Reconstruction of the Correct Temporal Order of Sensor Network Data

Matthias Keller, Lothar Thiele, and Jan Beutel
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
{kellmatt,thiele,beutel}@tik.ee.ethz.ch

ABSTRACT

Collecting highly accurate scientific measurements asks for highest data quality and yield. But, satisfying these requirements is nontrivial, when considering phenomena common to wireless sensing systems such as clock drift, packet duplicates, packet loss and device reboots. Previous experience shows that these problems have not been resolved sufficiently by system design. In this paper, we introduce an offline approach to improve data quality by (a) providing a formal system model, (b) verifying conformance of packets received to this model, (c) providing the corrected packet sequence, and (d) providing additional information on packet generation inferred from temporally adjacent packets. We apply this method to a substantial amount of data from a real-world deployment and show the usefulness of this new intermediate packet processing step. In our validation of the proposed algorithm, we find that our approach successfully reconstructs the correct order of packet data streams. On application of the proposed data cleaning only a single violation is found when cross-validating a sequence of more than 4.6 million packets with ground truth derived from duplicate sensor data recovered from external storage post-deployment. The proposed method is thus suitable for both enhancing data accuracy on the occurrence of faults as well as the validation of data integrity.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Performance, Design, Measurement, Reliability, Verification

Keywords

Wireless Sensor Networks, Environmental Monitoring, Long Term, Data Analysis

IPSN'11, April 12–14, 2011, Chicago, Illinois.

1. INTRODUCTION

The field of wireless sensor networks (WSNs) is now in a stage where serious applications of societal and economical importance are in reach such as industrial process monitoring and control, environmental monitoring, logistics, healthcare applications, home automation, and traffic control. Most prominently, wireless sensor networks are today used for the collection of observations from the surrounding environment. For example, recent applications include ecosystem management [21], monitoring of heritage buildings [5], and data center monitoring [15].

In contrast to earlier visions of sensor networks, in most of these applications all sensor data samples and especially their integrity are of significance and once acquired must not be lost. Moreover, data must arrive ordered, sensors are often calibrated, sensor network deployments and their maintenance are labor-intensive and expensive. We argue that in order to significantly advance application domains of sensor networks as a novel means of observation and interaction, it is a requirement that such a tool be created as a quality instrument with well-understood and predictable properties.

Looking at an end-to-end system there are two major cases that lead to degradation in data quality. In the first case, 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 [22]. Secondly, artifacts originating in the data transmission system of a sensor network from the data source to the final data sink may exist. For example, when analyzing data from a real system [3] 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.* [1, 2] 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 [25].

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 [8]. 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 [23]. Lastly, data integrity validation is a valuable tool even if a system is designed and/or operating correctly.

While there has been extensive research for the first type of problems, *i.e.*, by using statistical methods [6], sensor fusion and intense data analysis of the transmitted packet payloads [25], 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 paper covers the first stage.

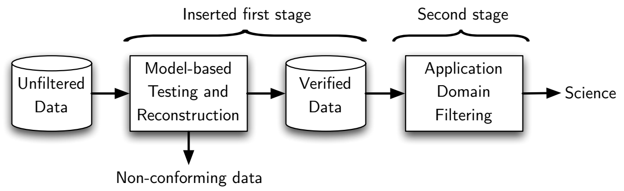


Figure 1: Two-staged process for improving sensor data quality. Arriving packets are first processed by only using application headers before filtering based on sensor data values is applied in the second stage.

In order to provide guarantees on the order of received packets, 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 received packets 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 [13] 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, duplicate packets as well as packets that do not conform to the formal system model are marked as such.

The contribution of this paper can be summarized as follows:

- We introduce an approach improving data quality by (a) providing a formal system model, (b) verifying conformance of received packets 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 29 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 4.6 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 paper is organized as follows. An overview of related work is given in Section 2. Section 3 provides a precise description of the considered problem. In Section 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 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 6. An overview of the broader applicability of our approach is given in Section 7.

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 [24, 25]. 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 [14, 20]. However, logical time is not sufficient in WSN applications that need to relate events in the physical world [7].

Time synchronization protocols such as FTSP [18] 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.* [26] 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 [19].

