International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:3, No:8, 2009

FleGSens — Secure Area Monitoring Using Wireless Sensor Networks

Peter Rothenpieler, Daniela Krüger, Dennis Pfisterer, Stefan Fischer
Institute of Telematics
University of Lübeck
Ratzeburger Allee 160
23538 Lübeck, Germany
{rothenpieler|krueger|pfisterer|fischer}@itm.uni-luebeck.de

Denise Dudek, Christian Haas, Martina Zitterbart
Institute of Telematics
University of Karlsruhe
Engesserstraße 2
76128 Karlsruhe, Germany
{dudek|haas|zit}@tm.uni-karlsruhe.de

Abstract—In the project FleGSens, a wireless sensor network (WSN) for the surveillance of critical areas and properties is currently developed which incorporates mechanisms to ensure information security. The intended prototype consists of 200 sensor nodes for monitoring a 500m long land strip. The system is focused on ensuring integrity and authenticity of generated alarms and availability in the presence of an attacker who may even compromise a limited number of sensor nodes. In this paper, two of the main protocols developed in the project are presented, a tracking protocol to provide secure detection of trespasses within the monitored area and a protocol for secure detection of node failures. Simulation results of networks containing 200 and 2000 nodes as well as the results of the first prototype comprising a network of 16 nodes are presented. The focus of the simulations and prototype are functional testing of the protocols and particularly demonstrating the impact and cost of several attacks.

Keywords—Wireless Sensor Network, Security, Trespass Detection, Testbed.

I. Introduction

OVER the past years, wireless sensor networks (WSNs) and their applicability for a vast number of scenarios have been the focus of research worldwide. Although the need for secure communication protocols in WSNs is widely accepted, having put forth many proposals to that end, these protocols have rarely been applied in a real-world environment.

One of the most interesting application scenarios for WSNs is the domain of area surveillance systems. The project FleGSens realises such a surveillance system for critical areas like e.g. borders or private properties. Its main objectives are the secure detection and signalling of trespassers within a predefined area in the presence of both malicious and non-malicious interference. Due to the heavy restrictions in terms of power supply, memory capacity and processing power that is typical for WSNs, it is mandatory to incorporate security considerations into every step of protocol design. As sensor networks are disproportionally more vulnerable to attacks than classical networks, the presented protocols are designed with special regard to the presence of a strong attacker that may even compromise a certain number of nodes in the WSN. The assumed attacker is a classical Dolev-Yao attacker [1] with some extensions to pay up to the specific possibilities that come with the use of wireless sensor nodes, notably physical access to the nodes. The applied security mechanisms focus on providing authenticity and integrity of the information reported by the network; to this end, cryptographic hash functions and message authentication codes (MACs) are used. It is not intended to guarantee confidentiality, seeing as no information is processed that needs particular protection through encryption.

This paper presents a basic system for area surveillance using only simple passive infrared sensors (PIR sensors) for trespass detection. The system consists of a trespass detection protocol that detects trespassers and signals the detection towards a dedicated gateway, and a node failure detection protocol which informs the gateway if a node fails to respond for a specified period of time. The authors present evaluation results of this system both in a simulated environment and in a first prototype.

The remainder of this paper is organised as follows: In section II an overview is given over previous works that have been proposed in the areas of surveillance and object tracking; section III discusses the security analysis, resulting in an application specific attacker model upon which the protocol design and implementation is based. In section IV the design of the trespass and node failure detection is presented in detail. Sections V and VI show the results of the simulation and real-world deployment of the developed system respectively. The paper is concluded in section VII with an outlook on the next steps.

II. Related Work

Surveillance and tracking applications for WSNs have been the subject of both academic and military research in the past.

An early work has been contributed by Yang et al. in 2003 [2]. The authors focus on algorithms for detecting and tracking an object by evaluation of basic information transmitted to a base station by a sensor network. The nodes broadcast information on whether an object moves towards them or away from them, as well as an estimation of the object's proximity. To estimate the object position and movement direction the authors use particle filters; however, the question of how to communicate the information towards the base station is left open, as well as any security aspects. In the field of military research, extensive analyses have been presented by Arora et al. [3]. The authors focus on the classification of objects, introducing three different categories thereof. They use multiple and complex sensors such as radar, magnetic or even optical sensors. As the sensors produce a large amount of data, the authors propose algorithms to track an identified object efficiently; however, the approach to the underlying communication remains a minor point in [3], and therefore aspects of information security are not discussed either. Xu et al. [4] propose a cluster-based communication protocol to convey information about targets identified by object tracking sensor networks. Both base station and sensor nodes estimate the target's position; the sensor nodes exchange the relevant information inside the cluster until sensory input and estimation results differ. Only in that case the sensor nodes notify the base station using multihop communication. Thus, communication overhead is reduced to a level suitable for sensor networks. No mention is made of security aspects or malicious attackers, so that questions of information security still remain open. Furthermore, the authors assume more complex sensors to produce the data needed for the estimation than are given in the FleGSens context. Another proposal to tackle the problem of efficient communication has been made by Yang et al. [5]. Similar to [4], the authors use cluster-based communication to reduce the overhead introduced by multihop communication and a prediction scheme to estimate an object's position. Clusterheads can wake up the nodes of their clusters when the object moves into their sensor range; this enables the authors to put most of the nodes into an energy-saving sleep mode most of the time. The authors, however, do not include the presence of an attacker in their reasoning, leaving security related questions open; also, cluster management is a non-trivial problem causing computational and communication overhead.

In summary, to the best of the authors' knowledge, the works proposed in the fields of object tracking and surveillance do not provide solutions for security problems induced by a malicious attacker in the sensor network. Furthermore, often communication aspects are only touched upon, leaving open the questions when and how to communicate information, as well as the arising issues of scaleability and robustness.

This paper focuses on communication and security regarding the detection of trespasses across an area monitored by a sensor network using only minimal sensor material. The paper presents a localised approach that does not need cluster communication, thus having to cope with much less overhead in that department, and evaluate the system by means of simulation and prototype testing.

III. Security Analysis

This section describes the security analysis that was performed before designing and implementing the protocols presented herein. The design of any secure architecture requires exact knowledge of what the architecture in question has to be able to defend against. Often, this is done intuitively or implicitly; however, this approach is error-prone and complicates debugging and maintenance work unnecessarily. Due to the need to include security considerations in every step of the development process, and to enable a validation of security-related properties of the protocols, both general and application-specific goals of an attacker are captured in an attacker model.

A. Attacker Model

An attacker model defines the goals as well as the means an attacker has at their disposal to achieve those goals. It is therefore necessary to analyse both in detail since the design of the whole security system is based upon the assumptions made about the attacker that is present in the network.

The word *attacker* in the context of network security often implicitly refers to a specific attacker – the so-called *Dolev-Yao* [1] or *Man-in-the-Middle* attacker. This attacker is capable of listening to any and all messages communicated over the network medium. They also may insert messages into the network or manipulate any message sent by a legitimate participant of the network. The Dolev-Yao attacker is a globally acting, collaborating attacker, which means that they may act at any location in the network and that all malicious instances – i.e. the attacker's nodes – share their information and knowledge about the network instantly. To this end, they may or may not use an out-of-band channel: the essential point is that they do not have to use the attacked network for their own communication. The Dolev-Yao attacker is limited in one aspect, though, as they are not able to guess cryptographic material – i.e. keys – by brute force in a reasonable time. Consequently, they can only ever use the keys they already know. Specifically, in the context of data integrity and authenticity as mentioned above in section I, this means the attacker is not capable of masquerading as a node whose key material they do not know.

