Fence Monitoring – Experimental Evaluation of a Use Case for Wireless Sensor Networks

Georg Wittenburg, Kirsten Terfloth, Freddy López Villafuerte, Tomasz Naumowicz, Hartmut Ritter, and Jochen Schiller

Department of Mathematics and Computer Science
Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
{wittenbu,terfloth,lopez,naumowic,hritter,schiller}@inf.fu-berlin.de

Abstract. In-network data processing and event detection on resource-constrained devices are widely regarded as distinctive and novel features of wireless sensor networks. The vision is that through cooperation of many sensor nodes the accuracy of event detection can be greatly improved. On the practical side however, little real-world experience exists in how far these goals can be achieved.

In this paper, we present the results of a small deployment of sensor nodes attached to a fence with the goal of collaboratively detecting and reporting security relevant incidents, such as a person climbing over the fence. Based on experimental data we discuss in detail the process of in-network event detection both from the conceptual side and by evaluating the results obtained. Reusing the same traces in a simulated network, we also look into the impact of multi-hop event reporting.

Keywords: Wireless sensor networks, in-network data processing, event detection, experimental evaluation, use case, fence monitoring.

1 Introduction, Goals and Motivation

The close cooperation of individual sensor nodes in order to achieve a common goal is a key feature of wireless sensor networks (WSNs). While a wealth of distributed algorithms has been proposed and evaluated in the areas of medium access and routing, the situation is different for distributed event detection: Several theoretical approaches have been described [1,2,3], but to the best of our knowledge none of them has been evaluated in a real-world experiment.

On the other hand, several high-profile deployments of WSNs have been undertaken and evaluated by various research groups [4,5,6]. However, they all have in common that they are largely data-agnostic and limit themselves to reporting raw data as collected by the sensor nodes deployed in the field. Only recently we have seen first evaluations of the accuracy of the readings and attempts to perform complex in-network data aggregation and event detection [7,8]. One of the key features of WSNs – in-network data processing and event detection – is thus still widely unexplored in practice.

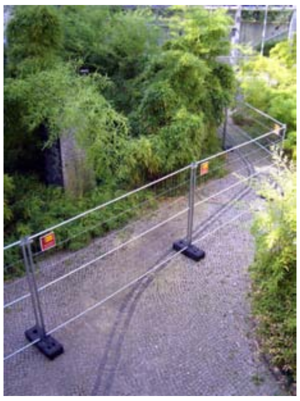


Fig. 1. Patio of institute with construction fence

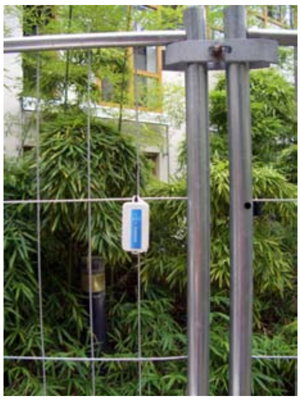


Fig. 2. Sensor node attached to the fence

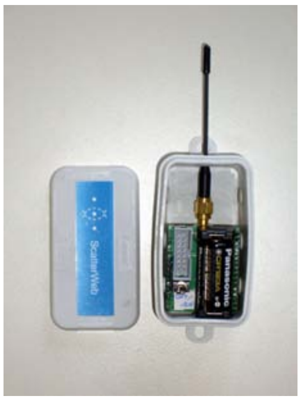


Fig. 3. Casing of a sensor node

In this paper, we present the results of a real-world experiment on in-network event detection. Our experiment is built around the example of a fence monitoring application whose task it is to detect and report any security related incidents that may occur on a fence, in our case a person climbing over the fence and entering a supposedly restricted area. The focus of this application is thus clearly different from those of other deployments which were mostly concerned with some flavor of environmental monitoring.

Furthermore, fence monitoring is an excellent example for the demand for in-network data processing: Similar to the questions raised in [9], the sheer volume of raw data caused by a single event makes it impractical to transmit the complete data to a base station for processing, especially when keeping energy considerations in mind. Further, the sensor readings caused by a security-related event can be expected to differ sufficiently from those of common every-day events, and thus there should be a reasonable chance for a distributed, in-network event detection algorithm to succeed.

Summing things up, the primary goals of our fence monitoring use case are:

– to establish whether fence monitoring is feasible with current WSN technology by setting up a working system,
– to quantify the accuracy of our event detection algorithm with a special focus on differences between node-local and distributed event detection, thereby putting a number to the value added by networked sensors, and
– to develop and describe a systematic approach to building a robust event detection and reporting algorithm that performs reliably even in a multi-hop scenario.