Non-applicability of network time synchronization protocols has been addressed by the idea of data driven time synchronization. Lukac *et al.* [17] use microseismics to reconstruct time information, Gupchup *et al.* [10] developed a similar approach for reconstructing the time from sunlight measurements.

Phoenix [9] 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 paper 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.

3. PROBLEM DESCRIPTION

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 received packets 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 $(o, s, p, \tilde{t}_s, t_b)$ consisting of the sender address o, the packet sequence number s, the payload p, 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, t_g^l and t_g^u. Here, id represents the temporal order of generation and t_g^l and t_g^u denote upper and lower bounds on the packet generation time.

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 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.

4.1 Packet Capturing Service

Following this modeling assumption, a sensor network contains several sensor nodes that run the packet capturing service. Each instance has the following properties:

- A source address o is unique in the sensor network. We will additionally use N_o for referring to the sensor node that runs the capture service having the source address o.
- 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 $\tau \bmod T = 0$, a data packet of the form (o, s, p) is generated that contains the source address o, a sequence number s and sensor data p. The sequence number is set to

$$s(i) = (i - i_{\text{offset}}) \bmod s_{\max} \tag{1}$$

where $s_{\max}$ bounds the transmitted sequence number. The space for storing and transmitting the sequence number s 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 to $s(i) := 0$. Our model abstracts the reset of the sequence number $s(i)$ 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 1.

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 the absolute capturing time $t_g(i)$ of a packet i is given as follows:

$$t_g(i) = t_g(i-1) + \frac{T}{1 + \rho(i)} \tag{2}$$

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

Table 1: 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 to be reset to $s(i) := 0$.

Sensor nodes can restart during operation. This can either be planned, *i.e.*, a reset button push, or unplanned, *i.e.*, the software watchdog resetting the sensor node due to an overrun of the task queue. 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 packet generations can be much smaller than the sampling period T. As a restart may occur directly after the generation of a packet, we find

$$0 < t_g(i) - t_g(i-1) \leq \frac{T}{1 + \rho(i)} \tag{3}$$

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 packets (o, s, p) generated by sensor nodes that run the capturing service.
- A packet is delivered to the sink S after a sojourn time t_s.
- It can duplicate packets arbitrarily, *i.e.*, it can generate an arbitrary number of copies from (o, s, p). These packets are forwarded independently from each other.
- It can delete packets arbitrarily, *i.e.*, a packet (o, s, p) 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 $(o, s, p, \tilde{t}_s, t_b)$ where (o, s, p) was the captured packet, $\tilde{t}_s$ is an estimate of the sojourn time t_s and t_b is the absolute time of arrival at S. The estimated sojourn time $\tilde{t}_s$ satisfies

  $$(1 - \hat{\rho}) \cdot t_s - \hat{h} \cdot \hat{t}_u < \tilde{t}_s \leq (1 + \hat{\rho}) \cdot t_s \tag{4}$$

  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.

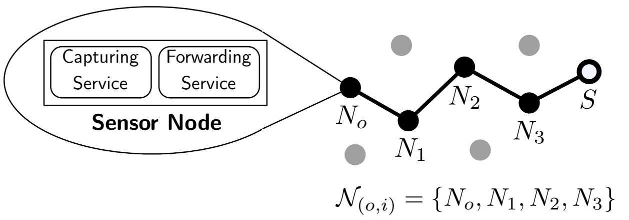


Figure 2: Travel of a packet i being generated at sensor node N_o. The packet is processed by all sensor nodes in $\mathcal{N}_{(o,i)}$ before it is finally received by the sink.

