW08], in-network debugging facilities [YSSW07, CAS+08], and in-network state loggers [LHZ+06, RM09], are now available for supporting developers during the whole development lifecycle of a WSN.

Once a system is ready for production, informed network operation again requires distributed state to be accessible. While debugging facilities that were used during the development are often too heavyweight or even not applicable at the final deployment site, only the most important runtime information is continuously collected after deployment. A common design found in many solutions, *e.g.*, [ZGE02, RCK+05, RB06, LMZL11, CKL+12], is to collect information within the network, and to offload the analysis process to the network sink. Variations include the usage of in-network aggregation techniques, *e.g.*, [ZGE02, RB06], or to run parts of the analysis already inside the network [RB06, MLML12]. Actively sending state can be completely avoided when a higher uncertainty can be tolerated [LLL10].

The system proposed in this work extends multi-hop network tomography (see previous Chapter 4), a passive method for the reconstruction of the packet path, the per-hop ordering, and the per-hop timing information of individual packets, with an as non-intrusive as possible active component that runs inside the network. Only very simple semantics need to be implemented on the nodes, only one bit of extra information is added to every packet.

When comparing this new approach to Sympathy [RCK+05], we first find a considerable amount of traffic being saved by the passive reconstruction of path information. While Sympathy is more flexible when it comes to freely choosing an reporting interval, *i.e.*, health data is not coupled with application traffic, our approach is automatically achieving a higher temporal resolution when the application is sending more data. In contrast, an active scheme such as Sympathy might even have to reduce its operation in that case.

Still, certain information that is reported by Sympathy, *e.g.*, the node uptime, is not covered by our approach. While not having this information certainly leaves more uncertainty in the analysis, there is also no other choice but active transmission when such information is required.

Agnostic Diagnosis [LMZL11] is another system that is very similar to Sympathy with the main difference of making use of so called correlation graphs instead of utilizing a simple decision tree. While transmitting more information, *i.e.*, periodically sampled counter values, certainly increases confidence, our solution is working on a comparable level of detail, *i.e.*, not only considering fatal errors, but already inefficient behavior, *e.g.*, packet retransmissions.

Having the purpose of efficiently transmitting spatial distributions of scalar values, *e.g.*, energy levels, eScan [ZGE02] can be considered rather orthogonal or even complementary to this work. If certain scalar metrics, *e.g.*, the node uptime, the current battery voltage, and the humidity within the enclosure, are required, using the presented in-network aggregation scheme would certainly help to reduce the overhead that is added to the network.

Probably closest to this work is PAD [LLL10], a passive health monitoring approach. Here, the root causes for sensor and node failures are inferred using a probabilistic approach. While the whole decision process is based on only sporadically inserted path marks that are then used to generate decision graphs, the detection performance of this approach is limited.

Hybrid Monitoring distinguishes itself from PAD and other systems mentioned by using proven to be correct (see previous Chapter 4) path and per-hop timing information of individual packets as its input. While the detection of sensor faults, *e.g.*, [GZH09], is orthogonal to this work, *Hybrid Monitoring* is also not focussing on node liveness, but on the detection of inefficient operation that already degrades system performance and may also lead to a fatal error if not discovered beforehand. Although technically being an active system, the overhead of *Hybrid Monitoring* is closest to that of passive systems, *e.g.*, PAD [LLL10].

5.3 Inferring State From Minimal Information

The utility of a sensor network application is threatened when measurement locations are temporarily not served due to node failures. The timely replacement of a node after the fact can be difficult when long and expensive travel is needed. Instead, an early diagnosis can reduce maintenance costs when degraded but not yet failed components can already be replaced during scheduled maintenance periods.

In this work, we want to explore the minimal amount of transferred information needed for making informed decisions with confidence. As the available computational power is an order of magnitude higher as well as cheaper after packets have been received at the sink, we are especially interested in solutions that ask for the minimum amount of communication and computation to take place inside the network. In this context, we want to find answers to the following questions: How much state is already known implicitly, *e.g.*, from the timing behavior of a system? How large is the gain from adding extra in-band information to an initially passive method?

5.4 Hybrid Monitoring

Various reasons can increase the waiting time of a packet at a node. Radio and wireless channel need to become available, sending over a lossy channel can require multiple transmission attempts. Exceptionally long waiting times can occur when a node is disconnected from the network. Events mentioned share the characteristic that any of their occurrences results in an increased end-to-end packet delay. Measuring the end-to-end delay of every individual packet is a common practice in data collection

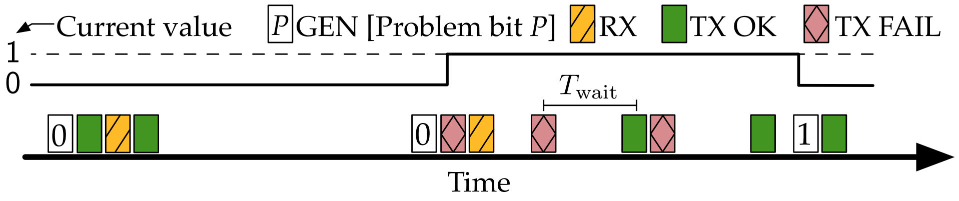


Fig. 5.1 Exemplary scenario with several packets being generated, received from other nodes and eventually being transmitted to a parent. The local variable that corresponds to the problem bit is set to 1 on the first occurrence of a failure event. It is again cleared after its current value has been copied into the most recently generated packet.

applications.

Looking at the exemplary situation in Figure 5.1, we can see three packets that are generated at the node under observation. Two more packets are received from other nodes. If a transmission failed, the next transmission is assumed to happen a fixed time T_{wait} later.

While we can see three transmission errors happening between the generation of the second and the third locally generated packet in Figure 5.1, the question now is how we can reconstruct the number of occurred failure events after packets have been received at the sink? The proposed solution for this problem is depicted in Figure 5.2. Assuming that we are able to split the measured end-to-end packet delay at the sink into individual contributions per node, this information can then be further decomposed and used for inferring the number of occurred failure events.

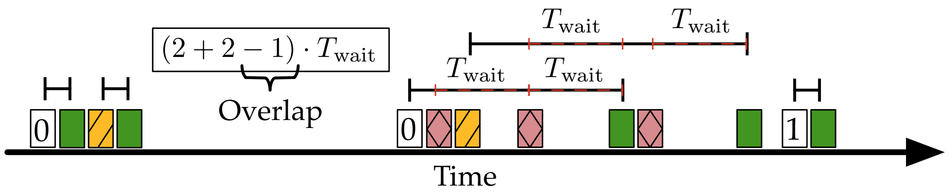


Fig. 5.2 Packet timing on a node under analysis. The additional delays are caused by packet transmission errors (see Figure 5.1) that force subsequent packets to wait. Given access to per-hop timing information, we can reconstruct the number of occurred failure events by decomposing the individual per-hop sojourn times of the involved packets.

In practice, several challenges need to be solved before the number of failure events can be estimated with confidence. The end-to-end delay first of all being composed out of the contributions of all nodes along the packet path, the contribution of each single node can again be split into disjoint events. Delays caused by a node being disconnected from the network must be filtered out in order to not be labeled as failure events. This separation is in particular made difficult by the fact that connected and disconnected operation can be arbitrarily interleaved.

In order to solve this problem, we propose that sensor nodes have to set the so called problem bit on the occurrence of a failure event during connected operation. As illustrated in Figure 5.1, the current value of the corresponding variable gets copied and then cleared every time when a new packet is generated.

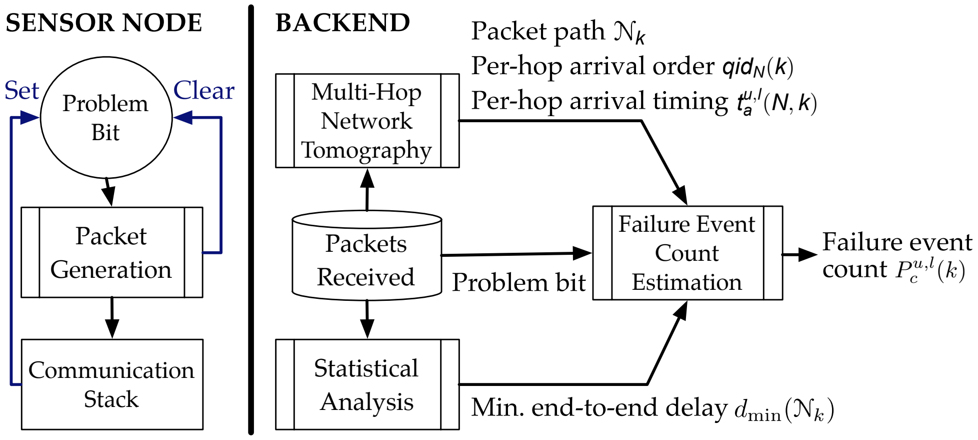

SENSOR NODE
BACKEND
Set
Clear
Problem Bit
Packet Generation
Communication Stack
Multi-Hop Network Tomography
Packets Received
Statistical Analysis
Packet path $\mathcal{N}_k$
Per-hop arrival order $qid_N(k)$
Per-hop arrival timing $t_a^{u,l}(N,k)$
Problem bit
Failure Event Count Estimation
Failure event count $P_c^{u,l}(k)$
Min. end-to-end delay $d_{\min}(\mathcal{N}_k)$

Fig. 5.3 Information from minimal collaboration inside the network is augmented after packets have been received at the sink.

The overall design of *Hybrid Monitoring* is shown in Figure 5.3. Each node maintains a local variable that corresponds to the problem bit. As long as a node is connected to a parent, the problem bit is set on any occurrence of a failure event. Every time when a new packet is generated, the current value of the problem bit variable is written into that packet and then cleared. Apart from helping to not mistake disconnected operation for a high number of failure events, the semantics used also support the later analysis by mitigating the effects of imperfect timing information.