Sensor networks, however, offer a potential attacker feasible ways to further influence the network's function. Typically – and in the case of FleGSens intrinsically – sensor nodes are deployed in non-protected environments. This means that anyone can relatively easily access the nodes physically. The Dolev-Yao attacker model is insufficient to pay up to this potential, as physical access is not considered in this model. Therefore, in WSNs, the Dolev-Yao attacker is often extended to include the WSN-specific avenues: the attacker may destroy nodes they have physical access to; they may read the memory of those nodes and thus gain access to the cryptographic secrets stored there; they may even reprogram the nodes. Nodes thus reprogrammed are called *compromised* nodes. It is important to note that node compromise is virtually impossible to detect; compromised nodes may be indefinitely well-behaved until they take malicious action.

The FleGSens WSN basically consists of two different kinds of nodes: there are one or several gateways in the network as well as ordinary sensor nodes. The node types differ only by the fact that gateways possess a secure and permanent connection to a base station as well as a permanent power supply. Gateways make up 1% of the overall node population. Protocols in the context of the FleGSens scenario must be able to deal with up to 5% of the nodes to be compromised. Compromised nodes are assumed to be approximately uniformly distributed in the network, the only exception being that gateways and the base station to which they are connected cannot be compromised. Furthermore, it is assumed that up to 10% of the nodes can be permanently down e.g. due to power exhaustion or damage. Table I summarises the conditions under which the FleGSens system must be able to function.

Parameter	Value
Compromised nodes	5%
Permanently failed nodes	10%
Gateways	1%

TABLE I
BOUNDARY CONDITIONS OF THE FLEGSENS SCENARIO

B. Application-specific Objectives

With the boundary conditions for both the FleGSens system and the attacker strength thus defined, the following application specific goals of the attacker can be identified:

a) Preventing the network from reporting trespassers: If a trespasser crosses the area monitored by the WSN, the network has to communicate and process the sensor events triggered by the trespasser and raise an alarm when indicated. The attacker therefore might want to hinder the communication or processing of sensor events. Given the attacker has compromised a sufficient amount of nodes in the network, they can achieve this by not relaying messages concerning the detection of the trespasser at critical points. This attack will be referred to as *Silence Attack* in the following. Redundant sensor coverage of the area and redundant routing paths through the network complicate this attack. Another venue the attacker might choose consists in destroying the nodes on the trespasser's path. This also motivates the deployment of a node failure monitoring protocol to ensure that missing nodes are reported to the gateway.

b) Delaying the report of trespassers: Instead of alarm prevention, the attacker may want to delay the report of trespassers, enabling them to cross the monitored area sufficiently long before the network maintainer arrives on site. This could be achieved by a typical *Jamming Attack* blocking the frequency used for communication and therefore forcing the network to establish alternative paths to a gateway. The use of redundant routing paths and several gateways reduces the risk of falling victim to this kind of attack.

c) Manipulating the information reported about a trespasser: If an alarm is raised, the network should simultaneously provide information as to the last known locations of the trespasser, e.g. as identified by the network addresses of nodes that register the trespasser via their sensors. If the attacker is able to manipulate the information sent in the according messages, it will be difficult up to impossible to locate the trespasser in a reasonable amount of time, resulting in their unhindered passage of the monitored area. If communication is authentic and information integrity is granted e.g. by using MACs at application layer, an outside attacker cannot manipulate the messages without being detected – inside attackers, e.g. in the form of compromised nodes may still launch the *Manipulation Attack* described here, as far as they are privy to the cryptographic keys needed.

d) Covering up node failures: If an attacker can cover up node failures a trespasser could choose their path across the monitored area according to where the failed nodes are and thus be able to cross over undetected. Covering up node failures could be achieved by replaying old messages of the failed nodes, thus suggesting that the nodes in question are still up and running. It is the objective of a secure node failure detection protocol to protect against this kind of *Replay Attack*.

IV. PROTOCOL DESIGN

This section of the paper describes in depth the design of the protocols running within the basic FleGSens system. As mentioned in section I, the basic system consists of two protocols at application layer: one for trespass detection and node failure detection respectively. First, the trespass detection protocol as the core of the application is introduced in Section IV-A. Where applicable according to the attacker objectives specified in Section III-B, the need for security mechanisms is pointed out. In Section IV-B the secure node failure detection protocol is described.

Beneath the application layer, the FleGSens protocol architecture uses a hop based routing network layer and an IEEE 802.15.4 link layer. The network itself follows a grid topology as illustrated by figure 1, where a and b denote length and width of the monitored area, n identifies the number of nodes and F illustrates the sensor range.

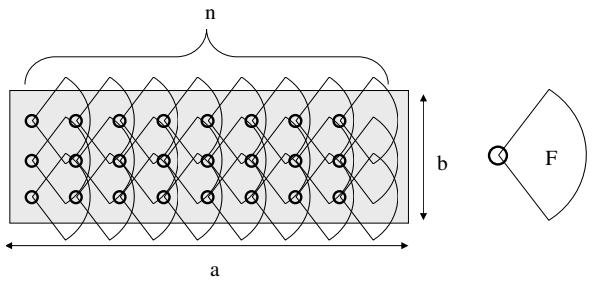


Fig. 1. Grid topology of the FleGSens scenario

A. Trespass Detection Protocol

A trespasser that moves within the range of a PIR sensor will create a so-called PIR event – i.e. an interrupt on the node whose PIR sensor has detected the movement. A PIR event may be characterised by the timestamp of its registration and the location or ID of the node that registers it. Note that no complex sensor information is needed to identify a trespasser.

The PIR sensors only provide binary information: movement or no movement.

The most intuitive and easiest way to communicate the detection of a trespasser towards one or several data gateways is to flood an *Event* message into the network with every registered PIR sensor event. An *Event* message contains the timestamp of the PIR interrupt, the position where the event has been detected and a message authentication code – calculated using a predistributed key shared between sender and data gateway – that ensures authenticity of the sender and information integrity during transfer. Due to the flooding mechanism at network layer, the routing is as redundant as possible, thus maximising the probability of all messages reaching at least one gateway. Having reached the gateway, the messages may be processed by the gateway to reconstruct the trespasser's path if possible. Although this intuitive protocol seems very simplistic, it does perform well in terms of computational overhead on the sensor nodes and key material needed to authenticate the messages. Since every message is addressed to the data gateways, nodes need only store as many keys as there are data gateways in the network. If the assumption holds that data gateways are uncompromiseable, there even is no need for more than one key per node that is meaningful to all gateways. The keys must be unique, thus identifying a node and serving for authentication purposes. Forged or manipulated events can be detected on the gateway when the verification of MACs fails.

The obvious downfall of this protocol is its high communication overhead – each PIR event will be flooded into the network, creating an enormous amount of traffic. The protocol does not discern between trespassers that characteristically create a chain of PIR events following their chosen path through the area on one hand, and sporadic PIR events that may be triggered by animals or even wind on the other. Seeing as the gateway can still filter those sporadic events due to its knowledge of the network's topology, this does not impede the detection of trespassers; however, the communication load puts a heavy strain on the nodes' energy consumption.