The sojourn time $t_s(N, i)$ is the time that packet i spent in the packet queue of some sensor node N of the forwarding network. We define $\mathcal{N}_{(o,i)}$ as the set of nodes that process a packet i originating from the sensor node N_o. Figure 2 describes the exemplary travel of a packet from sensor N_o to the sink S. The total sojourn time $t_s(i)$ of a packet is thus calculated as

$$t_s(i) = \sum_{N \in \mathcal{N}_{(o,i)}} t_s(N, i) \tag{5}$$

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 $\tilde{t}_s(i)$ of the sojourn time $t_s(i)$ for each packet available which is

$$\tilde{t}_s(i) = \sum_{N \in \mathcal{N}_{(o,i)}} \tilde{t}_s(N, i) \tag{6}$$

where $\tilde{t}_s(N, i)$ is the locally determined estimate of the sojourn time of packet i in node N. Considering uncertainties due to the drift as well as the resolution of the local clock, we have $\tilde{t}_s(N, i) \in ((1 - \hat{\rho}) \cdot t_s(N, i) - \hat{t}_u, (1 + \hat{\rho}) \cdot t_s(N, i)]$. The resulting interval for $\tilde{t}_s(i)$ is presented in (4).

5. DATA ANALYSIS

The goal of the analysis is 1) to identify and exclude duplicate packets, 2) to test the conformance of received packets 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.

5.1 Packet Generation Time Intervals

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

$$\tilde{t}_g(i) = t_b(i) - \tilde{t}_s(i) \tag{7}$$

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 $\rho(i)$ 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

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

This equation firstly addresses the introduced worst-case error of measuring the packet sojourn time $\tilde{t}_s(i)$ 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 $t_g(i)$

$$t_g(i) \in [t_g^l(i), t_g^u(i)]$$

where

$$t_g^u(i) := t_b(i) - \frac{\tilde{t}_s(i)}{1 + \hat{\rho}} \tag{8}$$

$$t_g^l(i) := t_b(i) - \frac{\tilde{t}_s(i) + \hat{h} \cdot \hat{t}_u}{1 - \hat{\rho}} \tag{9}$$

In Section 5.4, we will introduce forward and backward reasoning for improving $t_g^u(i)$ and $t_g^l(i)$.

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, 2) packet duplicates have the same sequence number s, and 3) packet duplicates have an equal payload p. Since packet duplicates travel through the network independently, they may have different estimated sojourn times $\tilde{t}_s$, but they will have overlapping generation time intervals $[t_g^l, t_g^u]$.

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, an equal sequence number s and equal payloads p. 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 $[t_g^l, t_g^u]$.

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:

$$(v, w) \in E \Leftrightarrow (t_g^u(v) \geq t_g^l(w)) \wedge (t_g^u(w) \geq t_g^l(v)) \tag{10}$$

In summary, duplicate filtering starts with separating a data set into subsets of a fixed originator o, a fixed sequence number s, and a fixed payload p. All subsets are analyzed independently. Firstly, the corresponding graph G of the subset is constructed. Then, we employ a standard algorithm [16] 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 p. 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.

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 $s(i) = (i - i_{\text{offset}}) \bmod s_{\max}$ 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 $s(i)$. More precisely, subsequent packets $i, i+1, ..., i+L-1$ belong to the same epoch if $s(i) = 0$, $s(j) = j - i$ for all $i < j < i + L$, and $s(i + L) = 0$.

- Epochs are labeled with incrementing index $e \in \mathbb{N}$, the index of the corresponding epoch of a packet i is denoted by $e(i)$.

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" $T_c(i)$ of a packet i which is computed offline according to

$$T_c(i) = \tilde{t}_g(i) - s(i) \cdot T \tag{11}$$

where $\tilde{t}_g(i) = t_b(i) - \tilde{t}_s(i)$ denotes the estimated generation time of packet i. 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.*, $\tilde{t}_g(i) = t_g(i)$, and the time difference between subsequent packets is T. Therefore, $T_c(i) = T_c(j)$ for all packets of an epoch $e(i) = e(j)$, *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 $s_{\text{max}} \cdot T$.

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 $s_{\text{max}} \cdot T$ 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 1. *All packets k, l that belong to the same epoch,* i.e., $e(k) = e(l)$*, satisfy*

$$|T_c(k) - T_c(l)| \leq \Delta T_c \tag{12}$$

where

$$\Delta T_c = (s_{max} - 1)(\hat{\rho}T + T - T') + T' + 2\hat{\rho}t_s^{max} \tag{13}$$