The simple semantics inside the sensor network are complemented by a set of algorithms that are executed outside the network after packets have been received at the sink. First, multi-hop network tomography (see previous Chapter 4) is used for reconstructing the packet path and per-hop timing information of individual packets. While obtained worst-case waiting times from this method can be very pessimistic, the final decision on the number of occurred failure events is made more robust by adding results from a statistical analysis of packet end-to-end delays as a second data source. Here, continuously updated estimates of the minimum end-to-end delay, *i.e.*, the sojourn time of a packet without waiting times due to disconnects or failure events, of every path are used for estimating the waiting time of delayed packets.

5.5 System Model

Multi-hop data collection. We assume a data collection application in which sensor nodes generate data that is then send to a sink. Sensor nodes and sink maintain a single FIFO queue that is used for both locally

Packet Application Headers	
$o(k)$	Source node network address
$\tilde{t}_g(k)$	Estimated packet generation time
$P(k)$	Problem bit
Added on Arrival at the Sink	
$t_b(k)$	Arrival time at the sink
From Post-processed Packet Headers	
$id_N(k)$	Packet generation index reflecting the correct order of generation for packets originating from a node N
From Multi-Hop Network Tomography	
$\mathcal{N}_k$	Packet path
$qid_N(k)$	Queue index reflecting the order of arrival at the queue of node N
$t_a^{u,l}(N,k)$	Upper and lower bounds on the arrival time of k at node N
$t_d^{u,l}(N,k)$	Upper and lower bounds on the departure time of k from node N
From Analysis	
$\Delta^{u,l}(k)$	Upper and lower bounds on the accuracy of the estimated packet generation time $\tilde{t}_g(k)$
$t_g^{u,l}(k)$	Upper and lower bounds on the unknown packet generation time $t_g(k)$
$d_{\min}(\mathcal{N}_k)$	Minimum end-to-end delay of path $\mathcal{N}_k$, partially constant
Implementation-specific Parameters	
T_{wait}	Waiting time before the next transmission attempt after a failure event

Tab. 5.1 Overview of system model variables

generated packets and forwarded packets. Communication is based on a tree-based multi-hop routing protocol. The network operation is subject to phenomena that are common to wireless sensor networks, *i.e.*, packet loss and packet duplication. The network sink is the only device that has access to a synchronized clock. Inside the network time is measured on clocks of limited resolution and with a bounded drift.

System model variables. Table 5.1 gives an overview of the variables used in this model. Packets are uniquely identified using a numerical index that reflects the order of arrival at the sink. For every received packet k, we first of all assume to have access to the source address $o(k)$, the problem bit $P(k)$, and the packet path $\mathcal{N}_k$. Further information can be grouped as follows:

- **Timing information:** We define $\tilde{t}_g(k)$ as the estimate of the unknown packet generation time $t_g(k)$. The error of this estimate is bounded by $t_g^l(k) = \tilde{t}_g(k) - \Delta^l(k) \leq t_g(k) \leq \tilde{t}_g(k) + \Delta^u(k) = t_g^u(k)$. Here, $\Delta^l(k)$ and $\Delta^u(k)$ denote the upper and lower bounds on the error of the time-stamping mechanism used. The arrival time at the sink $t_b(k)$ is assumed to be measured on a perfect clock. For each hop N that the packet visited, *i.e.*, $N \in \mathcal{N}_k$, we define $t_a^u(N,k)$ and $t_a^l(N,k)$ as the upper and lower bounds on the unknown arrival time $t_a(N,k)$ of packet k at node N. Likewise, $t_d^u(N,k)$ and $t_d^l(N,k)$ denote the upper and lower bounds on the not accurately known departure time $t_d(N, k)$ of packet k at node N. It holds that $t_a^l(N, k) \leq t_a(N, k) \leq t_a^u(N, k)$ and $t_d^l(N, k) \leq t_d(N, k) \leq t_d^u(N, k)$, respectively.

- **Sequencing information:** Apart from the index k that is used to identify packets in this model, we also assume the order of which packets at a node N were generated to be reflected by the packet generation index $id_N(k)$. The packet generation index of a packet k is assumed to be by one larger than the packet generation index of the packet that was generated at the same source $o(k)$ immediately before k. Similarly, the queue index $qid_N(k)$ reflects the order in which packets arrived at the queue of a node N. A packet k that arrived at a node N has a higher queue index than any other packet that arrived at N before k. Please note that the packet generation index $id_N(k)$ not only allows to order packets from the same source N, but can also be used to detect packet loss. In contrast, the queue index $qid_N(k)$ allows to order packets from any source that visited node N, but can not be used for detecting missing packets.

Passive reconstruction. Packet path $\mathcal{N}_k$, queue index $qid_N(k)$, and per-hop timing information $t_a^{u,l}(N, k)$ and $t_d^{u,l}(N, k)$ are not implicitly transferred as part of every packet, but reconstructed using multi-hop network tomography. For us being able to use this passive reconstruction method, our system must also fulfill the requirements of multi-hop network tomography. Most notably, multi-hop network tomography assumes that all sensor nodes are generating data. Furthermore, it is assumed that the address of the first receiver of a packet, *i.e.*, the current parent of the source node at the moment of transmission, is transmitted as part of the packet. Further assumptions made including case studies in which multi-hop network tomography is applied to systems running CTP [GFJ+09] and Dozer [BvRW07] can be found in the previous Chapter 4.

Protocol operation. Lastly, we make the following further assumptions regarding the protocol operation:

- Sensor nodes are aware of their connectivity state. A sensor node is in a connected operation state as long as it can regularly communicate with its designated parent node, *e.g.*, successfully receive beacons and packet acknowledgements. The transition to a disconnected state is based on certain rules defined by the protocol, *e.g.*, the expiration of a timer or an error counter hitting a threshold. The phase of no communication prior to the disconnect decision still counts as connected operation.

- Certain errors, *e.g.*, a lost beacon message, a missed packet acknowledgement, or the radio being in the wrong state, can require the transmission of a packet to be postponed or repeated. In such a case, we assume that the next transmission attempt is made after waiting for a implementation-specific time T_{wait}. T_{wait} can include a jitter $J \ll T_{\text{wait}}$.

- The minimum end-to-end delay $d_{\text{min}}(\mathcal{N}_k)$ of a path $\mathcal{N}_k$ is defined as the time that a packet needs for traversing $\mathcal{N}_k$ in the absence of both transmission errors and disconnected operation. $d_{\text{min}}(\mathcal{N}_k)$ is assumed to be constant with a bounded deviation smaller than a defined δ for at least small numbers of subsequent transmissions along the same path.

5.6 From a Single Bit to a Scalar Value

For every received packet k with a set problem bit, *i.e.*, $P(k) \equiv 1$, we want to determine the upper and lower bounds $P_c^u(k)$ and $P_c^l(k)$ on the unknown number of failure events that occurred at the source of packet k between the generation of packet k and the generation of the immediate predecessor of k. In contrast to the introductory example that was presented in Section 5.4, we cannot access perfect per-hop timing information in the real case. Instead, our algorithms are required to be robust to imperfect timing information.

Estimating the unknown number of occurred failure events involves two main steps. First, we need to determine the set $\mathcal{W}$ of all packets that may have been waiting for transmission at node $o(k)$ within the temporal scope of $P(k) \equiv 1$, *i.e.*, before the generation of packet k and after the generation of the immediate predecessor of k. In other words, the set $\mathcal{W}$ contains all packets whose end-to-end delay may be affected by the failure events that caused the problem bit $P(k)$ to be set. The number of occurred failure events is then estimated by analyzing the timing information of all packets $w \in \mathcal{W}$.

The analysis of the waiting time of a single packet k at a node N is presented in Section 5.6.1. An approach that constructs the number of occurred failure events as the sum of non-overlapping, individual contributions of multiple packets is then shown in Section 5.6.2. Section 5.6.3 presents the algorithm that is used for continuously estimating the current minimum end-to-end delay $d_{\text{min}}(\mathcal{N}_k)$ of a path $\mathcal{N}_k$.

For clarity and brevity, it is assumed that packet path and per-hop timing information of all packets can be reconstructed. In practice, this information is missing for a small fraction of packets that are not "reliable" and can thus not be used in a multi-hop network tomography. In consequence, this small amount of packets can not be used in the following analysis.

5.6.1 Packet Waiting Time Analysis

Given a packet k with a problem bit $P(k) \equiv 1$, the analysis starts with determining the set $\mathcal{W}$ of packets whose timing may be affected by the failure events that caused the problem bit $P(k)$ to be set. As the temporal scope of the problem bit is defined as the time period that starts after the generation of the previous packet and ends before the generation of the current packet k, we first need to find the predecessor f of packet k. This packet f has the two properties that it was generated at the same node as k and that its packet generation index $id_{o(k)}(f)$ must be exactly one lower than the packet generation index $id_{o(k)}(k)$ of packet k:

$$\begin{aligned} f := \arg\max_x id_{o(k)}(x) \text{ for all } x : o(x) \equiv o(k) \\ \wedge\, id_{o(k)}(x) \equiv id_{o(k)}(k) - 1 \end{aligned} \tag{5.1}$$

The temporal scope of the problem bit can now be expressed by the precise packet generation times of f and k: $t_{p,s}(k) = t_g(f) < t < t_g(k) = t_{p,e}(k)$. As precise packet generation times are not known in practice, we must resort to the usage of upper and lower bounds $t_g^{u,l}(f)$ and $t_g^{u,l}(k)$ that reflect the uncertainties of the obtained packet generation times. We define the upper and lower bounds on the beginning and end of the time interval $]t_{p,s}(k), t_{p,e}(k)[$ for which the problem bit $P(k)$ is valid as follows:

$$\begin{aligned} t_{p,s}^l(k) &:= t_g^l(f) & t_{p,e}^l(k) &:= t_g^l(k) \\ t_{p,s}^u(k) &:= t_g^u(f) & t_{p,e}^u(k) &:= t_g^u(k) \end{aligned}$$

The set $\mathcal{W}$ of all packets that may be affected by the failure events covered by $P(k)$ is then build by selecting all packets that (i) arrived at the node $N := o(k)$ before packet k and (ii) may have departed after the generation of packet f:

$$\mathcal{W} := \{w \mid qid_N(w) < qid_N(k) \wedge t_d^u(N, w) > t_g^l(f)\} \tag{5.2}$$