To these ends, and as shown in Figures 1-3, we deployed a construction fence in the patio of our institute. To each element of the fence we attached a ScatterWeb sensor node equipped with an accelerometer to measure its movement. We then first calibrated the sensors to respond to the typical movements of fence

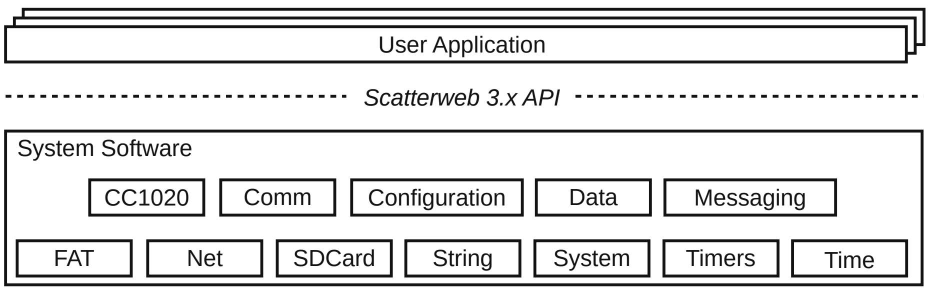


Fig. 4. The ScatterWeb 3.x software architecture

elements, and proceeded afterwards to gather samples of raw data corresponding to different types of events. Based on these, we isolated distinctive features of the raw data corresponding to different types of events, and implemented and evaluated a distributed event detection algorithm.

After a brief introduction into the WSN platform and tools we used in our experiments in Section 2, we present the details of our deployment and the software architecture of our distributed event detection algorithm in Section 3. In Section 4, we thoroughly evaluate this algorithm based on both real-world experiments and simulations. Finally, in Section 5 we review related work and in Section 6 we conclude and point out directions for future research.

2 The ScatterWeb WSN Platform

For setting up our experiment, we used a new version of the ScatterWeb research hardware [10], which has been recently developed with a focus on modularity: The Modular Sensor Board (MSB) as already depicted in Figure 3 consists of a core communication module, add-on sensor modules and an optional interface board. The core communication module consists of the TI MSP430 16-bit microcontroller, the Chipcon CC1020 868MHz radio transceiver, and connectors for analog and digital sensors and actuators. Furthermore, a Freescale Semiconductor MMA7260Q accelerometer is soldered directly onto the board.

An add-on board allows for a broad variety of sensors if needed, ranging from luminosity to motion and from sound to GPS. The core module and the sensors of the add-on board can be powered either by a 3V battery or by the interface board which in addition also provides a flash interface and a USB connection for debugging and power supply.

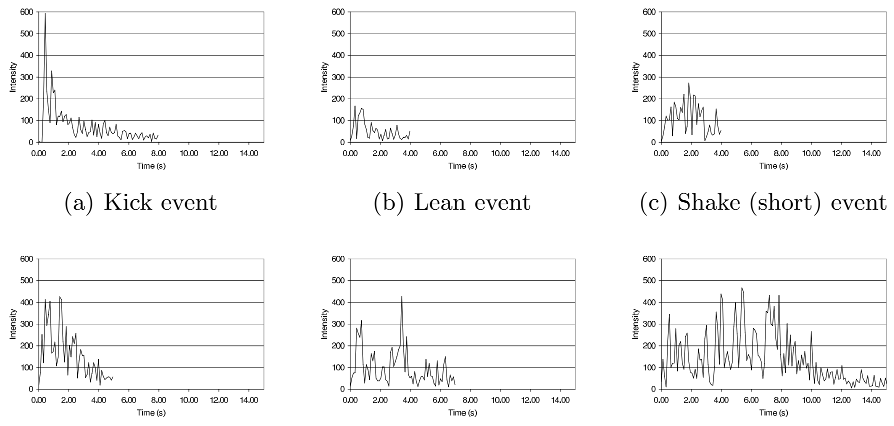
On the software side, we were able to depend on the broad range of features already available for this platform. These include the ScatterWeb system software, which is responsible for supporting basic tasks such as interrupt handling, packet handling, medium access, management of run levels and debugging options, as well as a rich application programming interface (API) as depicted in Figure 4. Aside from the work on the ScatterWeb core, extensive efforts have been undertaken to supply tool support with the ScatterWeb Software Development Kit (SDK) which is based on Microsoft Visual Studio 2005.

The experiments run on top of the FACTS middleware, a framework especially designed for WSNs and implemented on top of the ScatterWeb platform [11]. Capturing the intrinsic challenges of dealing with low-resource devices and event-centric programming at the language level, the core of FACTS is built around the rule-based Ruleset Definition Language (RDL). Using RDL, a developer is able to specify sets of interacting rules to define node and network behavior. At runtime, these rules are evaluated locally against the fact repository, a central data storage entity on a sensor node, by a sandboxed execution environment.

Currently RDL supports standard operations for fact manipulation, filtering, comparison and different flavors of aggregation. Support for running native code, e.g. to efficiently implement complex mathematical computations or to access hardware directly, is integrated into the language. The design of FACTS aims towards low resource consumption, thus a lot of work has been spent on reducing the memory footprint down to 8KB in terms of the middleware components installed on the sensor nodes.

3 Experimental Setup and Software Architecture

Based on the available hard- and software components introduced in the previous section, we will now present how our construction fence testbed was set up and which types of events we considered in our experiments. We will then continue to describe the architecture of our distributed event detection algorithm.