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

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

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

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

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

$$\begin{aligned} t_g(k) - s(k)T &= t_g(i_0) + s(k)\frac{T}{1-\hat{\rho}} - s(k)T \\ &\leq t_g(i_0) + (s_{\text{max}} - 1)\hat{\rho}T \end{aligned}$$

where i_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

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

Solving this equation for B and using the abbreviation

$$T' = \frac{1}{\frac{1+\hat{\rho}}{T} + \frac{1}{t_{\text{reset}}}}$$

yields a lower bound

$$B = (s_{\text{max}} - 2)T' \tag{14}$$

Now we can use this bound in order to determine

$$t_g(l) - s(l)T \geq t_g(i_0) + (s_{\text{max}} - 2)T' - (s_{\text{max}} - 1)T$$

As a result, we find now

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

which finishes the proof. □

THEOREM 2. *Suppose that the generation period T satisfies*

$$T > 2(1+\hat{\rho})\frac{\Delta T_c}{s_{max}} \tag{15}$$

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

$$T_c(l) - T_c(k) > \Delta T_c \tag{16}$$

where ΔT_c is defined in Theorem 1.

PROOF. The proof uses results from the proof of Theorem 1. In particular, we know that

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

If $e(l) = e(k) + 1$, then we can not use the same reference time $t_g(i_0)$ of the first packet of the common epoch anymore. Instead, the reference points of packets k and l differ by at least $s_{\text{max}} \cdot T/(1+\hat{\rho})$. Therefore, we find

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

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

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

which leads to the condition in the theorem. □

The condition on the nominal generation period T in Theorem 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:

$$t_s^{\text{max}} < \frac{1}{2\hat{\rho}}\left(\frac{s_{\text{max}} \cdot T}{2(1+\hat{\rho})} - (s_{\text{max}} - 1)(\hat{\rho}T + T - T') - T'\right) \tag{17}$$

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

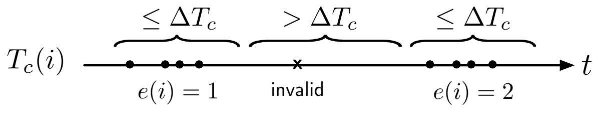


Figure 3: Assignment of packets to epochs. Epoch centers $T_c(i)$ of packets of the same epoch must lie within ΔT_c, see (12). In turn, epoch centers $T_c(i)$ of two packets that belong to disjoint epochs must be at least ΔT_c apart from each other, see (16).

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 i, j, and k where $T_c(i) - T_c(j) \leq \Delta T_c$ and $T_c(j) - T_c(k) \leq \Delta T_c$. In this case, i, j, and k need to belong to the same epoch due to Theorem 1. But if $T_c(i) - T_c(k) > \Delta T_c$, then i and k can not belong to the same epoch due to Theorem 2 which is a contradiction. The following Algorithm 1 uses this fact to mark or remove such packets.

Algorithm 1: Annotation of epoch e and index id to all packets. $pop()$ returns the next packet from the ordered data set, or *false* if all packets have been pulled.

```
input  : Packets of a single node, ordered by increasing $T_c(i)$
output : Packets with annotated epoch $e(i)$ and index $id(i)$
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
                $epoch \longleftarrow epoch + 1$ ; $f \longleftarrow k$ ;
        $e(k) \longleftarrow epoch$ ; $id(k) \longleftarrow epoch \cdot s_{\max} + s(k)$ ;
        $l \longleftarrow k$ ;
```

The question arises, why packets could violate Theorems 1 and 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, 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 $2\Delta T_c$ in order to guarantee a sufficient separation of epochs.

At first, all packets that do not satisfy the bound on the sojourn time in (17) are removed. The estimated epoch centers $T_c(i)$ are calculated for all packets. Algorithm 1 firstly checks for packets that violate Theorems 1 and 2. Non-violating packets are then annotated with the index of the corresponding epoch $e(i)$ and an index $id(i)$ which reflects the ordering of packets, *i.e.*, it satisfies $id(i) > id(j)$ if $t_g(i) > t_g(j)$. After executing Algorithm 1, all packets that have the same index id and epoch e also need to be marked as non-conforming.