For each packet $w \in \mathcal{W}$, we then determine the minimum and maximum number of T_{wait} long intervals that w can have spent at node $o(k)$ within $]t_{p,s}(k), t_{p,e}(k)[$. To achieve that, we first determine the shortest and longest time interval that packet w can have spent at $o(k)$ within $]t_{p,s}(k), t_{p,e}(k)[$. The shortest time interval $s_{\min}(w)$ is based on the assumption that w arrived as late as possible while again leaving as early as possible:

$$s_{\min}(w) := \min\left(t_d^l(w), t_{p,e}^l(k)\right) - \max\left(t_a^u(w), t_{p,s}^u(k)\right) \tag{5.3}$$

Likewise, the calculation of $s_{\max}(w)$ is based on the counter assumption of arriving as early as possible while leaving as late as possible:

$$s_{\max}(w) := \min\left(t_d^u(w), t_{p,e}^u(k)\right) - \max\left(t_a^l(w), t_{p,s}^l(k)\right) \tag{5.4}$$

As the longest possible time interval $s_{\max}(w)$ is often pessimistic, the final calculation of the number of T_{wait} long intervals also includes an estimation of the maximum added delay that packet w can have experienced along the complete path due to failure events. The maximum added delay due to failure events $e_{\max}(w)$ is the difference of the actual end-to-end delay of packet w and the minimum end-to-end delay $d_{\min}(\mathcal{N}_w)$ along the path $\mathcal{N}_w$ that packet w travelled along:

$$e_{\max}(w) := t_b(w) - \tilde{t}_g(w) - d_{\min}(\mathcal{N}_w) \tag{5.5}$$

The calculation of the minimum and maximum number of T_{wait} long intervals that packet w spent at node $o(k)$ finally also accounts for a jitter $J \ll T_{\text{wait}}$:

$$v_{\min} := \left\lfloor \frac{s_{\min}(w) + J}{T_{\text{wait}}} \right\rfloor, v_{\max} := \left\lfloor \frac{\min(s_{\max}(w) + J, e_{\max}(w))}{T_{\text{wait}}} \right\rfloor \tag{5.6}$$

5.6.2 Failure Event Count Estimation

In order to estimate the total number of failure events that occurred at node $N := o(k)$ within the time interval $]t_{p,s}(k), t_{p,e}(k)[$, we must combine the waiting times of all packets $w \in \mathcal{W}$. Here, the challenge is that the time intervals in which different packets can have waited at node N may partially or fully overlap, *e.g.*, when a single failure event causes multiple packets to wait.

The algorithm used for determining the upper and lower bounds $P_c^{u,l}(k)$ on the unknown number of occurred failure events at node N is presented in Algorithm 5.1. Given a packet k, bounds are immediately set to zero if the problem bit of k is not set (line 2). Else, the procedure starts with determining packet f that was generated at node N directly before packet k (line 3). Next, the set of packets $\mathcal{W}$ that might be affected by a problem at N during the time interval $]t_{p,s}(k), t_{p,e}(k)[$ is determined (line 5).

If the first analyzed packet arrived at N before packet f was generated and also departed from N after packet k was generated (line 8), the estimation of the number of occurred failure events is hindered by both the best-case and the worst-case waiting time being set too pessimistic, *i.e.*, as the temporal distance of packets f and k. In this case, we set the number of failure events to at least one and to at most the highest number of T_{wait} long intervals that fit into the time interval between the generation of f and k (line 9).

Otherwise, all packets $w \in \mathcal{W}$ are processed in the order of arrival at node N (line 13). As packet are assumed to leave a node in the same order as they arrived, packets that arrived earlier are analyzed before later arriving and thus also later departing packets.

Newly introduced variables r^u and r^l have the purpose of avoiding waiting times to be counted more than once. r^u and r^l are first initialized with the packet generation times of packet f (line 11), and then

Algorithm 5.1: Algorithm for estimating the number of occurred failure events

Input : Packet k

Output: Upper and lower bounds $P_c^{u,l}(k)$ on the number of failure events at node $N := o(k)$ between the generation of packet k and its immediate predecessor f

```
1  $P_c^u(k) \leftarrow 0; P_c^l(k) \leftarrow 0; \mathcal{S} \leftarrow \{\}$ ;
2  if $\neg P(k)$ then return;
3  $f \leftarrow \arg\max_x id_N(x)$ for all $x : o(x) \equiv N$
4          $\wedge\; id_N(x) \equiv id_N(k) - 1$ ;
5  $\mathcal{W} \leftarrow \{w \mid qid_N(w) < qid_N(k) \wedge t_d^u(N,w) > t_g^l(f)\}$ ;
6  if $\mathcal{W} \equiv \emptyset$ then return;
7  $w \leftarrow \arg\min_x qid_N(x)$ for all $x : x \in \mathcal{W}$ ;
8  if $t_a^u(N,w) < t_g^l(f) \wedge t_d^l(N,w) > t_g^u(k)$ then
9      $P_c^l(k) \leftarrow 1; P_c^u(k) \leftarrow \lfloor (t_g^u(k) - t_g^l(f))/T_{\text{wait}} \rfloor$ ;
10     return;
11 $r^l \leftarrow t_g^l(f); r^u \leftarrow t_g^u(f)$ ;
12 while true do
13     $w \leftarrow \arg\min_x qid_N(x)$ for all $x : x \in \mathcal{W} \setminus \mathcal{S}$ ;
14     $r^l \leftarrow \max(t_a^l(N,w), r^l)$ ;
15     $r^u \leftarrow \max(t_a^u(N,w), r^u)$ ;
16     $v_{\min}(w), v_{\max}(w) \leftarrow$ AnalyzePacket($w, r^l, r^u, k$)
17     $P_c^l(k) \leftarrow P_c^l(k) + v_{\min}(w)$ ;
18     $P_c^u(k) \leftarrow P_c^u(k) + v_{\max}(w)$ ;
19     $r^l \leftarrow r^l + v_{\min}(w) \cdot T_{\text{wait}}; r^u \leftarrow r^u + v_{\max}(w) \cdot T_{\text{wait}}$ ;
20     $\mathcal{S} \leftarrow \mathcal{S} \cup \{w\}$ ;
21     if $\mathcal{W} \setminus \mathcal{S} \equiv \emptyset$ then break ;
22 end
23 if $P_c^l(k) \equiv 0$ then $P_c^l(k) \leftarrow 1$;
24 if $P_c^u(k) \equiv 0$ then $P_c^u(k) \leftarrow 1$;
```

continuously incremented when either the arrival of the next analyzed packet is not overlapping with the waiting times of the previous packet w (lines 14 and 15), or when at least one new waiting interval of length T_{wait} was detected (line 19).

The actual estimation of the minimum and maximum waiting time of the currently analyzed packet w (line 16) is located in Algorithm 5.2. The beginning of the time interval in which this algorithm looks for waiting times is defined by r^u and r^l, the end of the time interval is given by the generation time of packet k. The upper and lower bounds on the total number of occurred failure events $P_c^{u,l}(k)$ are determined by building the sum of the individual, non-overlapping contributions (lines 17 and 18)

Processing the next packet w continues until all packets $w \in \mathcal{W}$ have been analyzed (line 21). Already analyzed packets are members of both the set $\mathcal{W}$ and the newly introduced set of already seen packets $\mathcal{S}$.

If all packets $w \in \mathcal{W}$ have been processed without any failure event being found, *e.g.*, because of packets being lost, the problem bit $P(k)$ being set allows us to infer that there must have been at least one occurrence of a failure event (lines 23 and 24).

Algorithm 5.2: AnalyzePacket

```
Input  : Analyzed packet $w$, start of analysis time window $r^{u,l}$, packet $k$
Output: Minimum and maximum number $v_{\min}(w)$ and $v_{\max}(w)$ of $T_{\text{wait}}$ long time
        intervals within $r^{u,l}$ and the generation of packet $k$
begin
    $s_{\min}(w) \leftarrow \min(t_d^l(N, w), t_g^l(k)) - r^u$ ;
    $v_{\min}(w) \leftarrow \lfloor (s_{\min}(w) + J)/T_{\text{wait}} \rfloor$ ;
    $s_{\max}(w) \leftarrow \min(t_d^u(N, w), t_g^u(k)) - r^l$ ;
    $e_{\max}(w) \leftarrow t_b(w) - \tilde{t}_g(w) - d_{\min}(\mathcal{N}_w)$ ;
    $v_{\max}(w) \leftarrow \lfloor \min(s_{\max}(w) + J, e_{\max}(w))/T_{\text{wait}} \rfloor$ ;
end
```

Depending on the concrete communication protocol used, implementation-specific details can require further refinements of the procedure that is described in Algorithm 5.1. For example, a sensor node running Dozer [BvRW07] needs approximately 60 seconds for negotiating a connection with a new parent node. In consequence, estimation results are improved when those 60 seconds are excluded from being accounted as waiting times due to a failure event.

5.6.3 Estimation of Minimal End-to-End Packet Delay

So far, the minimum end-to-end delay $d_{\min}(\mathcal{N}_k)$ of the path $\mathcal{N}_k$ was assumed to be known. The algorithm that is used for continuously updating the estimates of the path-dependent minimum end-to-end delay is shown in Algorithm 5.3. Here, the main assumption is that the minimum end-to-end delay equals to the end-to-end delay of a packet that has not been delayed inside the network due to a failure event or a disconnect. While that information is not perfectly known, the two requirements are verified using a worst-case analysis and a heuristic approach.