(a) Kick event (b) Lean event (c) Shake (short) event

(d) Shake (long) event (e) Peek event (f) Climb event

Fig. 5. Raw data of different event types

3.1 Construction Fence Deployment

As already illustrated in Figure 1, we deployed a ten-element construction fence in the patio of our institute. Each fence element is 3.5m wide and 2m high. The exact layout of this deployment is shown in Figure 8. We rigged this installation with one ScatterWeb MSB sensor node per fence element, each node attached to the right hand side of the element at a height of 1.65m (Figure 2). Weather-proof junction boxes with a size of 80mm $\times$ 40mm served as casing of the sensor nodes (Figure 3). It is however worth noting that the sensor nodes themselves fit nicely into the hollow metal frame of a fence element, a location at which they would be even more protected from the environment, possibly at the expense of radio transmission quality.

3.2 Types of Events

In our experiments, we considered the following six events as typical scenarios that a fence monitoring system will be exposed to:

Kick: A person kicks against the fence.
Lean: A person gently leans against the fence.
Shake (short): A person shakes the fence for a short period of time.
Shake (long): Same as above, but for a prolonged period of time.
Peek: A person climbs up the fence with the intention to take a look into the restricted area.
Climb: A person climbs over the fence.

Assuming that a person climbing over a fence is the only event with security implications, the important question is how a WSN can be programmed to reliably identify this single type of event that is worth reporting. We did not consider telling the other events apart to keep our use case as realistic as possible, although this would have been possible with additional rules similar to the one shown in Listing 1.1.

We programmed a ScatterWeb MSB sensor node to sample its accelerometer at 10Hz with a sensitivity setting of 1.5g. In Figure 5, we show the raw data in terms of the sum of the differences of elements between the previous three dimensional acceleration vector $\overrightarrow{\boldsymbol{v}}_{last}$ and the current vector $\overrightarrow{\boldsymbol{v}}_{cur}$. In the following, we will refer to this scalar quantity as the *intensity* I of an event.

$$I = |\ (v_{x\_last} - v_{x\_cur})\ | + |\ (v_{y\_last} - v_{y\_cur})\ | + |\ (v_{z\_last} - v_{z\_cur})\ |$$

Looking at the figures, we note that each event has a more or less unique pattern which may be used for event detection. To prevent problems possibly arising from limited memory resources, we chose not to implement a pattern matching algorithm based on raw data. Instead, we propose a layered architecture to handle this task.

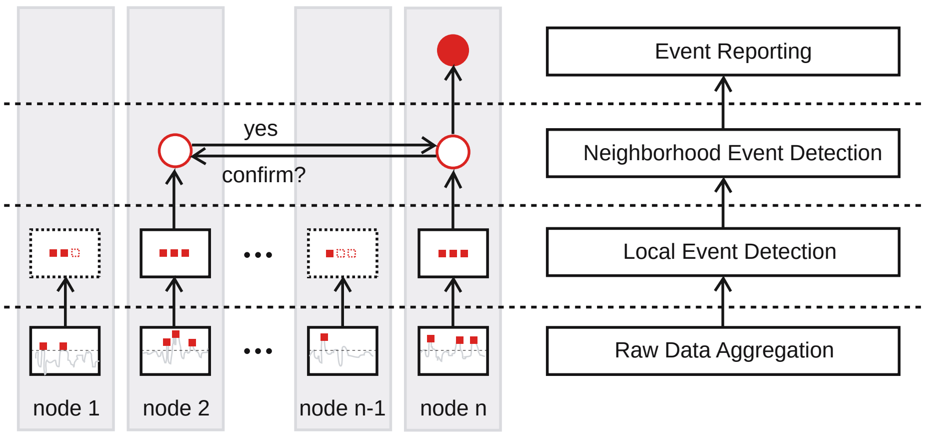


Fig. 6. Layers of the distributed event detection algorithm

3.3 A Layered Software Architecture for Event Detection

The different layers of our architecture implement a distributed, multi-step event detection algorithm. In the lowest layer, raw sensor readings are isolated from background noise and aggregated into a set of characteristic properties. The next layer checks whether known patterns appear in these aggregated values and identifies them as event candidates. In the next layer, the sensor nodes collaboratively decide whether a noteworthy event has in fact occurred within an n-hop neighborhood by exchanging information about recently observed event candidates. Finally, the uppermost layer reports confirmed events to the base station of the deployment. Figure 6 illustrates how these layers interact and the level of abstraction at which they process data.

We first introduce the exact functionality of each layer in this section, while returning to calibration parameters and implementation details in the following section.

Raw Data Aggregation: This layer periodically retrieves the current acceleration vector from the accelerometer and converts it into the intensity as described above. As soon as the intensity surpasses a predetermined threshold value, the sampling rate is increased and new intensity values are aggregated upon retrieval from the sensor. Once the intensity falls below a second threshold value, the sampling rate is decreased for energy efficiency and a tuple of the aggregated data values is reported to the local event detection layer. In the following, we will refer to this tuple of aggregated data values as *basic event*. Typical aggregated data items are the minimum, average, and maximum of all intensity values sampled during the basic event, as well as the total duration of the basic event. Depending on the nature of the event patterns to be recognized, additional aggregated data items may also be considered.