The basic idea of the trespass detection protocol used in the FleGSens system is to collect PIR event messages locally before flooding an *aggregate* of $aggregate\_size$ events into the network. To this end, each node, if it registers a PIR event, broadcasts a so-called *AllEvents message* to its immediate neighbours. This message contains all – yet unsent – events the sender node knows up to then, represented by their respective timestamps and positions. The receiving nodes keep the new event(s) in their memory. An event is deleted from the memory if the node does not register or receive any new events within the next $max\_event\_lifetime$ seconds, where $max\_event\_lifetime$ is a configurable parameter of the trespass detection protocol. This maximum lifetime of an event is necessary as otherwise the protocol would collect sporadic PIR events and interpret them as trespass.

As soon as a node has thus collected $aggregate\_size$ events, it floods them cumulatedly into the network. This is referred to as flooding aggregated events. The result is a protocol behaviour equivalent to flooding the network under a certain condition – in this case the condition that there have been at least $aggregate\_size$ PIR events in the local region of the flooding node. Figure 2 shows the protocol states of the gateway; basically the gateway simply waits for *AllEvents* messages. The protocol states of the sensor nodes are illustrated by figure 3. In the *StandBy* state, the nodes wait for *AllEvents* messages broadcast to them by their neighbours. Upon receipt of an *AllEvents* message, the nodes check how many events they have stored. If the sending threshold of $aggregate\_size$ events is reached, they flood the events known to them in an *allEvents* message and return into the default – i.e. *StandBy* state.

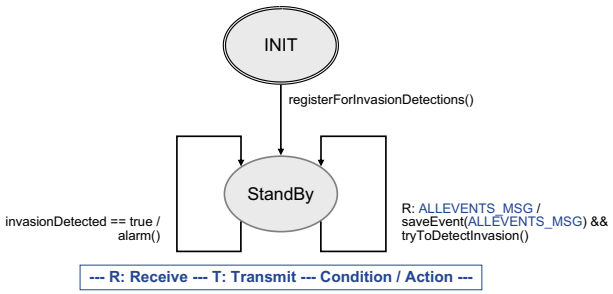


Fig. 2. States of the gateway

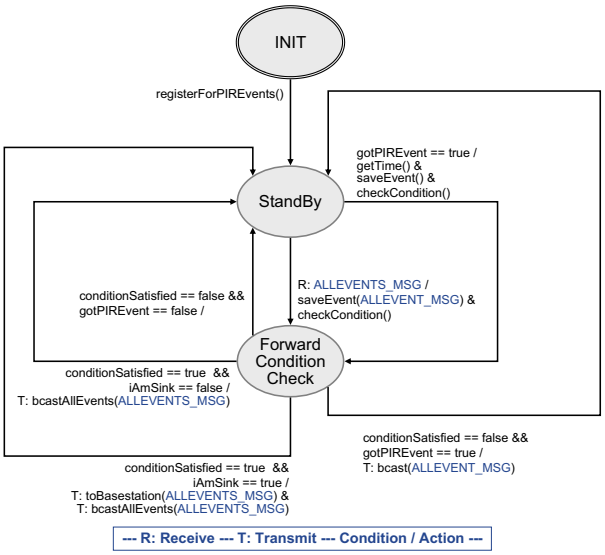


Fig. 3. States of the sensor nodes

The advantage of this protocol compared to the intuitive approach is twofold: First, the communication overhead caused by flooding the network is decimated by a factor of $aggregate\_size$ on average. Second, due to the characteristic trail of PIR events a trespasser causes *locally*, the protocol allows to detect trespassers while at the same time being able to filter sporadic events using the maximum lifetime $max\_event\_lifetime$ of an event. Note that – compared to the intuitive protocol – it is still not necessary to use more cryptographic material than one pairwise key for each node, i.e. the key they share with the gateway. Using MACs, the

gateway is able to detect forged or manipulated events, so that, on the whole, the attacker's influence is limited by the number and position of nodes they have compromised.

B. Node Failure Detection Protocol

The coverage of the trespass detection protocol is highly dependant on the number of nodes that are monitoring the desired area. As a node may fail permanently due to damage or battery depletion, thus decreasing the coverage of the tresspass detection protocol, the need for a node failure detection protocol arises. A secure node failure detection protocol enables the network maintainer to repair or replace any failed node. The basic idea of the FleGSens node failure detection consists in the nodes broadcasting *heartbeats* at certain points in time to indicate they are still up and running. Since a node is not able to detect its own failure, other (i.e. surrounding) nodes must assume this task. Hence, nodes must keep track of their neighbours' messages and report a failure if no messages were received from a neighbour for a certain period of time. As both heartbeats and messages sent to report a node failure require a share of the medium's available bandwidth, a number of measures have to be taken to avoid collisions. Primarily, collisions occur if a large number of nodes try to send a message at approximately the same time. This, for instance, is the case if all the failed node's neighbours report the failure, since in a multihop scenario that uses flooding on the network layer, each failure report must be forwarded by each node of the network. Thus, it is necessary to reduce the number of failure reports in very dense networks; it would be sufficient if at least one node reports the failure. However, since this node may be corrupted, it is important to select a subset of neighbours for each node's livelihood surveillance. In addition to the number of messages sent per time interval, the length of messages contributes to the risk of collisions. Any messages sent need an integrity and authentication mechanism to prevent a potential attacker from spoofing or replaying messages to conceal a node failure. Since the nodes in a WSN suffer from resource constraints which prohibit the use of public key cryptography and broadcast authentication mechanisms such as proposed by [6] do not perform well for many senders, authentication costs scale in $O(n)$ where n is the number of recipients of a message. This is due to the fact that for each intended recipient separate message authentication codes must be included in the message, resulting in longer messages. This is particularly relevant for heartbeats sent by all nodes at regular intervals; also, keeping track of node states for all neighbours in a dense network puts a strain on the nodes' available memory. Both authentication costs and statefulness of the protocol thus reinforce the need to select a subset of neighbours for the task of monitoring a node.

Nodes have to keep track of the keys to each node they monitor as well as to each node that monitors them. Therefore, it is reasonable to use bidirectional relations instead of unidirectional. Since the security mechanisms which are described in the next section rely on symmetric key cryptography, the keys can be used for bidirectional communication anyway. In FleGSens, the HARPS key distribution protocol [7] is used to generate symmetric pair wise keys between nodes in the network. In the following, a protocol for the secure detection of node failures is presented in which a subset of neighbours monitor the activity of one node which in turn monitors exactly that subset, resulting in bidirectional relations called *buddy relations*. The configurable parameters min_buddy_count and max_buddy_count specify a lower and upper bound for the number of buddies respectively.