5.4 Forward/Backward Reasoning

In Section 5.1, we introduced $t_g^u(i)$ and $t_g^l(i)$ as upper and lower bounds of the valid range of the unknown, perfect packet generation time $t_g(i)$. 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 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 the packet indices i are ordered, *i.e.*, we have $id(i) < id(i+1)$. The basis for the algorithm are bounds on the time difference between the generation of packets k and l with $k < l$

$$0 < t_g(l) - t_g(k) \leq (id(l) - id(k)) \frac{T}{1-\hat{\rho}} \tag{18}$$

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 (18) iteratively starting from the initial upper and lower bounds, see (19)-(22). 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 (19) with increasing index i and then (21) with decreasing index i. Likewise, the improved upper bounds are determined after firstly executing (22) with decreasing index i and then (20) with increasing index i.

(Forward reasoning)

$$t_g^l(i) := \max\left(t_g^l(i), t_g^l(i-1)\right) \tag{19}$$

$$t_g^u(i) := \min\Bigg(t_g^u(i), \min_{k>0}\Big\{t_g^u(i-k) + (id(i) - id(i-k))\frac{T}{1-\hat{\rho}}\Big\}\Bigg) \tag{20}$$

(Backward reasoning)

$$t_g^l(i) := \max\Bigg(t_g^l(i), \max_{k>0}\Big\{t_g^l(i+k) - (id(i+k) - id(i))\frac{T}{1-\hat{\rho}}\Big\}\Bigg) \tag{21}$$

$$t_g^u(i) := \min\left(t_g^u(i), t_g^u(i+1)\right) \tag{22}$$

If we use this order of execution, then in (19) and (22) we only need to take the nearest neighbor into account, *i.e.*, $k = 1$. In addition, no fixed point iteration is necessary. Let us show this for t_g^l only, as the other case can be handled similarly.

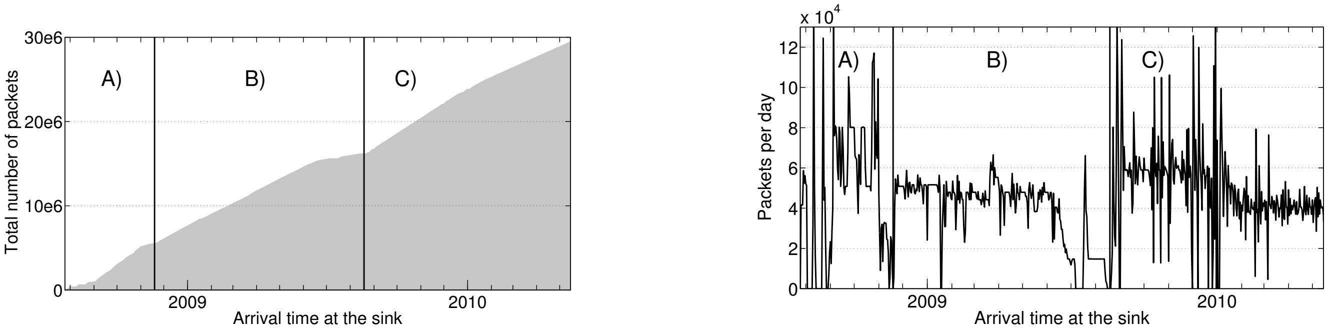
Obviously, after the forward phase, we have $t_g^l(i) \leq t_g^l(i+k)$ for all $k > 0$, and after the backward phase, (21) holds for all packets i. Therefore, we only have to show that $t_g^l(i) \leq t_g^l(i+k)$ still holds after the backward phase. Suppose that this is not the case, *i.e.*, there exists some i such that $t_g^l(i) > t_g^l(i+1)$. Then there must exist a packet $j > i$ which increased the bound for i in the backward phase to the new (larger) value, *i.e.*, $t_g^l(i) = t_g^l(j) - \Delta_1$ for some positive value Δ_1. $j = i+1$ is not possible as $t_g^l(i+1) < t_g^l(i)$ and therefore, we have $j > i+1$. Therefore, during the backward phase, packet j also may have changed packet $i+1$: $t_g^l(i+1) \geq t_g^l(j) - \Delta_2$ for some positive value Δ_2. As $j - i > j - (i+1)$ we have $\Delta_1 > \Delta_2$. Now, we can write $t_g^l(i) = t_g^l(j) - \Delta_1 < t_g^l(j) - \Delta_2 \leq t_g^l(i+1)$ which contradicts the assumption.