The advantages of this design for the raw data aggregation layer are twofold: Memory usage is kept at a minimum by aggregating sensor readings as they are being sampled, and excessive energy consumption is avoided during intervals in which no events occur. The drawback is that the raw data itself is not available for event detection. However, we regard this as unproblematic given the right selection of data items to aggregate.

Local Event Detection: Based on the basic events reported by the raw data aggregation layer, this layer matches the aggregated data contained in the basic events against previously established patterns. The goal of this procedure is to aggregate one or many basic events into an *event candidate*, i.e. to infer the action that just occurred at the fence from patterns in the aggregated data.

To ensure reliable event detection, patterns in the aggregated data contained in the basic events must be sufficiently distinctive. This of course depends largely on the application and on the types of events that may occur. Hence, the patterns in basic events that lead to event candidates need to be established carefully by the means of either a manual or automatic process before the WSN can be deployed. Once identified, the event candidates are handed up to the neighborhood event detection layer.

Neighborhood Event Detection: In this layer, event candidates are propagated within an n-hop neighborhood of the sensor node that originated the candidate. While theoretically this procedure involves multi-hop communication, given the fact that for the fence monitoring application the radio range of current sensor nodes exceeds the area in which sensors gather data related to an event, we limit ourselves to communication within the one-hop neighborhood.

Upon receiving an event candidate, each sensor node evaluates its own currently available basic events and event candidates and depending on the settings of the distributed aggregation algorithm sends an acknowledgement to the originating node. Similar to the local event detection layer, the parameters of whether to acknowledge an event candidate or not depend on the application and need to be carefully established before an actual deployment. If the sensor node that originally broadcasted the event candidate within its neighborhood receives enough positive replies within a certain period of time, it may safely regard the event candidate as a *confirmed event*, and thus send it to the base station.

Event Reporting: The event reporting layer is not an intrinsic part of in-network event detection and we include it in our model merely for architectural completeness. The task of this layer is to route the confirmed events from the sensor nodes that reported them to the base station. A great variety of routing algorithms for WSNs have been proposed and we omit a thorough evaluation for brevity. In our implementation, we have used a simple spanning tree routing algorithm with the base station being the root of the tree.

At each layer, several parameters need to be configured in order to reliably identify basic events, event candidates, and confirmed events. For different application scenarios, e.g. different types of fence elements, these parameters need to be carefully established, possibly as part of a calibration phase before the system becomes operational. During this calibration phase, one should keep in mind that event detection must be triggered locally on at least one sensor node. Therefore, the calibration should aim at setting the detection threshold rather low for local event detection and only afterwards try to eliminate false positive event candidates in the neighborhood event detection layer.

3.4 Design Alternatives and Robustness Considerations

There are several ways to refine parts of the event detection architecture described in the previous section which we did not implement in our experiments for simplicity. Most of them are related to increasing the robustness of the distributed event detection algorithm with regard to packet loss on the wireless medium.

In our current implementation, there is no mechanism to ensure reliable delivery of event candidates within the neighborhood. As a result, if an event candidate is not delivered due to a packet collision on the wireless medium, no `ACK`s are sent and a possibly valid event candidate is discarded. The way to solve this issue is by replying to all event candidates with either an `ACK` as described above or with a `NACK` in case local data fails to confirm that this event occurred. The originating sensor node can then count the number of `ACK`s and `NACK`s received and retransmit the event candidate if this number is below a threshold. This procedure incurs an increase in communication which may well have side effects on energy consumption and packets collisions.

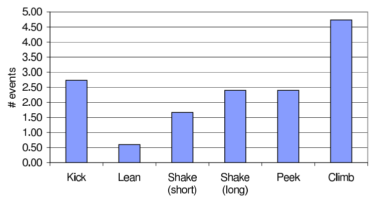
Similar to the problem lined out above, it may also occur that confirmed events are lost on their way to the base station. This may either be solved as part of the routing protocol, or alternatively the sensor node may retransmit the event if no `ACK` was received from the base station after a certain time interval. Further, it would be desirable to only report exactly one confirmed event to the base station for each real-world event.

4 Deployment and Experimental Results

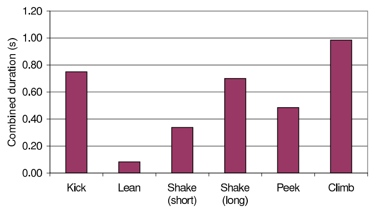
We conducted over 40 test runs comprising all possible events described in Section 3.2 on the deployed construction fence equipped with our fence monitoring WSN. Each of these runs included one event occurrence per type, thus the fraction of climb events per run is $\frac{1}{6}$. Ten of these runs were used for the calibration of the raw data aggregation layer and 15 runs for the local event detection layer respectively.