Nodes periodically send *heartbeat messages* and report the failure of a buddy given the absence of a certain number of these messages. Instead of using specific *heartbeat* messages, it would be possible to utilise regular network traffic for this task, but the use of *heartbeat* messages has two advantages: First, *heartbeat* messages contain data to provide authenticity, integrity and protection against replay-attacks. Such data would otherwise have to be included in all regular traffic, leading to additional overhead in networks with more regular traffic than *heartbeat* messages. Second, *heartbeat* messages have to be sent periodically to make it possible to detect their absence. If regular network traffic lacks any peridodic quality, additional messages and application logic is be needed to avoid false positive failure detections. The proposed node failure detection protocol uses authentication mechanisms that rely on symmetric key cryptography like those in AES-CCM* provided by devices that implement the IEEE 802.15.4 standard [8]. The protocol is divided into three subsequent phases as shown in Fig. 4:

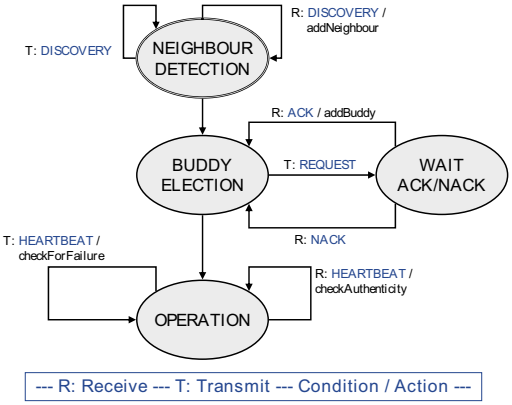


Fig. 4. Protocol phases of the node failure detection

The first phase is the *neighbour detection phase*, in which all nodes in the network periodically broadcast *discovery messages* containing their ID. A node that receives a *discovery* message inserts the contained ID into its neighbour list and keeps track of the received signal strength. Transitions to the following phases are triggered by the receipt of *control messages* sent by the gateway to all nodes in the network. Control messages are not part of the node failure detection protocol itself but belong to the main application.

The next phase is the *buddy election phase* during which the buddy relations mentioned above are formed. All messages used in this phase contain a timestamp and a message authenti-

cation code. The MAC is calculated over the payload including the timestamp, using the symmetric key shared between the sender and the receiver. This enables the receiver to determine the age, integrity and authenticity of the message.

At the beginning of the *buddy election* phase, each node first sorts its neighbour list by signal strength and then starts sending *buddy request messages* to the neighbour with the best signal strength. A *buddy request* message is resent up to 3 times until either an ACK or NACK is received as described below. Neighbours with better signal strength are favoured as buddies, seeing as it is reasonable to assume they provide a higher packet arrival ratio during the subsequent operation phase and thus produce fewer false positive failure detections.

During the *buddy election phase*, nodes iteratively send *buddy request* messages to the next node in their sorted neighbour list. The corresponding neighbour either accepts or rejects the buddy relation, depending on its current number of buddies and the result of the message authenticity check. If the neighbour has less than max_buddy_count buddies and the authenticity can be verified, a *buddy ACK* is sent, otherwise a *buddy NACK* is sent. Each node then continues sending *buddy request* messages to the next node in its neighbour list until it has at least min_buddy_count buddies or there are no more nodes left in the list. At the end of this phase, node N_i has $buddies(N_i)$ buddies which are stored in its buddy list, containing the ID and a *heartbeat counter* for each buddy; the heartbeat counter is used to keep track of the number of *heartbeat* messages that were consecutively missed.

The last phase is the *operation phase* in which all nodes perform the following steps every $heartbeat\_interval$ seconds: First, they broadcast a *heartbeat* message, then they check the *heartbeat counter* of each buddy and afterwards increase it by one. A *heartbeat* message sent by Node N_i contains a timestamp followed by $buddies(N_i)$ MACs. The k^{th} MAC is calculated over the payload including the timestamp, using the symmetric key between the sender and its k^{th} buddy. This way a node has to broadcast only one heartbeat message, which then enables all its buddies to determine the age, integrity and authenticity of this heartbeat message. Once a node receives a *heartbeat* message from one of its buddies, it checks the message integrity.

If the message's authenticity and integrity can be verified and the timestamp of the message is younger than $heartbeat_{timeout}$ seconds, the receiving node resets the according *heartbeat counter*. The timestamp check against $heartbeat_{timeout}$ is used to prevent replay attacks on *heartbeat messages*. Since the *heartbeat counter* of every node is increased by one every $heartbeat\_interval$ seconds and reset to zero every time a authentic *heartbeat* message is received, the *heartbeat counter* equals the number of consecutively missed *heartbeat* messages from the corresponding buddy. This counter is therefore used to detect node failure by comparing the counter to a fix threshold t_{hb}. The threshold t_{hb} together with the value of $heartbeat\_interval$ therefore determines the timespan needed to detect node failure, given $t_{detect} = heartbeat\_interval * (t_{hb} + 1)$. Please note that $(t_{hb} + 1)$ is used for the detection duration since a *node failure message* is created, after more than (t_{hb}) heartbeats are missing. Both the $heartbeat\_interval$ and the parameter t_{hb} should be selected according to the needs of the application. The lower t_{hb} is, the higher the amount of false positives due to regular message loss, the higher t_{hb}, the higher the time needed for detecting the node failure. After the *heartbeat counter* has reached the threshold t_{hb}, an *node failure message* is send to the gateway which contains the IDs of the reporting and failed node, the timestamp of the alert and a MAC. The MAC is calculated the same way as described above, using the key between the reporting node and the gateway.

To lower the impact of *heartbeat messages* on the remaining network traffic in terms of collisions and bandwidth utilisation which may lead to false positive detections, the process of sending *heartbeat messages* and checking the *heartbeat counter* of buddy nodes in FleGSens is organised as follows: In the FleGSens application, the duty cycle protocol enables all nodes to send and receive messages every $duty\_cycle\_interval$ but requires them to send a *heartbeat message* only every $heartbeat\_interval$. Given e.g. $duty\_cycle\_interval = 1$ sec and $heartbeat\_interval = 2$ sec, nodes split into two groups based upon their ID. Nodes with an even ID perform the steps in the even duty cycle intervals and nodes with an odd ID perform the steps in the odd intervals.

C. Further Protocols

To ensure the secure detection of trespassers reliably over a long period of time, further protocols are required which are described briefly in this section.

First, it must be ensured that all messages arrive at their destination, especially alarm messages intended for a gateway must reach it over at least one path. This can only be achieved if a path between any sending node and one or more of the gateways exists. Hence, a protocol must detect if the network is partitioned, the so-called *partition detection protocol*. It operates in a way that gateways periodically exchange messages and check if and via which nodes the message from the other gateway arrived.

Second, realising a network lifetime larger than a few days despite the limited energy of sensor nodes requires saving energy. As radio and processor consume most of the energy, it is sufficient to turn off these components while the PIR sensor remains active and wakes up the processor by an interrupt in case it detects a moving object. Periodically, the radio is turned on for message exchange. A duty cycling protocol organises the duty cycles of all nodes in a way that neighbours can always communicate and the end-to-end delay is minimised.

With coordinated duty cycles, the need for synchronised clocks arises. The time synchronisation process is initiated by the gateway, which first synchronises all its neighbours. Synchronised nodes again synchronise their neighbours until the whole network is synchronised. The synchronisation protocol includes mechanisms based upon the TESLA protocol [6] to ensure message authenticity, as well as methods for outlier detection to increase overall synchronisation accuracy.

Since the events generated by the trespass detection protocol need location information which then can be displayed at the

gateway, a localisation protocol is needed. To this end, the SHOLOC protocol described in [9] was chosen. It features hop based authentication using the TESLA protocol [6] similar to the aforementioned time synchronisation protocol.