Algorithm 5.3: UpdatePathDelayEstimate

```
Input  : Newly arrived packet $k$
Output: Updated estimate of the minimum end-to-end delay $d_{\min}(\mathcal{N}_k)$ on path $\mathcal{N}_k$
begin
    foreach $N \in \mathcal{N}_k$ do
        $\mathcal{P} \leftarrow \{x \mid o(x) \equiv N \wedge t_g^u(x) > t_a^l(N, k)$
                $\wedge\, t_g^l(x) < t_d^u(N, k) \wedge P(x) \equiv 1\}$ ;
        if $\mathcal{P} \not\equiv \emptyset$ then  break ;
    end
    if $\mathcal{P} \equiv \emptyset$ then
        if $t_b(k) - \tilde{t}_g(k) < d_{min}(\mathcal{N}_k) + \delta$ then
            $d_{\min}(\mathcal{N}_k) := t_b(k) - \tilde{t}_g(k)$ ;
        end
    end
end
```

Before the first packet of a path $\mathcal{N}_k$ has been received, the minimum end-to-end delay $d_{\min}(\mathcal{N}_k)$ of this path is initialized with a protocol-dependent parameter, *e.g.*, the product of the path length $|\mathcal{N}_k|$ and a defined maximum processing time of a packet $T_{\text{queue,max}}$.

Given a newly arrived packet k, the algorithm starts with traversing all nodes N along the path $\mathcal{N}_k$ of packet k (line 2). At each node N, it is then analyzed if k may have been at node N during a problem. This is done by first identifying all packets $\mathcal{P}$ that were generated at node N during the maximum time that k can have spent at N, and then determining if the problem bit of any of those packets was set (line 3). Packet k can only be used to set a new minimum end-to-end delay $d_{\min}(\mathcal{N}_k)$ if all involved packets had the problem bit not set (line 7).

The exclusion of k being delayed due to a disconnect is based on the comparison of the end-to-end delay of packet k with the existing value of $d_{\min}(\mathcal{N}_k)$. Packet k is assumed to not have been delayed by a disconnect if its end-to-end delay is at most by a δ higher than the former value (line 8).

5.7 Multi-Testbed Evaluation

In this section, we present the results of extensive testbed experiments that we conducted in order to (i) quantify the gain in confidence when nodes are actively transmitting one extra bit per packet, and (ii) to evaluate the performance of *Hybrid Monitoring* when compared to ground truth.

All testbed experiments are based on an implementation of *Hybrid Monitoring* on top of the TinyOS implementation of Dozer [BvRW07]. Dozer is a state-of-the-art low-power data collection protocol that is used in multiple long-term deployments [KWL+11]. Apart from implementing the semantics of the problem bit and adding instrumentation code for outputting ground truth information over the serial port of the node, no further modifications were done to the standard implementation of Dozer.

Three sets of nodes are used in this evaluation: 96 Tmote Sky (MSP430+CC2420 radio) nodes of the TWIST [HKWW06] testbed allow us to study the performance of the *Hybrid Monitoring* in a large network. Tests with two kinds of radio hardware are possible on the FlockLab [LFZ+13] testbed. Here, we are using 30 Tmote Sky nodes and 30 Tinynode 184 (MSP430+SX1211) nodes. Because the 868 MHz radio of the Tinynode 184 is operating on a different band than local Wi-Fi networks, traces obtained using Tinynode hardware include less failure events due to packet loss.

The integration of *Hybrid Monitoring* into the Dozer data collection protocol is described in Section 5.7.1. The four test configurations used and the characteristics of the data obtained are presented in Section 5.7.2. In Section 5.7.3, the performance of *Hybrid Monitoring* is compared with a completely passive system that does not make use of the problem bit.

Comparisons with ground truth show significant gains in confidence when the problem bit is used. The performance of *Hybrid Monitoring* in the context of an runtime monitoring application that triggers further action, *e.g.*, notifies a network operator, based on a pre-defined threshold value is presented in Section 5.7.4.

5.7.1 Integration Into the Dozer Data Collection Protocol

The operation of the Dozer protocol is divided into rounds of a fixed length T_{round}, a round can be further divided into a number of slots. The slot assignment is based on a local schedule that includes communication with the parent node, communication with child nodes, a phase in which new child nodes can join, and the execution of local processing tasks. The local schedule of a node is modified every time when the node switches to another parent, or when a new child node connected. Additionally, a sensor node can decide to shift the position of the local processing slots, *e.g.*, because the old position is likely to interfere with a communication slot.

Waiting packets are transmitted during the slots that are reserved for communication with the parent node. If a packet transmission fails, the next transmission is postponed to the following round. Thus, the waiting time T_{wait} is $T_{\text{wait}} := T_{\text{round}}$. For the remainder of this evaluation, T_{round} is set to 30 seconds with a jitter of ±2 seconds.

The problem bit is set on the occurrence of at least one of the following failure events that share the common characteristic of postponing the next packet transmission by $T_{\text{wait}} :=$ 30 seconds plus jitter:

- **Missed beacon:** A beacon message from the parent did not arrive within the expected time. In consequence, Dozer is not trying to upload packets.
- **Missed acknowledgement:** A packet acknowledgement from the parent did not arrive within the expected time. In consequence, the packet at the head of the queue must be retransmitted.
- **Busy radio:** The radio was not ready for a packet transmission. This can happen when the radio is occupied by another task, *e.g.*, currently handling a connection request of another node.
- **Full parent queue:** None or not all locally queued packets could be transmitted because of a full queue at the parent node.

Apart from setting the problem bit, the listed events also cause a log entry to be generated. Further log entries are generated on the generation of a packet, on the arrival of packet from another node, each time when the node tries to transmit a message to its parent, when the node disconnects from its parent, and when the node connected to a new parent. Log

	N	D	IPI	H	DY	RECEIVED (GROUND TRUTH) TOTAL	 $P(k) \equiv 1$	 F. EVENTS
					Tmote Sky			
A)	96	17 hours	120 sec	5	99.5%	49,305	15,772	23,536
B)	26	14 hours	120 sec	4	99.7%	10,654	3,768	6,823
C)	30	9 hours	60 sec	4	99.8%	14,940	6,750	10,496
					Tinynode			
D)	30	24 hours	120 sec	5	99.9%	20,613	7,015	11,053

(a) Characterization of the input data

	AFTER TOMOGRAPHY		WAIT TIME UNCERT.		MIN. DELAY ERROR		
	TOTAL	$P(k) \equiv 1$	$p_{0.9}$	$p_{0.98}$	MEAN	$p_{0.85}$	$p_{0.98}$
				Tmote Sky			
A)	96.5%	96.6%	98 sec	120 sec	12 sec	27 sec	44 sec
B)	92.8%	93.2%	120 sec	178 sec	11 sec	27 sec	44 sec
C)	89.7%	87.2%	78 sec	118 sec	11 sec	26 sec	45 sec
				Tinynode			
D)	95.8%	95.8%	111 sec	123 sec	9 sec	25 sec	41 sec

(b) Results of multi-hop network tomography

Tab. 5.2 Characterization of data from testbed experiments A) to D) before and after multi-hop network tomography.

entries are immediately timestamped using the local node clock and then put into the memory of the node. The outputting of buffered log entries over the serial port is done at the next slot that has been assigned for local processing tasks. Likewise, the generation of new packets is also executed in such a slot.

5.7.2 Data Preparation Results

The test configurations used and the characteristics of results obtained are shown in Table 5.2. For each experiment A) to D), Table 5.2 lists the number of nodes (N), the duration (D) of the test in hours, the inter-packet interval (IPI) used, the height of the routing tree (H), the achieved data yield (DY), the total number of received packets, the number of received packets with a set problem bit, and the number of failure events that occurred during the experiment. Chosen test durations are mainly driven by the availability of testbed resources. Periodic packet generation is no requirement of our method, however, periodic packet generation allows us to better analyze the statistical properties of the received timing information.

Furthermore, Table 5.2 also lists the amount of packets for which information can be reconstructed using multi-hop network tomography.

Numbers obtained are within the results of previous experiments, see Section 4.7. The reconstruction performance of multi-hop network tomography is independent of the value of the problem bit $P(k)$. This is as expected given that problem bit information is not used by the multi-hop network tomography algorithm. The waiting time uncertainty is defined as the difference between the maximum (worst-case) and the minimum (best-case) time that a packet can have spent at a node. Results are described by the 90th and the 98th percentile. Found uncertainties are also in line with previous results.

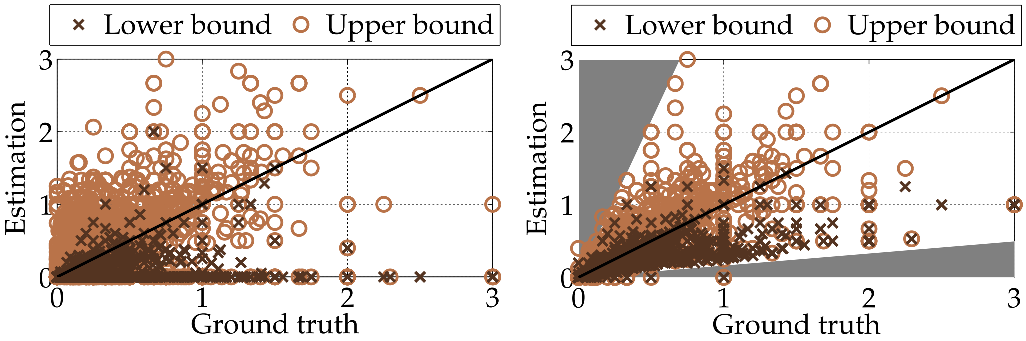
The last three columns of Table 5.2 quantify the results of Algorithm 5.3 that is used for estimating the minimum end-to-end delay of individual packets paths. Results listed include the mean value, the 85th percentile, and the 98th percentile. Here, mainly non-determinisms found in the Dozer protocol operation, *e.g.*, slots for local processing tasks being moved from the end to the beginning of a round, introduce an estimation error that is larger than T_{wait} for approximately one fifth of the packets.

5.7.3 Comparison With Completely Passive System

For being able to quantify the gain by adding one bit to every packet, we created completely passive versions of Algorithm 5.1 and Algorithm 5.3. The resulting system does not make use of the problem bit. While *Hybrid Monitoring* uses the problem bit for deciding if the number of failure events is definitely zero or at least one (Algorithm 5.1, lines 23 and 24), the completely passive system lacks of this information. In consequence, the completely passive system is executing the waiting time analysis (Algorithm 5.2) for all packets received.