4.1 Calibration Values and Implementation Details

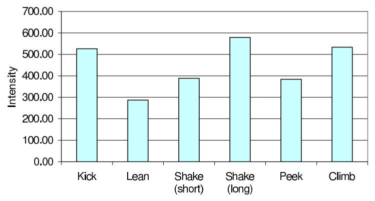
From the raw data already presented in Figure 5, we concluded that an intensity threshold of 200 nicely filters out background noise and minor events. As soon as the intensity value surpasses this threshold, the sampling rate is increased from 1Hz to 10Hz, and the values of the accelerometer are aggregated into basic events. Each basic event contains the duration of its sampling period in milliseconds and



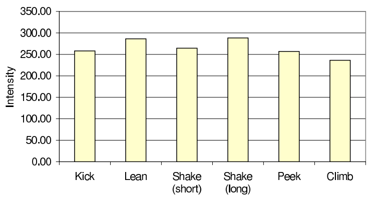
(a) Number of events



(b) Combined duration



(c) Maximal intensity



(d) Minimal intensity

Fig. 7. Aggregated data of different event types on one node

the average intensity during this time. The data did not warrant using a different second threshold value for the hysteresis, hence the sampling rate is reduced to 1Hz again once the intensity falls below the same threshold.

The motivation for chosing the duration and the average intensity as parameters of the basic events is related to the distinctive patterns in different aggregated data values. In Figure 7, we show a selection of these patterns for some of the aggregated data values we considered. For both the number of basic events and their combined duration (Figures 7(a) and 7(b)), we note that the values for climb events differ from all others. On the other hand, no clear pattern can be observed for maximal or minimal intensity (Figures 7(c) and 7(d)). Therefore, on the local event detection layer, the number of basic events produced by the lower layer and their combined duration are suitable values for event detection while maximal and minimal intensity are not. Also note that the average intensity is not used for event detection on this layer, but rather passed to the upper layers for later evaluation.

While there are several options on how to implement our layered event detection architecture, we opted for a rule-based implementation supported by our middleware. Rules such as the one shown in Listing 1.1 suit this particular application because the event-centric semantics of the programming language map nicely to the problem of event detection. For instance, the rule shown will trigger as soon as three conditions evaluate to true: The number of basic events generated has to be greater or equal to three (line 2), the sum of the duration of these events has to be greater or equal to 0.49s (line 3) and smaller or equal to 1.71s (line 4). Once again, we derived these values by studying trace files, thus manually calibrating the local event detection layer. Once all conditions are met and this rule fires, an event candidate is generated which also records the

Listing 1.1. Ruled-based local event detection

```
1 rule aggregateBasicEvents 100
2 <- eval ((count {basicEvent}) >= 3)
3 <- eval ((sum {basicEvent duration}) >= 0.49)
4 <- eval ((sum {basicEvent duration}) <= 1.71)
5 -> define eventCandidate [intensity = (max {basicEvent
      intensity})]
6 -> retract {basicEvent}
```

maximal intensity of the basic events that trigger its creation (line 5) and all basic events are retracted from the system (line 6). The entire event detection ruleset, including for example the rule that purges unused basic events from the system after 30s, consists of 15 rules and has a memory footprint of 1.4KB. A full introduction into the Ruleset Definition Language (RDL) and the exact semantics of the FACTS runtime environment are available in [11].

In the neighborhood event detection layer, we programmed a sensor node to broadcast an event candidate within its one-hop neighborhood since this range covers all nodes that may have been exposed to the possible climb event. Upon reception of an event candidate and given sufficient local information that an event occurred, a node confirms this by sending an `ACK` to the originating node. If an `ACK` is received by the originator within a 1s interval, the event is regarded as confirmed and handed to the event reporting layer, which in turn forwards this information to the base station.

In order to properly evaluate the event reporting layer of our WSN use case, we decided to focus on a much larger deployment than the one at our disposal. We therefore resorted to a simulation-based evaluation using the traces of basic events obtained during our experiments and the "ScatterWeb on `ns-2`" simulation approach which allows to run unmodified algorithms on both ScatterWeb sensor nodes and the `ns-2` network simulator [12]. As a typical scenario we chose the construction site of the U.S. embassy located in the center of Berlin.

The layout of the simulated deployment is shown in Figure 10. It consists of 105 sensor nodes placed 3.5m apart from each other along the fence line. Taking into account the expected decrease in signal strength if sensor nodes are placed within the metal frame of a fence element, we set the transmission range to 10m as part of the configuration of the two-ray ground radio propagation model.

4.2 Results and Discussion

We use two statistical metrics for binary classification, sensitivity and specificity, to quantify the accuracy of our event detection algorithm.[1] The goal is to maximize both values, i.e. to correctly classify both events and non-events.

[1] Sensitivity is the ratio of correctly identified climb events and all climb events that occurred, i.e. `sensitivity = #true positives / (#true positives + #false negatives)`. Specificity is the ratio of correctly identified other events and all other events, i.e. `specificity = #true negatives / (#true negatives + #false positives)`.