V. Simulation

A. Simulation Framework

To simulate the implemented protocols, the network simulator Shawn [10] was used. Shawn has built-in support for the hardware platform that is used in the real-world experiments. This makes application development more efficient, since the protocols and applications can be written once and then compiled for the simulator and the hardware using the same code. The simulator further includes a CSMA Transmission model that is capable of simulating the IEEE 802.15.4 [8] radio interface used by the hardware platform. The CSMA model was parameterised to further fit the characteristics of the hardware, the communication range was set to 30m and link quality was modelled based upon extensive measurements.

B. Scenario

To analyse the functionality and scalability of the proposed protocols, they were tested in two scenarios of different size as shown in Table II. While *scenario A* contains 200 nodes, *scenario B* comprises 2000 nodes and is used to show the scalability of the protocols. In each scenario, the nodes were aligned in a grid with fixed distances between rows and columns; the simulations were repeated using 20 different randomly generated seeds. To test the performance of the trespass detection protocol, the passive infrared sensor was modelled to generate alarms caused by virtual trespassers. The virtual trespasser moves along configurable waypoints and generates events at a node in the network at the time they enter the area monitored by its PIR sensor.

	Scenario A	Scenario B	Prototype
Number of nodes	200	2000	16
Rows * Columns	4x50	8x250	4x4
Row / Column distance	7.5m/7.5m	7.5m/7.5m	4m/5m
Diameter	13 hops	65 hops	1 hop
Runtime	2 hours	2 hours	1 hour

TABLE II
Selected scenario sizes

As introduced in section IV-A, the trespass detection protocol uses two configurable parameters: the maximum lifetime $max\_event\_lifetime$ of an event and the flooding condition $aggregate\_size$ that determines the number of events a node has to collect before flooding an *AllEvents* message into the network. For Scenario A, $max\_event\_lifetime$ was set to 11 seconds, scenario B uses a $max\_event\_lifetime$ of 30 seconds whereas for the prototoype, $max\_event\_lifetime$ was set to 10 seconds. Bot the small scenario A and the prototype aggregate 2 events locally before flooding the aggregate into the network, whereas Scenario B uses the $aggregate\_size$ parameter set to 3 events.

As for the node failure detection protocol, all nodes in the network try to form buddy relations to a total number of 3 buddies, and, in the operation phase, periodically send heartbeats every 2 seconds, reporting the failure of one of their buddies if more than $t_{hb} = 9$ heartbeats appear to be missing. At a fixed time, 10% of the nodes are selected randomly and switched off simultaneously to evaluate the time needed for failure detection. All reported failures are recorded and compared to the list of nodes that have actually been deactivated to evaluate the number of false positive alarms. Table III shows how the parameters were chosen for the different simulation scenarios. The results of all simulative tests are described in the following section.

	Scenario A	Scenario B	Prototype
$max\_event\_lifetime$	11	30	10
$aggregate\_size$	2	3	2
min_buddy_count	3	3	3
max_buddy_count	7	7	7
$heartbeat\_interval$	2	2	2
t_{hb}	9	9	9
Number of failed nodes	20	200	2

TABLE III
Protocol parameters

C. Results

To evaluate the trespass detection protocol, a total number of 50 different paths representing a trespasser through the monitored area were defined. Three basic categories of paths were identified: *Straight paths* cross the area the shortest way possible from one side to the other. *Diagonal paths* still follow a straight line, though not the shortest there is, from one side of the area to the other. *Complex paths* may be completely random, even leaving the area on the same side as entering it. In each simulation run, a trespass every 100 seconds was simulated to avoid concurrency effects during the detection. The protocol was evaluated under the following aspects:

- Reliability of detection: Number of correctly detected trespasses
- Communication load: Number of messages needed per PIR event
- Robustness against false events: Number of false alarms
- Robustness against attackers: Number of correctly detected trespasses in the presence of an attacker

The attacker is modeled to launch several attacks according to their objective as described in section III. The first attack consists in letting compromised nodes drop any and all *AllEvents* messages they receive. Furthermore, if a compromised node detects a PIR event, it will not broadcast the according *AllEvents* message to its neighbours, thus trying to prevent the network from detecting a trespass successfully. This attack is called *silence attack*.

The second attack considers the goals of delaying and and adulterating tresspass detections respectively. To this end, the attacker manipulates timestamps and location information in *AllEvents* messages sent by compromised nodes. Note that – as compromised nodes know the cryptographic material – the compromised nodes recalculate the MACs accordingly. Thus, manipulations cannot be detected by MAC verification failure. The attack is referred to as an *manipulation attack*.

The third attack modeled in the simulation environment aims to destroy *AllEvents* messages. The attacker randomly manipulates the information contained in every *AllEvents* message their compromised nodes send or forward. The attacker does not recompute the MACs; thus, they are able to affect a larger number of messages than using the manipulation attack. This attack is called *destruction attack*.

Table IV shows the protocol's performance in terms of detection reliability; the figures indicated by the ± symbol identify the 95% confidence interval. In each run, 50 trespasses were simulated; all trespasses were correctly detected.

Trespasses	Correctly detected trespasses	Number of alarms
50	50	60.5 ± 0,266

TABLE IV
TRESPASS DETECTION

Note that on average, there is more than one alarm raised by the base station per trespass. This is due to the fact that the base station was configured to detect a valid path through the area if time and location of three PIR events can be reasonably connected. Thus, especially considering the categories of diagonal and complex paths where paths are longer than in the straight path category, there is the possibility of detecting several *sections* of one trespass. The more connectible PIR events are assumed, the less redundancy is achieved. However, this also increases the risk to falsely not detect a trespass due to lost *AllEvents* messages.

Table V shows how many PIR events were registered on the nodes per simulation run; the figures indicated by the ± symbol identify the 95% confidence interval. Out of 2660 PIR events, on average 26.85 events were lost due to message loss during the transfer of the according *Allevents* messages. Message losses were caused by collisions on the link layer.

PIR events	Lost events
2660	26,85 ± 0,259
Delivered PIR events	**Delay [s]**
2633	4.85 ± 0.141

TABLE V
EVENTS DURING TRESPASS DETECTION

Furthermore, table V shows that the average time delay between the registration of a PIR event as a hardware interrupt and the receipt of an *AllEvents* message containing the event at the gateway amounted to 4.85 seconds. All events not lost during transfer were reported to the gateway in less than five seconds.

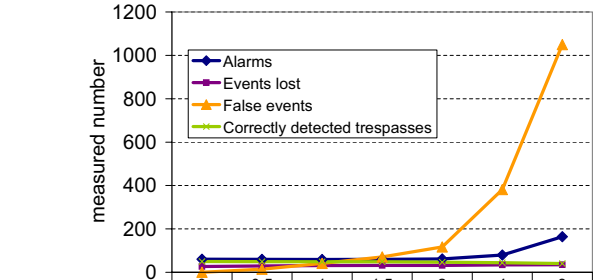
Table VI shows the simulation result in terms of communication load; figures indicated by the ± symbol identify the 95% confidence interval. The table shows the overall number of messages transmitted in the network per PIR event. As the simple protocol floods all event messages into the network directly, the figure represents exactly all events being sent by each node once.