In the following, we compare obtained upper and lower bounds on the number of failure events $P_c^{u,l}(k)$ with ground truth. The relationship between estimated bounds and the actual number of failure events in experiment A) is shown in Figure 5.4. Shown results have been aggregated based on information from a connection table that has been obtained from ground truth. Each entry of this table corresponds to a connection between a child node and a parent node, each entry of the connection table is represented by a circle and a cross. Shown values are calculated by dividing the number of failure events by the number of packets that were successfully submitted during the corresponding time interval. Estimated values are plotted over ground truth, ideally all data points should be located on the diagonal line that represents the identity function.

First of all, we can see that worst-case and best-case estimations are separated by the identity function in both figures Figure 5.4(a) and Figure 5.4(b). However, solutions obtained from the completely passive system are significantly higher or lower, respectively, than ground truth in a large number of cases.



(a) Completely passive system **(b)** Nodes transmit problem bit

Fig. 5.4 Comparison of a completely passive system with *Hybrid Monitoring*. Results are based on the traces of testbed experiment A), see Table 5.2. Both plots show the failure event estimations $P_c^{u,l}(k)$ that have been obtained by the corresponding system over ground truth. Each data point corresponds to one connection between a child node and its parent. As the connection duration varies, shown results have been normalized by dividing the number of occurred and estimated failure events by the number of total packets that were transmitted during a connection. Shaded areas in Figure 5.4(b) highlight the gain in terms of improved estimation results over the completely passive system when *Hybrid Monitoring* is used.

This observation can be further quantified using Pearson's correlation coefficient. In Figure 5.5, the amount of problem bit information used for estimating the number of failure events $P_c^{u,l}(k)$ is alternated between 0% and 100%. For example, 10% means that the problem bit $P(k)$ of 10% of the packets is used. In turn, the problem bit $P(k)$ of the remaining packets is ignored in this configuration. Modified versions of Algorithm 5.1 and Algorithm 5.3 randomly select the actual packets of which problem bit information is being used. For each analyzed fraction between 0% and 100%, the estimation of the number of failure events is repeated 30 times with varying random number generator seeds being used. Estimated upper and lower bounds are then correlated with ground truth. For every analyzed fraction of problem bit information used, Figure 5.5 shows the mean and the total range of the 30 runs that have been conducted for every fraction. Pearson's correlation coefficient grows almost linearly with the fraction of problem bit information used. While we obtain values $\rho_{\text{upper}} = 0.92$ and $\rho_{\text{lower}} = 0.86$ when comparing ground truth with results from the *Hybrid Monitoring* system, *i.e.*, 100% problem bit information being used, significantly lower values $\rho_{\text{upper}} = 0.75$ and $\rho_{\text{lower}} = 0.56$ are retrieved when correlating ground truth with results from the completely passive system, *i.e.*, 0% problem bit information being used.

The gain of using the problem bit in *Hybrid Monitoring* can be further highlighted when using data from both systems as input for an anticipated alarming application. Here, we consider a runtime filter that triggers an action, *e.g.*, informs a network operator, if the number of failure events that occurred at a node within a moving time window of Δ length exceeds a pre-defined threshold value.

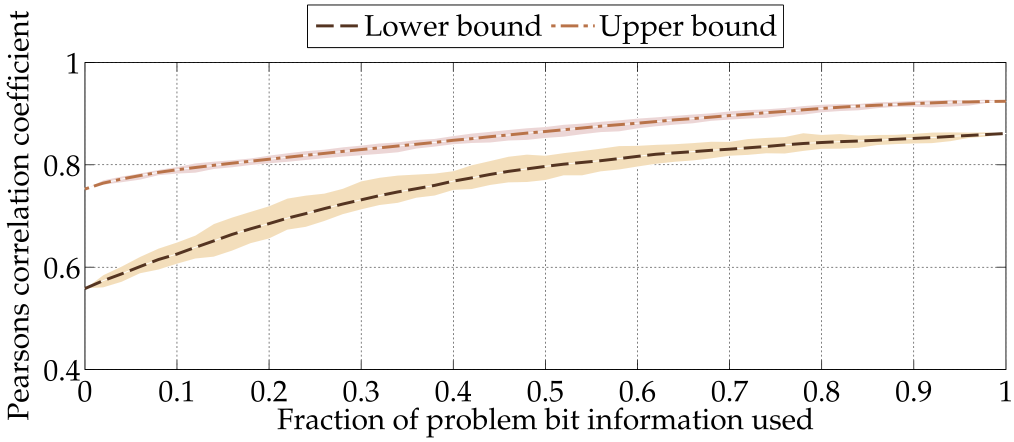


Fig. 5.5 Correlation between the estimated number of failure events and ground truth. The fraction of problem bit information used for obtaining failure event count estimations is alternated between 0% and 100%. For example, a fraction of 10% corresponds to the problem bit information of 10% of all packets being used. Packets are selected randomly, the analysis has been carried out 30 times over the complete range. Shown lines denote the mean value over 30 runs for every fraction, shaded areas denote the range of results obtained.

To achieve this, the first step is to calculate the occurred and estimated number of failure events in intervals of Δ length. Corresponding results of generating one aggregated value per node every $\Delta := 30\,\text{min}$ are shown in Figure 5.6. Results that originate from *Hybrid Monitoring* are denoted by the two lines immediately above and below the line that corresponds to ground truth. The performance of the completely passive system is denoted by the outer boundaries of the two shaded areas. The shaded areas itself therefore correspond to the gain of using *Hybrid Monitoring* instead of the completely passive system. Here, we can again see results of the completely passive system being significantly too high or too low, respectively, in many cases.

In consequence, the performance of the final alarming application is also significantly lower when compared to the performance of *Hybrid Monitoring*. The fraction of correct decisions, *i.e.*, the inferred decision from estimated values is correct, false positives, *i.e.*, an alarm despite the correct value being below the set threshold, and false negative, *i.e.*, missed alarms, are shown in Figure 5.7. Again, shaded areas denote the improvement by adding one bit of extra information per packet.

5.7.4 Runtime Monitoring

The already introduced runtime monitoring scenario is now further evaluated on all four testbed experiments A) to D). Results for two configurations of a fixed evaluation interval Δ are shown in Table 5.3(a) and Table 5.3(b). The threshold value for deciding if an alarm must be raised is again alternated between 0 and 20 failure events within $\Delta = 10\,\text{min}$, and between 0 and 60 failure events within $\Delta = 30\,\text{min}$,

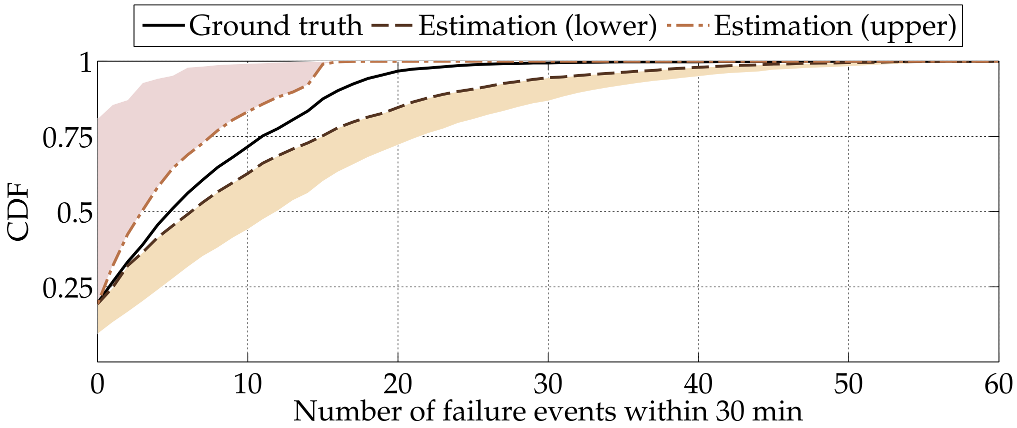


Fig. 5.6 Sum of occurred and estimated failure events in 30 minute long intervals. Results that originate from *Hybrid Monitoring* are denoted by the two lines immediately above and below the line that corresponds to ground truth. The performance of the completely passive system is denoted by the outer boundaries of the two shaded areas. The shaded areas itself therefore represent the gain in estimation accuracy by using *Hybrid Monitoring* instead of a completely passive system.

respectively. Decisions are made based on the estimated upper bound of failure events that occurred within Δ. Values obtained when basing decisions on estimated lower bounds are comparably well.

Decisions made are correct in $\geq$ 90.7% of all cases. The quality of decisions made is the better for the more packets for which multi-hop network tomography could return results. Ranking testbed experiments either by the fraction of correct decisions made or the fraction of packets for which additional information could be reconstructed (see Table 5.2) yields the exactly same order. This behavior is to be expected given that only packets that passed multi-hop network tomography are used as input for estimating the number of failure events that occurred within Δ.

5.8 Broader Applicability and Limitations

The presented approach is based on the assumption that certain events inside the network contribute a measurable, additional delay to the end-to-end delay of waiting packets. For example, Dozer [BvRW07] waits for approximately 30 sec before starting a new transmission attempt. A measurable penalty is potentially also added by other slotted approaches, *e.g.*, slotted programming [FW10], that may be set to postpone every retransmission attempt to the next slot. Here, it is to expect that slot lengths of a few seconds or even less are sufficient in combination with an accurate enough time-stamping mechanism. In a broader scope, existing data collection protocols may only require small modifications in order to increase the delay penalty that is added after a transmission failure. For instance, one possibility to adapt CTP [GFJ+09] would be to increase the firing interval of the retransmission timer.