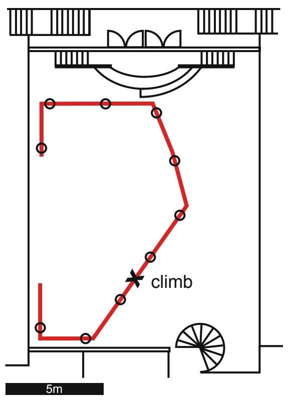


Fig. 8. Construction fence layout in the patio of our institute

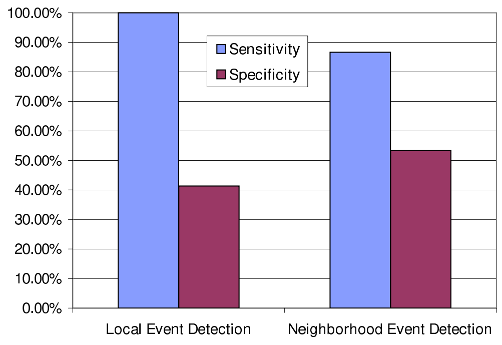


Fig. 9. Experimental results of in-network event detection accuracy

In Figures 8 and 9, we illustrate the construction fence deployment in the patio of our institute and the experimental results obtained. At the local event detection layer, a single node of our system performs with a sensitivity of 100% and a specificity of 41.3%. These values indicate that one of our design goals - setting the detection threshold rather low in this layer - has been achieved, since all climb events have been recognized. On the downside, 59.7% of the all other events are also classified as event candidates. We observe that the specificity is increased by 12.0% by the neighborhood event detection layer. This increase comes at the expense of incurring a 13.3% decrease in sensitivity. These values show that our neighborhood event detection layer does well at filtering out false event candidates, but regretably also correct detections. Still, this is consistent with our design principle of a low event detection threshold in the local event detection layer and a higher threshold in the neighborhood event detection layer.

This level of accuracy observed is less than the one we had expected after our initial test runs, especially with regard to the high number of false positives. We attribute this to a variety of factors: On the technical side, we had two node failures during the experiment and of course this resulted in unforeseen inaccuracies during neighborhood event detection. Further, we suspect that gathering trace data at the same time as performing event detection also adversely affected accuracy. More important and consistent with the evaluation found in [7] is however the fact, that the evaluation of the traces suggests that slight variations of the parameters would have significantly improved the results obtained. From this we have to conclude that our manual calibration of the algorithm needs to be improved. On the non-technical side, we note that the event patterns changed over time as our test candidates got more proficient in climbing over the fence.

Before proceeding to simulate a large scale deployment to quantify the impact of multi-hop event reporting, we verified the accuracy of the simulation by rebuilding our original experiment within the simulator and playing back the

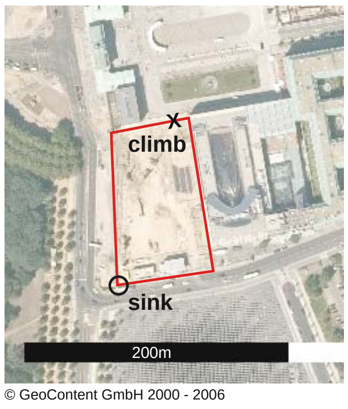


Fig. 10. Simulated construction fence layout around the U.S. embassy construction site in Berlin

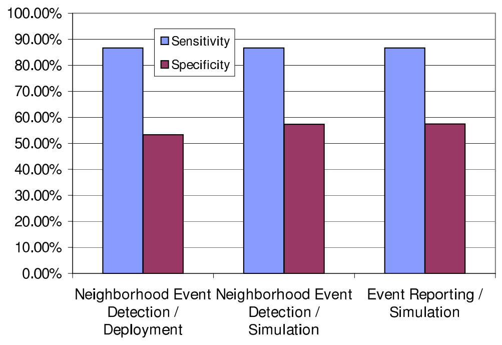


Fig. 11. Experimtental and simulation-based results of in-network event detection and reporting accuracy

original traces. We then ran a series of ten simulations, the average results of which are shown in Figure 11 alongside with the real-world experimental data. As we observed only a minor 4% increase in specificity due to slightly larger packet loss, we concluded that running large scale simulations is appropriate.

The average results of ten simulation runs of the U.S. embassy construction site scenario as illustrated in Figure 10 are included in Figure 11 labeled as "Event Reporting / Simulation". We note that even our very simplistic approach to event reporting, relying on little more than a spanning tree routing, does not have a negative impact on the results, with variations in both sensitivity and specificity below 1%. While our simulation does not include node failures, based on our data and in light of the progress in robust routing protocols for WSN, we still tend to regard event reporting and routing as only a minor problem in the use case of fence monitoring.

Apart from the increase in accuracy, in-network event detection has the additional benefit of reducing the data that needs to be sent to the base station. In Figure 12, we quantify this advantage by looking at the number of packets transmitted during the entire simulation. The figure contains the number of packets sent by our complete layered architecture as well as the same numbers for the two hypothetical cases in which either basic events or event candidates are transmitted to the base station for centralized event detection.

For a transmission range of 10m, our data shows that locally aggregating basic events into event candidates reduces the overall traffic by 79.3%. Aggregating event candidates into confirmed events reduces the overall traffic by another 68.4%. This corresponds to a total reduction of 93.4% of the traffic by our layered event detection architecture.