As can be read from the table, communication load is reduced by factor two using the localised protocol as described in section IV. This is due to the fact that each node, before flooding the network, has to collect $aggregate\_size = 2$ events locally.

Protocol	PIR events	Flooded msgs.	msgs./PIR event
Simple	266	53200	200
Localised	266	25019.7 ± 49.573	95.05

TABLE VI
COMMUNICATION LOAD DURING TRESPASS DETECTION

The simulation results in terms of robustness to sporadic events are shown in Fig. 5. The false event rate was varied between 0 and 8 events per two hours runtime. It is assumed that the statistical probability of false events follows a uniform distribution and is the same on all nodes in the network.



	0	0,5	1	1,5	2	4	8
Alarms	60,5	59,6	59,1	59,5	61,5	79,35	164,35
Events lost	26,85	29,25	31,25	31,5	31,65	33,95	34,75
False events	0	14,5	41,3	71,15	117,1	381,6	1049,9
Correctly detected trespasses	50	49,05	48,45	48,2	46,5	44,3	40,3

False events/node*2h runtime

Fig. 5. Influence of false events on trespass detection

The figure shows that up to about 2 false events per two hours and node the influence on the trespass detection protocol is relatively low. The number of alarms falls with rising false event rate. This is due to the fact that the more events are produced, the more events are lost – as shown by Fig. 5. Especially considering the category of straight paths, this affects detection, since, in that case, only few events are even registered. Consequently, the number of correctly detected trespasses decreases as the false event rate rises. If the false event rate is higher than 2 false events per two hours, the number of alarms rises dramatically; however, numerous outdoor measurements were performed using the hardware confirmed a maximum false event rate of 2 false events per two hours, putting the large figures into perspective again.

Table VII summarises the effects of an attacker as specified above with 5% compromised nodes; the figures indicated by the ± symbol identify the 95% confidence interval. As above, the nodes aggregate two events before broadcasting; the base station was configured to detect a trespass if it is able to connect three individual events according to their timestamp and locality information.

While the manipulation attack – wherein the attacker only changes messages sent by compromised nodes – is virtually effectless, the table shows different results with regard to the *silence* and *destruction* attacks. The attacker influences

Attack	Alarms	PIR events lost	Detected trespasses
Silence	51,4 ± 1,019	42,6 ± 2,129	43,4 ± 1,093
Manipulation	60,55 ± 0,233	26,85 ± 0,313	50 ± 0
Destruction	53,5 ± 0,713	34,55 ± 1,267	45,9 ± 0,696

TABLE VII
TRESPASS DETECTION IN THE PRESENCE OF AN ATTACKER

relatively many messages, thus increasing their chance to impede trespass detection. On average, 6.6 and 4.1 trespasses could not be correctly detected respectively. In the 4x25 node scenario, this is to be expected: if three events must be connected to form a valid path and the shortest path across the area contains only four events, the loss of a single event is already critical for the detection. Since *AllEvents* messages always contain two events and both are lost or destroyed by the attacker, the effect is noticeable especially for straight paths. The simulations were repeated with a base station configured to connect two instead of three events to an alarm and could thereby validate this reasoning: all trespasses were correctly detected using the new configuration.

The results of the node failure detection protocol tests are summarised in Table VIII. Since scenario B is formed using 8 instead of 4 rows like scenario A, this leads to nodes having 24 neighbours in scenario A and 34 neighbours in scenario B. The desired number of 3 buddies was achieved by almost all nodes in both scenarios. One significant difference between both scenarios is the number of reported false positive node failures. The simulation of scenario A and B incorporates the duty cycle management and additional protocols mentioned in Section IV-C which are running at the same time. This has impact on the packet loss and thus the number of false positives. The average number of false positives generated in scenario A is 0,1 and 16,8 in the case of scenario B. The higher number of false positives in the scenario comprising 2000 nodes is a result of the higher number of neighbours. This leads to the increase of concurrent network traffic and the hidden terminal problem, thus leading to increased collisions.

	Scenario A	Scenario B
Average number of neighbours	24	34
Average number of buddies	2,9	3,0
Reported false positives	0,1	16,8

TABLE VIII
NUMBER OF NEIGHBOURS, BUDDIES AND FALSE POSITIVES IN SCENARIOS A AND B

Fig. 6 shows the time needed for a node to report the failure of one of its buddies to the gateway, which in both scenarios varies between 18 and 22 seconds. The time needed to report the failure to the gateway depends on the parameters of the failure detection protocol, $heartbeat\_interval$ and t_{hb}. The expected duration is the product of both values and thus equals 20 seconds in the simulation scenarios. Depending on the timespan between a node's last *heartbeat message* and its deactivation, this detection duration may vary as follows: If the node is deactivated immediately before it sends its next *heartbeat message*, the failure may already be detected after $t_{detect} - heartbeat\_interval = 18$ seconds. If, on the other hand, the node is deactivated immediately after it sends its last *heartbeat message*, the failure may be detected after $t_{detect} + heartbeat\_interval = 22$ seconds in the given scenario.

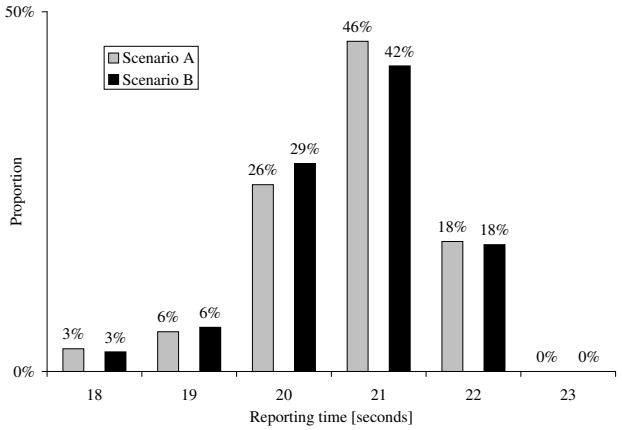


Fig. 6. Node failure reporting time, Simulation of scenario A and B

VI. PROTOTYPE EXPERIMENTS

A. Hardware

In order to demonstrate the effectiveness of the presented protocols, they were tested on iSense sensor nodes from coalesenses [11]. The iSense nodes use a 16MHz 32 Bit RISC wireless controller (JN5139) and have 120 KB of external flash and 90 KB RAM. Their radio operates in the 2.4 GHz band and is IEEE 802.15.4 [8] compliant with a data rate of 250 kbit/s and provides hardware AES encryption. Furthermore, the Security Sensor Module comprising a passive infrared (PIR) sensor was applied. The PIR sensor AMN14112 is capable of detecting moving objects whose temperature differs from that of their environment in distances of up to 10 meters within an 110° angle and will be used for trespasser detection. Field tests have shown that the detection range increases to up to 15 m with larger temperature difference, higher speed and bigger size of the object.

Real-world deployment of sensor networks is a difficult task as described in [12] and memory and debugging possibilities are limited on the hardware platform when compared to the simulator. Due to this, a step-by-step approach towards the intended prototype of 200 nodes was chosen. In the next section, the first prototype scenario is described. It comprises 16 nodes and features the trespass detection and the node failure protocol. Based upon the findings in this first prototype, the gathered data were used to modify the simulation parameters used in Section V. Using these new parameters, simulations of the prototype scenario were run and the results were directly compared with the prototype.