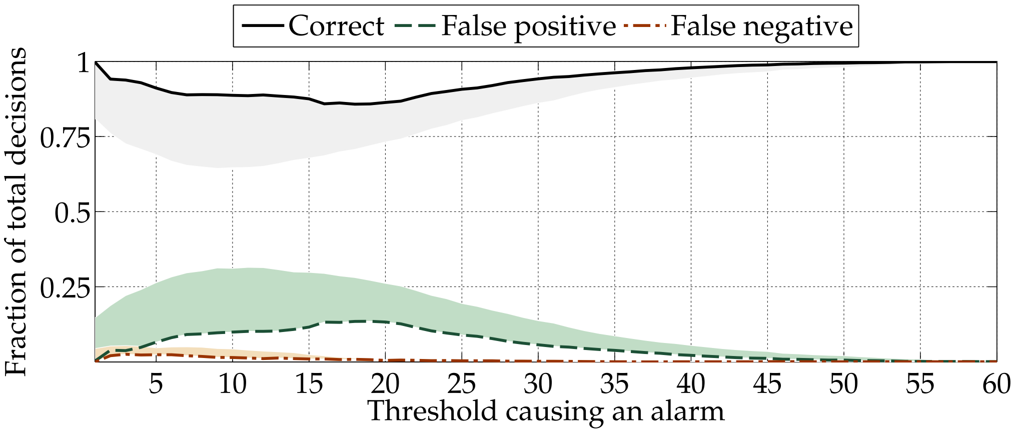


Fig. 5.7 Comparison of runtime monitoring performance. The triggering of an alarm is based on a pre-defined threshold value that is incremented from zero to up to 60 failure events within 30 min. For every 30 min long interval and varying thresholds, it is first evaluated if the number of actually occurred failure events reached the threshold value. Likewise, it is evaluated if the upper estimation of the number of failure events reached the threshold value. A decision is correct if both results lead to the same decision, *i.e.*, alarm or no alarm. The shaded areas again represent the gain in estimation accuracy by using *Hybrid Monitoring* instead of a completely passive system.

			DECISIONS	
	# INTERVALS	MEAN CORRECT	MIN. CORRECT	MAX. CORRECT
A)	9,841	93.5%	84.4%	99.7%
B)	2,084	91.6%	80.8%	99.8%
C)	1,485	90.7%	84.0%	99.8%
D)	4,112	92.4%	80.3%	99.8%

(a) Up to 20 failure events within $\Delta = 10$ min long intervals

			DECISIONS	
	# INTERVALS	MEAN CORRECT	MIN. CORRECT	MAX. CORRECT
A)	3,319	94.1%	85.7%	99.9%
B)	697	92.5%	79.9%	100.0%
C)	505	91.0%	82.7%	99.4%
D)	1,393	93.2%	81.7%	99.9%

(b) Up to 60 failure events within $\Delta = 30$ min long intervals

Tab. 5.3 Runtime monitoring application that raises an alarm if the number of failure events within Δ long intervals reached a threshold value. Starting with zero failure events within Δ, the threshold value is again incremented up to the maximum possible number of failure events within Δ. A decision is correct if both the real number of failure events and the estimated number of failure events lead to the same decision, alarm or no alarm. Results of all threshold configurations are aggregated and presented by their minimum, mean and maximum values.

Apart from requiring transmission failures to cause a measurable delay, a system must also conform to the formal model of multi-hop network tomography (see Section 4.5 of the previous chapter). Exemplary systems that have been verified to be conforming to this model are CTP on top of LPL and Dozer. Systems that are not compatible to multi-hop network tomography are for instance protocols that randomly select the receiver of every packet. Likewise, multi-hop network tomography and in consequence the presented approach in this chapter are also not applicable to systems that use flooding as their communication primitive, *e.g.*, the low-power wireless bus [FZMT12].

5.9 Conclusions

We presented *Hybrid Monitoring*, a novel health monitoring system that complements a passive health monitoring method, *i.e.*, the analysis of passively reconstructed timing information, with a minimally active component. Adding one bit of extra information to every packet is already sufficient for mitigating the uncertainties that have the largest impact on the accuracy of the passive health monitoring method. As a result, the number of failure events that occurred between the generation of two subsequent packets can be estimated with a high confidence. Based on experiments on testbeds of up to 96 nodes, we find the amount of correct decisions made by a runtime monitoring application to be significantly improved when one more bit of information is available to the estimation process.

6

Conclusions

This thesis presented algorithms and systems that have the goal of establishing wireless data collection systems as dependable and precise scientific instruments. Covered aspects of this thesis are (i) the mitigation of artifacts that have been introduced by the wireless sensing system, (ii) giving guarantees on the quality of proven to be correct data samples, (iii) the interactive, visual inspection of very large data sets, (iv) making usually hidden interactions that occur inside a multi-hop network observable, and (v) the resource-saving monitoring of wireless sensing systems at runtime. The performance of presented algorithms and systems is backed by a strong empirical evidence, *i.e.*, results from testbed experiments, results from the application to multi-year traces from several real-world deployments, and results from the long-term operation in a production environment.

Results obtained are not strictly limited to low-power wireless sensor networks, but can also be applied to a broader class of multi-hop communication systems. For instance, packet duplication and packet loss are common phenomena of wireless communication. Likewise, path changes are also unavoidable in any system that employs a path-based routing scheme. While more available resources certainly allow for more capabilities, *e.g.*, in-network duplicate suppression and end-to-end packet acknowledgements, to be integrated into a communication protocol, it remains a case by case decision if the overall scenario, *e.g.*, delay-tolerant networking [Fal03], renders such efforts as practical.

Apart from solving selected problems in existing systems, models and algorithms that have been presented in this thesis also yield learnings for the design of future systems. Here, the vision is to design for data quality and observability from early on. While information on the further data processing steps was yet missing when initial systems were built, knowledge that has now been gained will help to build less complex, yet better performing systems.

The work presented can be extended in several directions. Concerning data cleaning, an open question is if and in which situations there is a relation between data anomalies that are caused by the communication layer and data anomalies that are caused by the sensing layer. In order to decide on the validity of sensor data received, solving this question will require the underlying geophysical phenomena to be modeled. A potential application of this research could be the detection of partially faulty, *e.g.*, due to component aging, sensor nodes and sensing equipment. Looking at multi-hop network tomography, an interesting question is if machine learning techniques can detect suspicious system behavior, *e.g.*, certain anomalies, when given the results of multi-hop network tomography as an input. The potential future of the proposed hybrid monitoring system is first of all to add the problem bit to deployed systems, but also to research if further system state can be retrieved in a similar fashion.

Bibliography

[AHS06] K. Aberer, M. Hauswirth, and A. Salehi. A middleware for fast and flexible sensor network deployment. In *Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB '06)*, pages 1199–1202, 2006.

[ASC+10] K. Aberer, S. Sathe, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Thiele. OpenSense: Open community driven sensing of environment. In *Proc. 1st Int'l Workshop on GeoStreaming (IWGS '10)*, pages 39–42, 2010.

[BBF+11] J. Beutel, B. Buchli, F. Ferrari, M. Keller, L. Thiele, and M. Zimmerling. X-Sense: Sensing in extreme environments. In *Proc. Conf. on Design, Automation and Test in Europe (DATE '11)*, pages 1–6, 2011.

[BBK00] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into web server design. *Computer Networks*, 33(1):1–16, 2000.

[BDF+99] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. Spatio-temporal retrieval with RasDaMan. In *Proc. 25th Int'l Conf. on Very Large Data Bases (VLDB '99)*, pages 746–749, 1999.

[BGH+09] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes. In *Proc. 7th Int'l Conf. Information Processing Sensor Networks (IPSN '09)*, pages 265–276, 2009.

[BIS+08] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. SensorScope: Out-of-the-box environmental monitoring. In *Proc. 7th Int'l Conf. Information Processing Sensor Networks (IPSN '08)*, pages 332–343, 2008.

[BISV08] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proc. 6th ACM Conf. Embedded Networked Sensor Systems (SenSys '08)*, pages 43–56, 2008.

[BvRW07] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *Proc. 6th Int'l Conf. Information Processing Sensor Networks (IPSN '07)*, pages 450–459, 2007.

[BYL+11] B. Buchli, M. Yuecel, R. Lim, T. Gsell, and J. Beutel. Feature-rich experimentation for WSN design space exploration. In *Proc. 10th Int'l Conf. on Information Processing in Sensor Networks (IPSN '11)*, pages 115–116, 2011.

[CAS+08] Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, and L. Luo. Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks. In *Proc. 6th ACM Conf. Embedded Networked Sensor Systems (SenSys '08)*, pages 85–98, 2008.

[CCD+11] M. Ceriotti, M. Corrà, L. D'Orazio, R. Doriguzzi, D. Facchin, S. Guna, G. P. Jesi, R. L. Cigno, L. Mottola, A. L. Murphy, et al. Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *Proc. 10th Int'l Conf. Information Processing Sensor Networks (IPSN '11)*, pages 187–198, 2011.

[CHINY02] A. Coates, A. Hero III, R. Nowak, and B. Yu. Internet tomography. *IEEE Signal Processing Magazine*, 19(3):47–65, 2002.

[CKL+12] Y.-H. Chiang, M. Keller, R. Lim, P. Huang, and J. Beutel. Light-weight network health monitoring. In *Proc. 11th Int'l Conf. Information Processing Sensor Networks (IPSN '12)*, pages 109–110, 2012.

[CMP+09] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *Proc. 8th Int'l Conf. Information Processing Sensor Networks (IPSN '09)*, pages 277–288, 2009.

[CPMW08] B. Chen, G. Peterson, G. Mainland, and M. Welsh. LiveNet: Using passive monitoring to reconstruct sensor network dynamics. *Distributed Computing in Sensor Systems*, 5067:79–98, 2008.

[CS11] U. Colesanti and S. Santini. The collection tree protocol for the Castalia wireless sensor networks simulator. Technical Report 729, Department of Computer Science, ETH Zurich, 2011.

[CWCT11] Y. Chen, Q. Wang, M. Chang, and A. Terzis. Ultra-low power time synchronization using passive radio receivers. In *Proc. 10th Int'l Conf. on Information Processing in Sensor Networks (IPSN '11)*, pages 235–245, 2011.