This reduction of traffic by means of in-network aggregation depends on the topology of the network. For instance, in the trivial case of a network in which all nodes are located within the 1-hop neighborhood of the base station,

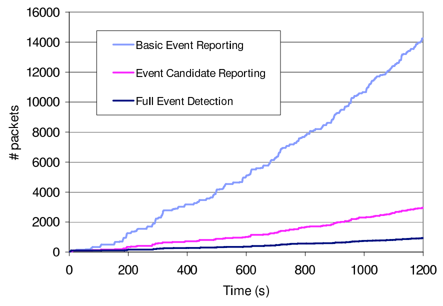


Fig. 12. Number of packets transmitted over time in the simulation with 10m transmission range at different levels of in-network aggregation

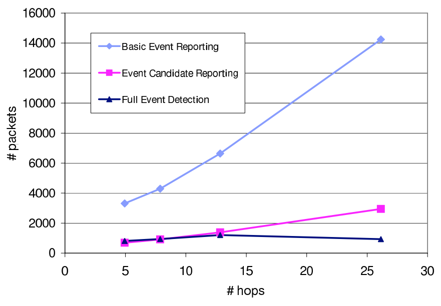


Fig. 13. Number of packets transmitted against hops between event source and base station at different levels of in-network aggregation

in-network aggregation will hardly reduce the number of packets transmitted. In fact, the distributed event detection algorithm even incurrs the overhead of data transmissions as part of the neighborhood event detection layer, which is not required if all event candidates are received by the base station.

In order to examine the impact of n-hop event reporting, we ran additional simulations with the transmission range of the simulated sensor nodes set to 10m, 20m, 30m, and 40m. These transmission ranges correspond to an average hop count of 26.14, 12.84, 7.88, and 4.94 between the source of the event and the base station respectively. The number of packets at each level of in-network aggregation for these hop counts is shown in Figure 13. We observe that the reduction of traffic attributed to the neighborhood event detection layer decreases when less hops are required to report the event candidates to the base station. As expected, the diagram also shows the small overhead of in-network aggregation for very well connected topologies. Further, the decrease in confirmed events for a 10m transmission range as opposed to a 20m transmission range, while not directly affecting the accuracy, shows that robustness becomes an issue for reporting events over higher numbers of hops. Possible solutions to this issue have already been discussed in Section 3.4.

Turning our attention to the reduction of traffic due to local aggregation, we note that even at low hop counts the number of packets is still reduced by 75.6%. This underlines that the value added by detecting events locally on the sensor nodes is largely independant of the network topology.

5 Related Work

As already briefly mentioned in the introduction, several theoretical approaches to event detection have been published. On the local and the neighborhood event detection layer, both Petri nets [1] and boolean expressions [2] have been proposed, however no evaluation of these algorithms is presented. In [3], the authors propose to employ Probabilistic Context Free Grammars (PCFGs) in a layered architecture similar to ours and evaluate this approach using a simulation with real traces. Their use case of recognizing motion patterns differs from ours in that it allows to infer semantic meaning of raw data locally on a sensor node, and their evaluation stops short of actually quantifying the accuracy of the system.

On the other hand, a wealth of deployments of WSNs have been described. To mention but a few, deployments have focussed on habitat monitoring [4], fire detection [5], and environmental monitoring [6]. While some of these applications are good candidates for in-network event detection, this functionality neither was an integral part of any of these deployments, nor did the authors report on the level of accuracy of the event detection algorithm used, if any. Instead, reports on deployments mostly limit themselves to describing the raw data collected.

Focussing more on event detection, research untertaken within the NEST project deals with discrimination of people, vehicles and noise using radar-enabled sensor networks [13]. While we opted for classifying the events observed within the network, the authors describe a base-station centerd classification approach and the trade-off between classification accuracy and latency.

The two projects most similar to our work were published by He *et al.* in [7] and by Werner-Allen *et al.* in [14,8]. In [7], He describes the VigilNet project, a system for surveillance missions with applications such as vehicle tracking. It has a broader scope than our work in that it comprises a deployment at a much larger scale and event detection is only one component of their system. The authors did not focus as much on the event detection algorithm as we did, as the only parameter that is mentioned is the degree of aggregation which corresponds to the number of `ACK`s send at our neighborhood event detection layer. It is also unclear how many different types of events they exposed their system to. Based on our experience, we can however support their claim that slight miscalibrations of the event detection algorithm have an immense impact on its accuracy.

In [14] and later followed-up by [8], Werner-Allen *et al.* evaluate a deployment of sensor nodes on an active volcano with the goal of monitoring volcanic eruptions. The traces obtained during the first deployment were used to both evaluate an offline event detection algorithm and calibrate the event detection algorithm for the second deployment. The architecture of the algorithm deployed differs from ours in that sensor nodes send basic events to the base station and in response to this the base station may decide to collect data from all nodes in the network, while our approach relies entirely on in-network event detection. The results of their second deployment as published in [8] indicate that the accuracy of their event detection architecture faces worse problems than ours under real-world conditions. Sensitivity is very low at 1.2% and specificity is at 100%, which we attribute to a miscalibration of the parameters of the event detection algorithm used during this deployment.