If after the execution of the algorithm there are packets i with $t_g^u(i) < t_g^l(i)$, 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.

6. CASE STUDY

The PermaSense project [11] 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 [3].

For this paper we consider a data set that consists of more than 29 million packets that have been gathered during a 23 month period.



(a) Total number of packets

(b) Number of packets per day

Figure 4: Number of received packets 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 deployment are denoted with A) to C).

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, are individually an locally time-stamped and a unique sequence number is added. A multi-hop data collection protocol [4] 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.

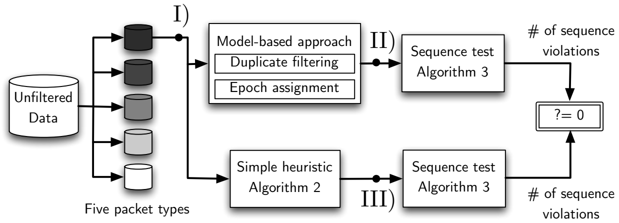


Figure 5: Case study validation strategy. We verify and compare the correctness of the packet sequences resulting from applying the model-based approach and a simple heuristic to the data set. Packet sequences are validated by a model of the behavior of the node uptime measurement. This measurement is included in *health packets*, one out of five packet types. Algorithm 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 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 paper.

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 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 4(b). The number of received packets per day is varying over time, the total number of received packets 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 duplicates in November 2008. In March 2009, initial data analysis identified a severe software problem that caused all sensor nodes to restart up to 40 times per day [12]. 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 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 4(b). While certainly an undesired behavior, the unique system design of Dozer [4] 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 5. An evaluation of the achieved gain by applying forward and backward reasoning concludes this case study.

6.1 Case Study Implementation

The results shown in this case study are based on a MATLAB implementation of the algorithms presented in Section 5. For processing our data set with the model-based approach, we use the parameter set which is shown in Table 2. 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

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}

Table 2: PermaSense system parameters

data from the database server, the execution time for analyzing the whole data set is less than one hour on a single processing core.

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 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 $0.8 \cdot s_{\max}$ allows to detect an overflow in the presence of a considerable amount of packet loss.

Algorithm 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
            $R \longleftarrow R \cup \{k\}$ ; $l \longleftarrow k$ ;
```

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, 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.

Algorithm 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         $l \longleftarrow k$ ;
```

We define $u(i)$ as the node uptime that has been transmitted within the payload $p(i)$ of a *health packet* i. Algorithm 3 lists the used test 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 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.

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: 40.6% of the data from the first deployment phase of non-conforming system operation are discarded. In contrast, 96.8% and 95.4% 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 3. The packet acceptance rate of 69.2% achieved with the simple heuristic on data from the first deployment phase is 9.8% higher than the acceptance rate of the model-based approach,

Counter	A) Jul 08-Nov 08	B) Nov 08-Aug 09	C) Sep 09-May 10
II) Model-based approach			
Accepted packets	632,058 (59.4%)	2,110,855 (96.8%)	2,579,444 (95.4%)
Sequence violations (Algorithm 3)	7 (0.0%)	11 (0.0%)	0 (0.0%)
Sequence violations (Ext. storage)	*n/a*	1 (0.0%)	0 (0.0%)
III) Simple heuristic			
Accepted packets	737,093 (69.2%)	2,100,558 (96.3%)	2,656,377 (98.2%)
Sequence violations (Algorithm 3)	3,063 (0.3%)	10 (0.0%)	1 (0.0%)
Sequence violations (Ext. storage)	*n/a*	0 (0.0%)	11 (0.0%)
I) Unfiltered data set			
Total packets	1,064,884 (100.0%)	2,180,684 (100.0%)	2,703,998 (100.0%)
Sequence violations (Algorithm 3)	189,645 (17.8%)	65,839 (3.0%)	46,987 (1.7%)
Sequence violations (Ext. storage)	*n/a*	69,004 (3.2%)	47,253 (1.7%)

Table 3: 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.

the simple heuristic also accepts 2.8% more data of the third phase.