[DAK$^+$09] P. Dutta, P. M. Aoki, N. Kumar, A. Mainwaring, C. Myers, W. Willett, and A. Woodruff. Common sense: Participatory urban sensing using a network of handheld air quality monitors. In *Proc. 8th ACM Conf. Embedded Networked Sensor Systems (SenSys '10)*, pages 301–318, 2009.

[DHJT$^+$10] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. sMAP: A simple measurement and actuation profile for physical information. In *Proc. 8th ACM Conf. Embedded Networked Sensor Systems (SenSys '10)*, pages 197–210, 2010.

[DHLT$^+$12] S. Dawson-Haggerty, S. Lanzisera, J. Taneja, R. Brown, and D. Culler. @scale: Insights from a large, long-lived appliance energy WSN. In *Proc. 11th Int'l Conf. Information Processing Sensor Networks (IPSN '12)*, pages 37–47, 2012.

[DMF12] L. Deri, S. Mainardi, and F. Fusco. tsdb: A compressed database for time series. In *Proc. 4th Traffic Monitoring and Analysis Workshop (TMA '12)*, pages 143–156, 2012.

[ECPS02] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *Pervasive Computing*, 1(1):59–69, 2002.

[EN03] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *Proc. 2nd ACM Int'l Conf. on Wireless Sensor Networks and Applications (WSNA 2003)*, page 87, 2003.

[ER03] J. Elson and K. Römer. Wireless sensor networks: A new regime for time synchronization. *ACM SIGCOMM Comp. Comm. Rev.*, 33(1):149–154, 2003.

[Fal03] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proc. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '03)*, pages 27–34, 2003.

[FW10] R. Flury and R. Wattenhofer. Slotted programming for sensor networks. In *Proc. 9th Int'l Conf. Information Processing Sensor Networks (IPSN '10)*, pages 24–34, 2010.

[FZMT12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proc. 10th ACM Conf. Embedded Networked Sensor Systems (SenSys '12)*, pages 1–14, 2012.

[FZTS11] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proc. 10th Int'l Conf. on Information Processing in Sensor Networks (IPSN '11)*, pages 73–84, 2011.

[GBG⁺12] L. Girard, J. Beutel, S. Gruber, J. Hunziker, R. Lim, and S. Weber. A custom acoustic emission monitoring system for harsh environments: Application to freezing-induced damage in alpine rock-walls. *Geoscientific Instrumentation, Methods and Data Systems*, 1(2):155–167, 2012.

[GCB⁺97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[GCME⁺10] J. Gupchup, D. Carlson, R. Musăloiu-E, A. Szalay, and A. Terzis. Phoenix: An epidemic approach to time reconstruction. In *Proc. 7th European Conf. on Wireless Sensor Networks (EWSN '10)*, pages 17–32, 2010.

[GFJ⁺09] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 1–14, 2009.

[GLvB⁺03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc. ACM SIGPLAN 2003 Conf. on Programming Language Design and Implementation (PLDI '03)*, pages 1–11, 2003.

[GM95] A. Gupta and I. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2):3–18, 1995.

[GMEST09] J. Gupchup, R. Musăloiu-E., A. Szalay, and A. Terzis. Sundial: Using sunlight to reconstruct global timestamps. In *Proc. 6th European Conf. on Wireless Sensor Networks (EWSN '09)*, pages 183–198, 2009.

[GZH09] S. Guo, Z. Zhong, and T. He. FIND: Faulty node detection for wireless sensor networks. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 253–266, 2009.

[HKWW06] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. TWIST: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proc. 2nd Int'l*

Workshop on Multi-hop Ad Hoc Networks (REALMAN '06), pages 63–70, 2006.

[HSL+11] T. W. Hnat, V. Srinivasan, J. Lu, T. I. Sookoor, R. Dawson, J. Stankovic, and K. Whitehouse. The hitchhiker's guide to successful residential sensing deployments. In *Proc. 9th ACM Conf. Embedded Networked Sensor Systems (SenSys '11)*, pages 232–245, 2011.

[HSST12] D. Hasenfratz, O. Saukh, S. Sturzenegger, and L. Thiele. Participatory air pollution monitoring using smartphones. In *Proc. 1st Int'l Workshop on Mobile Sensing: From Smartphones and Wearables to Big Data*, 2012.

[HTB+08] A. Hasler, I. Talzi, J. Beutel, C. Tschudin, and S. Gruber. Wireless sensor networks in permafrost research – Concept, requirements, implementation and challenges. In *Proc. 9th Int'l Conf. on Permafrost (NICOP '08)*, volume 1, pages 669–674, 2008.

[JCH84] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report 301, DEC, 1984.

[KBM+09] M. Keller, J. Beutel, A. Meier, R. Lim, and L. Thiele. Learning from sensor network data. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 383–384, 2009.

[KBT09] M. Keller, J. Beutel, and L. Thiele. Demo abstract: MountainView – Precision image sensing on high-alpine locations. In *Adjunct Proc. 6th European Workshop on Sensor Networks (EWSN '09)*, pages 15–16, Cork, Ireland, 2009.

[KDL+06] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler. Elapsed time on arrival: A simple and versatile primitive for canonical time synchronisation services. *Int'l Journal of Ad Hoc and Ubiquitous Computing*, 1(4):239–251, 2006.

[KGL10] M. Kazandjieva, O. Gnawali, and P. Levis. Visualizing sensor network data with Powertron. In *Proc. 8th ACM Conf. on Embedded Networked Sensor Systems (SenSys '10)*, pages 395–396, 2010.

[KHLK09] M. Kazandjieva, B. Heller, P. Levis, and C. Kozyrakis. Energy dumpster diving. In *Proc. 2nd Workshop on Power Aware Computing (HotPower'09)*, pages 1–5, 2009.

[KKP99] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for "smart dust". In *Proc. 5th ACM/IEEE Int'l Conf. on Mobile Computing and Networking (MobiCom '99)*, pages 271–278, 1999.

[KWL+11] M. Keller, M. Woehrle, R. Lim, J. Beutel, and L. Thiele. Comparative performance analysis of the PermaDozer protocol in diverse deployments. In *Proc. 6th IEEE Int'l Workshop on Practical Issues in Building Sensor Network Applications (SenseApp '11)*, pages 969–977, 2011.

[KYB09] M. Keller, M. Yuecel, and J. Beutel. High resolution imaging for environmental monitoring applications. In *Proc. Int'l Snow Science Workshop (ISSW '09)*, pages 197–201, 2009.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. of the ACM*, 21(7):558–565, 1978.

[LBL+13] Y. Lee, S. Bang, I. Lee, Y. Kim, G. Kim, M. Ghaed, P. Pannuto, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1 mm3 die-stacked sensing platform with low power i2c inter-die communication and multi-modal energy harvesting. *IEEE Solid-State Circuits*, 48(1):229–243, 2013.

[LBV06] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proc. 20th Int'l Parallel and Distributed Processing Symposium (IPDPS '06)*, pages 8–15, 2006.

[LDCE09] M. Lukac, P. Davis, R. Clayton, and D. Estrin. Recovering temporal integrity with data driven time synchronization. In *Proc. 8th Int'l Conf. Information Processing Sensor Networks (IPSN '09)*, pages 61–72. IEEE Computer Society, 2009.

[LFS+12] J. J. Li, B. Faltings, O. Saukh, D. Hasenfratz, and J. Beutel. Sensing the air we breathe – the OpenSense Zurich dataset. In *Proc. 26th Int'l Conf. on the Advancement of Artificial Intelligence (AAAI '12)*, pages 323–325, 2012.

[LFZ+13] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proc. 12th Conf. on Information Processing in Sensor Networks (IPSN '13)*, 2013.

[LHZ+06] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with EnviroLog. In *Proc. 25th*

IEEE Int'l Conf. on Computer Communications (INFOCOM '06), pages 1–14, 2006.

[LLL10] Y. Liu, K. Liu, and M. Li. Passive diagnosis for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 18(4):1132–1144, 2010.

[LLWC03] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys '03)*, pages 126–137, 2003.

[LMZL11] K. Liu, Q. Ma, X. Zhao, and Y. Liu. Self-diagnosis for large scale wireless sensor networks. In *Proc. 30th IEEE Int'l Conf. on Computer Communications (INFOCOM '11)*, pages 1539 – 1547, 2011.

[LSW09] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal clock synchronization in networks. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 225–238, 2009.

[Lub85] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. 17th Annual ACM Symposium on Theory of Computing (STOC '85)*, pages 1–10, 1985.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, 1(23):215–226, 1989.

[MHL$^+$09] L. Mo, Y. He, Y. Liu, J. Zhao, S.-J. Tang, X.-Y. Li, and G. Dai. Canopy closure estimates with GreenOrbs: Sustainable sensing in the forest. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 99–112, 2009.

[MKSL04a] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys '04)*, pages 39–49, 2004.

[MKSL04b] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys '04)*, pages 39–49, 2004.

[MLML12] Q. Ma, K. Liu, X. Miao, and Y. Liu. Sherlock is around: Detecting network failures with local evidence fusion. In *Proc. 31st IEEE Int'l Conf. on Computer Communications (INFOCOM '12)*, pages 792–800, 2012.

[MOH05] K. Martinez, R. Ong, and J. Hart. Glacsweb: A sensor network for hostile environments. In *Proc. 1st IEEE Communications Society Conf. Sensor, Mesh and Ad Hoc Communications and Networks (SECON '04)*, pages 81–87, 2005.

[MP11] L. Mottola and G. Picco. MUSTER: Adaptive energy-aware multi-sink routing in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 10(12):1694–1709, 2011.

[MPC⁺10] L. Mottola, G. P. Picco, M. Ceriotti, c. Gunǎ, and A. L. Murphy. Not all wireless sensor networks are created equal: A comparative study on tunnels. *ACM Trans. Sen. Netw.*, 7:15:1–15:33, September 2010.

[NRC⁺09] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava. Sensor network data fault types. *ACM Trans. Sens. Netw.*, 5(3):1–29, 2009.

[NT06] H. Nguyen and P. Thiran. Using end-to-end data to infer lossy links in sensor networks. In *Proc. 25th Int'l Conf. on Computer Communications (INFOCOM '06)*, pages 1–12, 2006.