6 Conclusion and Future Work

The goal of this paper was to explore how a security-focussed system relying on the in-network data processing capabilities of WSNs can be constructed. We chose the example of a fence monitoring application due to both its demanding requirements on distributed event detection and realism of the use case. Putting our layered approach to event detection into practice, we have built and evaluated fence monitoring deployments both in real-world and simulated experiments.

Our system showed a sensitivity of of 86.7% and a specificity of 53.3% during these experiments. Distributed event detection contributed to the specificity by eliminating false event candidates, however at the same time decreased the sensitity by eliminating correct detections. Further, our layered approach to in-network event detection was able to reduce the overall network traffic by up to 93.4% depending on the network topology as compared reporting aggregated sensor data to the base station for centralized processing. Our results are novel in so far as to the best of our knowledge no previous work has quantitatively evaluated the impact of in-network processing based on real-world experiments.

At the same time it must be noted that the level of accuracy we achieved in our experiments is by far not sufficient for a production-level deployment. In the future, we need to focus on refining the calibration phase of the event detection algorithm with the goal of reducing the number of false positive detections. This may include looking at the raw data in a transformed domain to optain a better differentiation of events and examining whether a pattern recognition approach (e.g. k-nearest neighbors) for classification is more suitable. Preferably, calibration should be an automated process instead of the manual calibration we utilized. Fortunately, our comparison between a real deployment and a simulation relying on the same traces indicates that simulation is a viable tool for studying this kind of application. Hence, given enough traces of raw data, it should be feasible to perform the calibration using simulation tools.

Another problem to be tackled is how to avoid that failures of individual nodes and the resulting variation in the average node degree of the WSN adversely affects neighborhood event detection. One possibility we plan to evaluate in this context of self-organization are periodic runs of recalibration phases. Once these adaptations prove successful, we hope to verify our findings in a large scale deployment over a longer period of time as part of which we can also evaluate the long-term energy consumption of our fence monitoring system.

References

1. Jiao, B., Son, S.H., Stankovic, J.A.: GEM: Generic Event Middleware for Wireless Sensor Networks. In: Proceedings of the Second International Workshop on Networked Sensing Systems (INSS'05), San Diego, U.S.A. (2005)
2. Kumar, A.V.U.P., Reddy, A.M., Janakiram, D.: Distributed Collaboration for Event Detection in Wireless Sensor Networks. In: Proceedings of the Third International Workshop on Middleware for Pervasive and Ad-hoc Computing, Grenoble, France (2005) 1–8
3. Lymberopoulos, D., Ogale, A.S., Savvides, A., Aloimonos, Y.: A Sensory Grammar for Inferring Behaviors in Sensor Networks. In: Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN'06), Nashville, U.S.A. (2006)

4. Szewczyk, R., Polastre, J., Mainwaring, A., Culler, D.: Lessons From a Sensor Network Expedition. In: Proceedings of the First European Workshop on Sensor Networks (EWSN'04), Berlin, Germany (2004)
5. Doolin, D.M., Sitar, N.: Wireless Sensors for Wildfire Monitoring. In: Proceedings of SPIE Symposium on Smart Structures & Materials / NDE'05, San Diego, California, U.S.A. (2005)
6. Martinez, K., Padhy, P., Riddoch, A., Ong, R., Hart, J.: Glacial Environment Monitoring using Sensor Networks. In: Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN'05), Stockholm, Sweden (2005)
7. He, T., Krishnamurthy, S., Stankovic, J.A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Zhou, G., Hui, J., Krogh, B.: VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance. ACM Transactions on Sensor Networks (TOSN) **2**(1) (2006) 1–38
8. Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity and Yield in a Volcano Monitoring Sensor Network. In: Proceedings of the Seventh USENIX Symposium on Operating Systems Design and Implementation (OSDI'06), Seattle, U.S.A (2006)
9. Marrón, P.J., Sauter, R., Saukh, O., Gauger, M., Rothermel, K.: Challenges of Complex Data Processing in Real World Sensor Network Deployments. In: Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06), Uppsala, Sweden (2006) 43–48
10. Schiller, J., Liers, A., Ritter, H.: ScatterWeb: A Wireless Sensornet Platform for Research and Teaching. Computer Communications **28** (2005) 1545–1551
11. Terfloth, K., Wittenburg, G., Schiller, J.: FACTS - A Rule-Based Middleware Architecture for Wireless Sensor Networks. In: Proceedings of the First International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE'06), New Delhi, India (2006)
12. Wittenburg, G., Schiller, J.: Running Real-World Software on Simulated Wireless Sensor Nodes. In: Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06), Uppsala, Sweden (2006) 7–11
13. Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H., Naik, V., Mittal, V., Cao, H., Demirbas, M., Gouda, M., Choi, Y.R., Herman, T., Kulkarni, S.S., Arumugam, U., Nesterenko, M., Vora, A., Miyashita, M.: A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. Computer Networks **46**(5) (2004) 605–634
14. Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., Welsh, M.: Monitoring Volcanic Eruptions with a Wireless Sensor Network. In: Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN'05), Istanbul, Turkey (2005)