B. Scenario

The first prototype comprised 16 sensor nodes and was located in the University of Lübeck college sports gymnasium.

Since this was the largest available test area, the scenario parameters described in Section V were adapted to fit the new circumstances. As shown in Table II, the nodes were placed in a 4x4 grid with distances from 4 to 5m between the nodes, thus forming a single-hop scenario. The marked node in the second row on the left side in Fig. 8 was selected as a gateway node. The gateway was connected to a laptop and used for monitoring the network. As shown in Fig. 7, the nodes were placed about 20cm from the ground on glass cylinders to limit the effect of ground features on the radio interface.

Fig. 7. Nodes at the prototype site

To test the functionality of the trespass detection protocol, several paths across the monitored area were defined, varying in their direction and their start and end position regarding the network topology. Three basic categories of paths are identified: straight paths are paths that choose the shortest way possible from one end to the other end of the area; diagonal paths cross the area approximately diagonally; U-formed paths enter the area and leave it on the same side. Fig. 8 shows examples for each path category.

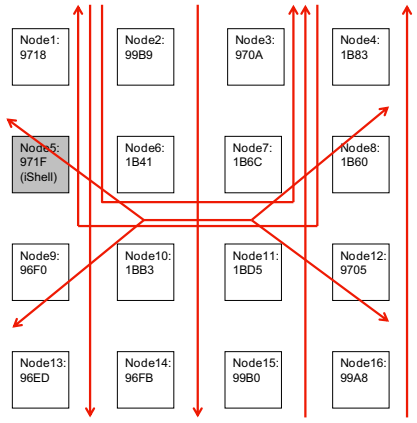


Fig. 8. Location nodes in the prototype and paths across the covered area

To conduct the experiments, a test person repeatedly performed a total of eight trespasses, four of which followed straight paths, two followed diagonal paths and two followed u-form paths through the area. The trespasser varied his speed during the experiment to show the flexibility of the trespass detection protocol. Additionally, the protocol was tested under heavy strain in order to ensure correct detection is possible *after* a phase of increased protocol activity and thus, to show that there are no unwanted side effects. For this purpose, several persons entered the area for a duration of three minutes, moving around randomly.

For the node failure detection protocol, eight field experiments in the scenario described above were run, each featuring a runtime of 1 hour. During the tests, the number of buddies that were elected per node, their IDs and information indicating whether the buddy relations were formed bidirectionally were recorded. During the operation phase, the messages sent by all nodes were logged to measure packet loss and false positive detections of node failures. At the end of the field experiment, two nodes in the network were deactivated to test the correctness of the failure detection and measure its duration.

To show the result of attacks on the described protocols under real-world conditions, the possibility to select nodes in the network and let them perform certain attacks was added. In the case of the node failure detection protocol, the selected node $N_{attacker}$ was told to perform a replay attack on the heartbeat of a node in its neighbourhood (N_{victim}). After this command was sent to the node, it began monitoring and recording the heartbeats from N_{victim}. Afterwards, N_{victim} was deactivated and $N_{attacker}$ performed the attack by replaying the recorded heartbeats, beginning shortly before the other nodes would be detecting the nodes failure due to the selected threshold t_{hb}. The time of the actual failure of N_{victim} and the time it took for its buddies to detect the failure with $N_{attacker}$ performing the replay attack were logged. The value given for $heartbeat_{timeout}$ as shown in Table III was selected considerably big to enable the demonstration of replay attacks. Depending on the accuracy of the time synchronization protocol, $heartbeat_{timeout}$ can be selected as low as the maximal time difference between two neighbouring nodes.

After the real-world experiments were finished, the collected data was analysed as described in the next section. The simulation parameters were improved based upon these results and repeated simulations with the same settings used in the prototype. These simulations comprised 16 nodes, had a runtime of 1 hour and were performed using the same 20 seeds used in the simulations described in Section V. The next section presents the results of the prototype and the modified simulations.

C. Results

During the experiment, each of the eight specified trespasses was performed 20 times. The trespass detection protocol was configured with $aggregate\_size = 2$, meaning that when a node knows of two PIR events, it floods the network with an *AllEvents* message.

With respect to the comparability of real-world and simulation result, it is foremost interesting which of the nodes register the according PIR events. Fig. 9 shows an example of a diagonal path; the simulation result is shown in the network sketch to the left of the separation bar, the figure to the right illustrates the prototype result.

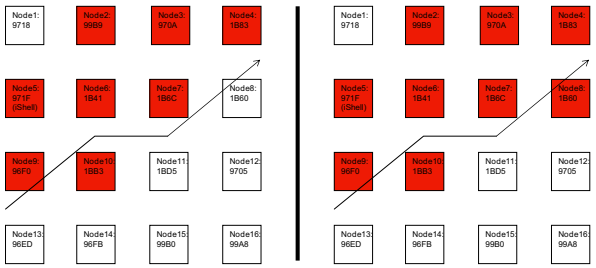


Fig. 9. Exemplary comparison of simulation and prototype result

The marked nodes registered a PIR event during the trespass. The figure shows that simulation and prototype behaviours only differ in the reaction of one node, which is a very convincing result. Both simulation and prototype experiment showed very little variance per path through the network, regarding which nodes detect a PIR event. This behaviour was observed regardless of the path category.

Furthermore, in both the prototype and the simulated scenarios, all trespasses were correctly detected. Due to the single-hop quality of the scenario, both the broadcast and flooded *AllEvents* messages were immediately received by the data gateway. This means the alarm was raised the same time the gateway received enough *AllEvents* messages; therefore, no detection delay was introduced by the flooding mechanism in this scenario.

The results of the *buddy election phase* were analysed to determine how many buddies each node elected and whether the buddy relations are bidirectional. Fig. 10 shows the distribution of the number of buddies that were elected by the nodes. While only 2% of the nodes have 2 buddies and 5 to 9% have 4 buddies, almost all nodes in the network have the desired number of 3 buddies. Since nodes stop sending requests to their neighbours if they already have the desired number of min_buddy_count buddies, there is only one possibility to accept more than min_buddy_count buddies: this happens if node N_a sends a *buddy request* message to N_b and then receives a *buddy request* message from another node N_c before it receives the *buddy ACK* from N_b. This leads to N_a becoming the buddy of both N_b and N_c, thus increasing the number of buddies of N_a by two. As shown in Fig. 10, 2% of the nodes end up with a number of 2 buddies. This happens if all neighbours surrounding a node already have the desired number of buddies before that specific node sends a *buddy request* message to them, causing the neighbour to reject the *buddy request* message. The analysis of the *buddy election* phase further showed that all buddy relations were bidirectional and that the results of the prototype and simulation are almost identical.