[ODE⁺06] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with COOJA. In *Proc. 31st Conf. Local Computer Networks (LCN '06)*, pages 641–648, 2006.

[PHC04] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. 2nd Int'l Conf. on Embedded Networked Sensor Systems (SenSys '04)*, pages 95–107, 2004.

[RB06] S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *3rd Annual Conf. on Sensor and Ad Hoc Communications and Networks (SECON '06)*, pages 575–584, 2006.

[RCK⁺05] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys '05)*, pages 255–267, 2005.

[RGR09] A. Rowe, V. Gupta, and R. R. Rajkumar. Low-power clock synchronization using electromagnetic energy radiating from AC power lines. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 211–224, 2009.

[RM09] K. Römer and J. Ma. PDA: Passive distributed assertions for sensor networks. In *Proc. 10th ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN '09)*, pages 337–348, 2009.

[RNC04] M. Rabbat, R. Nowak, and M. Coates. Multiple source, multiple destination network tomography. In *Proc. 23th Int'l Conf. on Computer Communications (INFOCOM '04)*, volume 3, pages 1628–1639, 2004.

[SAM03] Y. Sankarasubramaniam, I. Akyildiz, and S. McLaughlin. Energy efficiency based packet size optimization in wireless sensor networks. In *Proc. 1st Int'l Workshop on Sensor Network Protocols and Applications (SNPA '03)*, pages 1–8, 2003.

[SBS$^+$11] I. Schweizer, R. Bärtl, A. Schulz, F. Probst, and M. Mühläuser. NoiseMap – Real-time participatory noise maps. In *Proc. 2nd Int'l Workshop on Sensing Applications on Mobile Phones (PhoneSense '11)*, pages 1–5, 2011.

[SDS10] T. Schmid, P. Dutta, and M. B. Srivastava. High-resolution, low-power time synchronization an oxymoron no more. In *Proc. 9th ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN '10)*, pages 151–161, 2010.

[SDTL10] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. An empirical study of low-power wireless. *ACM Trans. Sen. Netw.*, 6:16:1–16:49, 2010.

[SMP$^+$04] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys '04)*, pages 214–226, 2004.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2:277–288, 1984.

[TMEC$^+$10] A. Terzis, R. Musaloiu-E, J. Cogan, K. Szlavecz, A. Szalay, J. Gray, S. Ozer, C. Liang, J. Gupchup, and R. Burns. Wireless sensor networks for soil science. *Int'l Journal of Sensor Networks*, 7(1):53–70, 2010.

[TPS$^+$05] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys '05)*, pages 51–63, 2005.

[WALJ+06] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. 7th Symp. on Operating Systems Design and Implementation (OSDI '06)*, pages 381–396, 2006.

[WLB+11] V. Wirz, P. Limpach, J. Beutel, B. Buchli, and S. Gruber. Temporal characteristics of different cryosphere-related slope movements in high mountains. In *Proc. 2nd World Landslide Forum*, 2011.

[WTC03] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys '03)*, pages 14–27, 2003.

[YSSW07] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: A comprehensive source-level debugger for wireless sensor networks. In *Proc. 5th ACM Conf. Embedded Networked Sensor Systems (SenSys '07)*, pages 189–203, 2007.

[ZG03] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys '03)*, pages 1–13, 2003.

[ZGE02] Y. Zhao, R. Govindan, and D. Estrin. Residual energy scan for monitoring sensor networks. In *Proc. Int'l Conf. on Wireless Communications and Networking (WCNC '02)*, pages 356–362, 2002.

Curriculum Vitæ

Name	Matthias Keller
Date of Birth	22 April 1983
Nationality	German

Education:

2008–2013	Computer Engineering and Networks Laboratory, ETH Zurich PhD thesis under the supervision of Prof. Dr. L. Thiele Recipient of best paper award at SAMOS 2009 and DCOSS 2013
2006–2008	Fakultät für Informatik, Technische Universität München M.Sc. in Computer Science (with distinction) Exchange student at ETH Zurich, ERASMUS programme Scholarship of Lothar and Sigrid Rohde Foundation
2006–2008	Center for Digital Technology and Management Honours Degree in Technology Management (with distinction)
2003–2006	Fakultät für Elektrotechnik und Informationstechnik, Technische Universität München B.Sc. in Information Technology (with distinction)
1993–2002	Robert-Bosch-Gymnasium Langenau University-entrance degree (Abitur)

Professional Experience:

2008–2013	Research assistant at ETH Zurich
2007	HW/SW Engineer at MRX Technologies, Perth, Australia
2005–2007	Intern and working student at DaimlerChrysler Research and Technology, Ulm, Germany
2003–2008	Self-employed software developer and network administrator
2002–2003	Alternative civil service

List of Publications

The following list summarizes the publications that constitute the basis of this thesis. The corresponding chapters are indicated in parentheses.

M. Keller, L. Thiele, and J. Beutel. **Reconstruction of the correct temporal order of sensor network data.** In *Proc. 10th Int'l Conf. on Information Processing in Sensor Networks (IPSN '11)*, pages 282–293, Chicago, IL, USA, 2011. (Chapter 2)

M. Keller and J. Beutel. **Demo abstract: Efficient data retrieval for interactive browsing of large sensor network data sets.** In *Proc. 10th Int'l Conf. on Information Processing in Sensor Networks (IPSN '11)*, pages 139–140, Chicago, IL, USA, 2011. (Chapter 3)

M. Keller, J. Beutel, O. Saukh, and L. Thiele. **Visualizing large sensor network data sets in space and time with Vizzly.** In *Proc. 7th IEEE Int'l Workshop on Practical Issues in Building Sensor Network Applications (SenseApp '12)*, pages 963–971, Clearwater, Florida, USA, 2012. (Chapter 3)

M. Keller, J. Beutel, and L. Thiele. **Poster abstract: Multi-hop network tomography: Path reconstruction and per-hop arrival time estimation from partial information.** In *Proc. ACM SIGMETRICS/PERFORMANCE Joint Int'l Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*, pages 421–422, London, UK, 2012. (Chapter 4)

M. Keller, J. Beutel, and L. Thiele. **How was your journey? Uncovering routing dynamics in deployed sensor networks with multi-hop network tomography.** In *Proc. 10th ACM Conf. on Embedded Networked Sensor Systems (SenSys '12)*, pages 15–28, Toronto, ON, Canada, 2012. (Chapter 4)

M. Keller, J. Beutel, and L. Thiele. **The Problem Bit.** In *Proc. 9th IEEE Int'l Conf. on Distributed Computing in Sensor Systems (DCOSS '13)*, pages 105–114, Cambridge, MA, USA, 2013. **Best Paper Award**. (Chapter 5)

The following list summarizes the publications that were written during the PhD studies, yet are not part of this thesis.

M. Keller, J. Beutel, and L. Thiele. **Demo abstract: MountainView – Precision image sensing on high-alpine locations.** In *Adjunct Proc. 6th*

European Workshop on Sensor Networks (EWSN '09), pages 15–16, Cork, Ireland, 2009.

M. Keller, J. Beutel, A. F. Meier, R. Lim, and L. Thiele. **Poster abstract: Learning from sensor network data.** In *Proc. 7th ACM Conf. on Embedded Networked Sensor Systems (SenSys '09)*, pages 383–384, Berkeley, CA, USA, 2009.

J. Beutel, S. Gruber, S. Gubler, A. Hasler, M. Keller, R. Lim, I. Talzi, L. Thiele, C. Tschudin, and M. Yuecel. **The PermaSense remote monitoring infrastructure.** In *International Snow Science Workshop 2009: Programme and Abstracts*, pages 187–191, Davos, Switzerland, 2009.

M. Keller, M. Yuecel, and J. Beutel. **High resolution imaging for environmental monitoring applications.** In *International Snow Science Workshop 2009: Programme and Abstracts*, pages 197–201, Davos, Switzerland, 2009.

M. Keller, G. Hungerbuehler, O. Knecht, S. Sheikh, J. Beutel, S. Gubler, J. Fiddes, and S. Gruber. **Demo abstract: iAssist – Rapid deployment and maintenance of tiny sensing systems.** In *Proc. 8th ACM Conf. on Embedded Networked Sensor Systems (SenSys '10)*, pages 401–402, Zurich, Switzerland, 2010.

J. Beutel, B. Buchli, F. Ferrari, M. Keller, L. Thiele, and M. Zimmerling. **X-Sense: Sensing in extreme environments.** In *Proc. Design, Automation and Test in Europe (DATE '11)*, pages 1–6, Grenoble, France, 2011.

M. Keller and J. Beutel. **Demo abstract: Efficient data retrieval for interactive browsing of large sensor network data sets.** In *Proc. 10th Int'l Conf. on Information Processing in Sensor Networks (IPSN '11)*, pages 139–140, Chicago, IL, USA, 2011.

S. Gubler, J. Fiddes, M. Keller, and S. Gruber. **Scale-dependent measurement and analysis of ground surface temperature variability in alpine terrain.** *The Cryosphere*, 5(2):431–443, 2011.

M. Keller, M. Woehrle, R. Lim, J. Beutel, and L. Thiele. **Comparative performance analysis of the PermaDozer protocol in diverse deployments.** In *Proc. 6th IEEE Int'l Workshop on Practical Issues in Building Sensor Network Applications (SenseApp '11)*, pages 969–977, Bonn, Germany, 2011.

Y.-H. Chiang, M. Keller, R. Lim, P. Huang, and J. Beutel. **Poster abstract: Light-weight network health monitoring.** In *Proc. 11th Int'l Conf. on Information Processing in Sensor Networks (IPSN '12)*, pages 109–110, Beijing, China, 2012.

Image Credits

The following figures contain material that is copyrighted and/or has been contributed by the following individuals and organizations:

Page 45, Figure 3.1: Google (Nexus phone), Monica Tarocco (street car, http://www.monicatarocco.com/)

Page 47, Figure 3.3: Google, GeoBasis-DE/BKG