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 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 $\tilde{t}_g(i)$. The non-conforming system behavior during the first deployment phase is again confirmed by 17.8% packets being marked as invalid due to a sequence violation. In opposite, only 3.0% and 1.7% 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 $\tilde{t}_g(i)$. In opposite, the new property $id(i)$ 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 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 the simple heuristic rejects less data leading to remaining 3,063 sequence violations, only 7 sequence violations 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 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 4.6 million packets.

More detailed results on the performance of the model-based filtering approach are shown in Table 4. It is notable, that only 130 packets of the second deployment phase, but 77,487 packets of the third deployment phase are discarded due to invalid generation time intervals. Regarding the two different underlying system behaviors, we must infer that the implementation of the system currently introduces an error for a considerable amount of packets having a high network sojourn time $\tilde{t}_s(i)$. This effect is currently not covered by our formal system model, further debugging facilities are needed for tackling this problem.

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.

6.5 Packet Generation Time Intervals

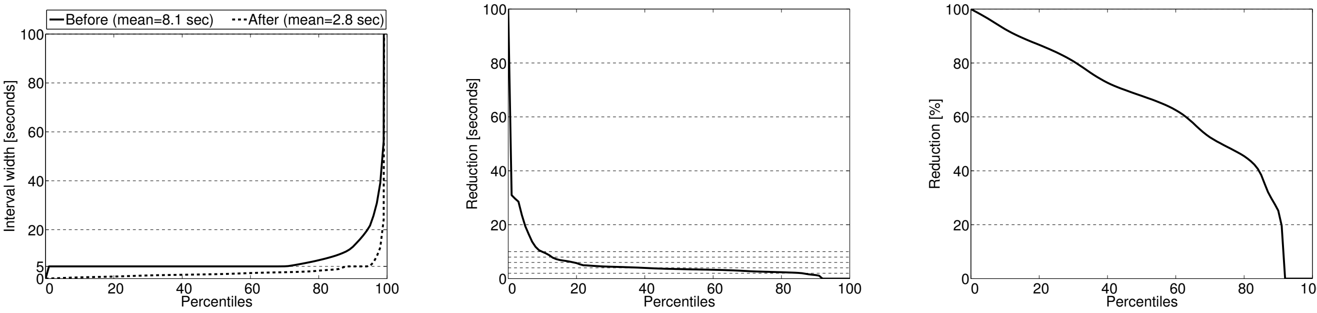
At the beginning of the data analysis, generation time intervals $[t_g^l(i), t_g^u(i)]$ were initially set by only including information from each single packet. In Section 5.4, we presented how these initially set intervals can be improved by also including information of temporally adjacent packets. 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 $t_g^u(i) - t_g^l(i)$ for 90% of the packets. The initial generation time interval width is at least reduced by half in three fourth of all cases. This corresponds to an absolute reduction between 2.6 and 100 seconds. The mean generation time interval width of all processed packets is significantly decreased by a factor of almost three from 8.1 seconds to 2.8 seconds.

Initial and improved generation time interval widths are shown in Figure 6(a). Circa 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 dominated by measuring the network sojourn time $t_s(i)$

Counter	A) Jul 08-Nov 08	B) Nov 08-Aug 09	C) Sep 09-May 10
Accepted packets	632,058 (59.4%)	2,110,855 (96.8%)	2,579,444 (95.4%)
Discarded packets	432,826 (40.6%)	69,829 (3.2%)	124,554 (4.6%)
Packet duplicates	4,020 (0.4%)	69,422 (3.2%)	44,601 (1.7%)
$t_s(i) > t_s^{\max}$	0 (0.0%)	0 (0.0%)	0 (0.0%)
Failed epoch assignment	235,927 (22.2%)	277 (0.0%)	2,466 (0.1%)
Invalid interval $t_g^{u,l}(i)$	192,879 (18.1%)	130 (0.0%)	77,487 (2.9%)
Total packets	1,064,884 (100.0%)	2,180,684 (100.0%)	2,703,998 (100%)

Table 4: Model-based approach, packet counters. 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 for the second and third deployment phases. The third deployment phase being an example for system operation with large transmission delays, we must infer that the implementation of the system currently introduces an error for a considerable amount of packets having a high network sojourn time $\tilde{t}_s(i)$.