Fig. 11 shows the time needed for a node to report the

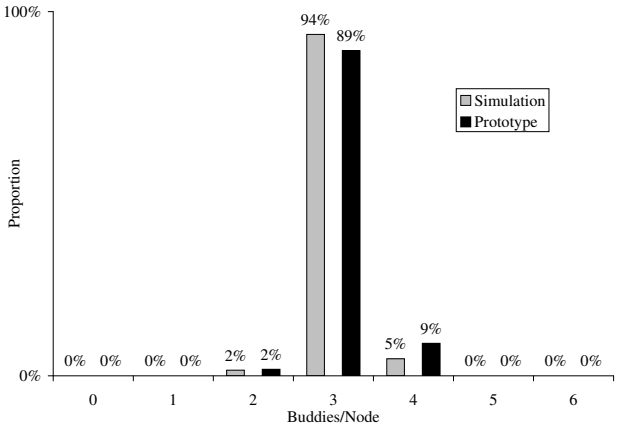


Fig. 10. Buddies per Node, Simulation and Prototype

failure of one of its buddies to the gateway. Detection duration ranges between 18 and 22 seconds in the prototype and 19 and 23 seconds in the simulation results. Both prototype and simulation show the same distribution of reporting times, which corresponds to the expectations based on the simulation results in Section V. A closer look at both distributions reveals that the real-world distribution is shifted towards shorter detection durations by a little less than approx. 1 second in comparison to the simulation. This is the result of measuring inaccuracy that is inevitable in real-world experiments.

The analysis of the *node failure message* and the remaining operation phase further showed that 100% of the node failures were detected by all corresponding buddies and that no false positive alarms were generated in the prototype scenario.

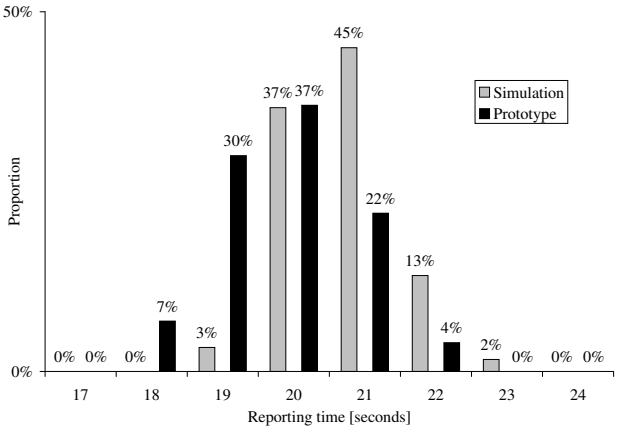


Fig. 11. Node failure reporting time, Simulation and Prototype

Table IX depicts the impact on replay attacks on the node failure detection duration. In the prototype, the detection duration ranges from 18 to 22 seconds with an average reporting duration of $19,87$ seconds. In the experiments, node $N_{attacker}$ began performing replay attacks at $t_{attack_start} = 15$ seconds after it heard the last *heartbeat* messages from N_{victim},

presuming that the buddies of N_{victim} might detect its failure at 18 seconds. The buddies of N_{victim} afterwards receive the replayed *heartbeat* messages which pass the authenticity tests imposed on the MAC fields but fail the age check against $heartbeat_{timeout}$ after some time. This can be seen in Table IX, where the detection duration in the presence of a replay attacks ranges from 33 to 37 seconds with an average reporting duration of $34,71$ seconds, suffering from a increase of approx. 15 seconds caused by the replay attack. The timespan is the result of the time $t_{attack_{s}tart} = 15$ seconds which is selected by the attacker dependant on $heartbeat_{timeout}$ and t_{detect}.

It has been shown that an attacker can only delay the node failure detection for up to an upper bound that is mainly dependant upon the $heartbeat_{timeout}$ value and the values chosen for t_{detect}. It is impossible for the attacker to delay the detection any further since they are unable to forge *heartbeat messages* without the knowledge of the symmetric keys between N_{victim} and each of its buddies.

	Without replay attacks	With replay attacks	Difference
Minimum	18	33	+15
Maximum	22	37	+15
Average	19,87	34,71	+14,84

TABLE IX
EFFECT OF REPLAY ATTACKS ON NODE FAILURE REPORTING TIME

VII. CONCLUSION

This paper presents FleGSens – a wireless sensor network for the surveillance of critical areas. The FleGSens application provides several protocols that ensure the secure detection of trespassers even if an attacker compromises a limited number of sensor nodes. Protocol performance was investigated first by an extensive set of simulations in different scenarios and then by application in a test bed. In general, the results of the simulations and the real-world experiments closely resemble each other. The trespass detection protocol shows the location of all trespassers independent of their paths as soon as the duty cycle allows for sending messages. The node failure detection protocol assures that all nodes monitor a subset of neighbours and detects all node failures once the threshold of missed messages is exceeded. The parameters are adjustable to the requirements of the application to achieve lower detection durations, lower false positive rates and supports the use of duty cycling. Attacks are either detected by non-compromised nodes or they have no effect on the employed protocols.

Currently the testbed is being extended for further outdoor experiments and to test the protocols in larger real-world scenarios comprising up to 200 nodes. A solar panel, lithium ion battery, charge controller module and a waterproof case will be added to support long-term testing in the outdoor environment.

ACKNOWLEDGMENT

This research is part of the FleGSens project funded by the german Federal Office for Information Security (BSI). FleGSens is a joined project between the Institute of Telematics of the University of Lübeck, the Institute of Telematics of the University of Karlsruhe and the coalesenses GmbH.

REFERENCES

[1] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983, iSSN: 0018-9448.

[2] C.-Y. Lin, W.-C. Peng, and Y.-C. Tseng, "Efficient in-network moving object tracking in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, pp. 1044–1056, 2006.

[3] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A line in the sand: a wireless sensor network for target detection, classification, and tracking," *Computer Networks*, vol. 46, no. 5, pp. 605 – 634, 2004, military Communications Systems and Technologies. [Online]. Available: http://www.sciencedirect.com/science/article/B6VRG-4CXKNBT-1/2/38791b700b4804d30e9df8aca23712e7

[4] Y. Xu, J. Winter, and W.-C. Lee, "Dual prediction-based reporting for object tracking sensor networks," in *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, Aug. 2004, pp. 154–163.

[5] H. Yang and B. Sikdar, "A protocol for tracking mobile targets using sensor networks," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, May 2003, pp. 71–81.

[6] A. Perrig, R. Canetti, D. Tygar, and D. Song, "The tesla broadcast authentication protocol," *Cryptobytes*, vol. 5, no. 2, pp. 2–13, 2002, http://www.rsasecurity.com/rsalabs/cryptobytes/.

[7] Mahalingam Ramkumar and Nasir Memon, "Harps: Hashed random preloaded subset key distribution," *Cryptology ePrint Archive*, 2003. [Online]. Available: citeseer.ist.psu.edu/727910.html

[8] IEEE 802.15 Working Group, "IEEE 802.15 WPAN Task Group 4 (TG4)," http://www.ieee802.org/15/pub/TG4.html.

[9] Y. Zeng, S. Zhang, S. Guo, and X. Li, "Secure hop-count based localization in wireless sensor networks," in *CIS '07: Proceedings of the 2007 International Conference on Computational Intelligence and Security*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 907–911.

[10] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer, "Shawn: A new approach to simulating wireless sensor networks," in *Design, Analysis, and Simulation of Distributed Systems 2005 (DASD' 05)*, Apr. 2005, pp. 117–124.

[11] C. Buschmann and D. Pfisterer, "iSense: A modular hardware and software platform for wireless sensor networks," 6. Fachgesprch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Tech. Rep., 2007. [Online]. Available: http://ds.informatik.rwth-aachen.de/events/fgsn07

[12] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli, "The hitchhiker's guide to successful wireless sensor network deployments," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 43–56.