(a) Generation time interval widths

(b) Absolute improvement

(c) Relative improvement

Figure 6: Improvement of generation time interval width $t_g^u(i) - t_g^l(i)$. The analysis covers 4,767,916 packets of the last two deployment phases. The initial generation time interval width could be shortened in 90% 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 40th percentile in Figure 6(a) corresponds to 40% 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 3.3 seconds or 62% of the initial value for 40% of the packets.

under clock drift in 12% of the cases.

Absolute and relative improvement by applying forward and backward reasoning are shown in Figure 6(b) and Figure 6(c), respectively. The initial generation time interval is reduced by up to 100 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 10% 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.

7. BROADER APPLICATION AND LIMITS

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 1 and 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 [18], 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 used data collection protocol. 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 [25] or environmental monitoring under extreme conditions [2]. Furthermore, glacier monitoring [19] 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. Even more, ensuring data qual-

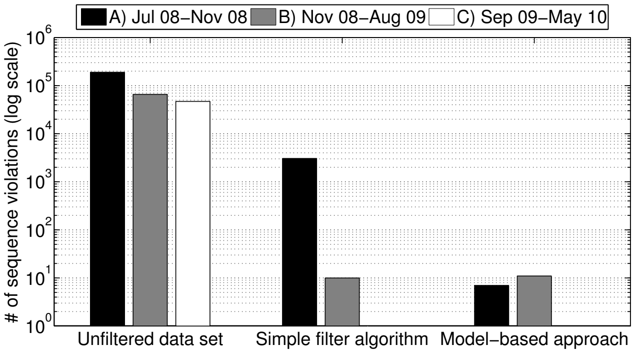


Figure 7: While both approaches succeed on data from the last two deployment phases of good system operation, only the model-based approach can deliver basically no sequence violations when applied to the first deployment phase of non-conforming system operation (Algorithm 3).

ity will become more complex when systems grow in scale.

There are several possible extensions to this work, *i.e.*, the generation of multiple packets in each sampling period, or additional sporadic, aperiodic generation of packets.

8. CONCLUSIONS

The proposed model-based approach is a viable method for reconstructing the correct sequence of packet generation and validation of data integrity at the sink. Only a single violation is found when cross-validating a sequence of more than 4.6 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 an important parameter for removing uncertainties when reconstructing the packet sequence. Second, our new approach for offline data processing also allows to shift complexity from resource-scarce sensor nodes to powerful computation devices in the backend. Depending on the application, services such as clock synchronization can be intentionally left out without sacrificing data quality.

9. ACKNOWLEDGEMENTS

We want to thank our shepherd Anish Arora, the anonymous reviewers, Federico Ferrari, and Marco Zimmerling for their valuable feedback that helped us to improve this paper. The work presented was supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322, Nano-Tera.ch, and the Swiss-Experiment, a project funded by the Competence Center Environment and Sustainability of the ETH Domain (CCES).

10. REFERENCES

[1] 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)*, 2008.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *Proc. 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA '03)*, pages 78–87, 2003.

[7] J. Elson and K. Römer. Wireless sensor networks: a new regime for time synchronization. In *SIGCOMM Comput. Commun. Rev.*, 33(1):149–154, 2003.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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 Comput.*, 1(4):239–251, 2006.

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

[15] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. Racnet: a high-fidelity data center sensing network. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys '09)*, pages 15–28, 2009.

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

[17] 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, 2009.

[18] 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.

[19] K. Martinez, R. Ong, and J. Hart. Glacsweb: a sensor network for hostile environments. In *Proc. 1st IEEE Int'l Conf. Sensor and Ad Hoc Communications and Networks (IEEE SECON '04)*, pages 81–87, 2004.

[20] F. Mattern. Virtual time and global states of distributed systems. In *Proc. Int'l Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.

[21] 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.

[22] 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 Transactions on Sensor Networks*, 5(3):1–29, 2009.

[23] 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.

[24] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proc. 1st European Workshop on Sensor Networks (EWSN '04)*, pages 307–322, 2004.

[25] 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.

[26] 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. Operating Systems Design and Implementation (OSDI '06)*, pages 27–27, 2006.