University of Technology Darmstadt, Germany
Department of Computer Science
IT Security Group

Diploma Thesis

A hybrid scheme for filtering injected false data in heterogeneous sensor networks

Nils Knappmeier

16th January 2007

Supervisor:
Prof. Dr. Claudia Eckert
Dipl.-Inform. Christoph Krauß

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, am 16. Januar 2007

Many thanks go to

- my supervisor Christoph for the excellent support and for the comments on each chapter of the thesis,
- Johannes, who read almost the whole thesis and suggested many formal improvements,
- my dad for proof-reading the whole thesis and my mum for proof-reading the german abstract,
- Julia, Gitte, Jacqueline, Svenja, Peter, Sebastian, Nia, and Astrid for finding all the grammatical and linguistical mistakes and spelling errors,
- Arne for donating CD covers,
- Jörg and Olena for helping me print onto the CD,
- Jan for reading the chapters one to four. Unfortunately there wasn't time to integrate the corrections.

This thesis has been typeset with LaTeX 2_ε and written with the editor `Kile`. Graphs and figures have been created with the programs `Graphviz`, `Inkscape`, `Xfig` and `GNUplot`. For the implementation, the Java `J2SE™ 5.0` Development Kit by Sun Microsystems and the `Eclipse` development platform have been used.

[1] http://creativecommons.org/licenses/by-nc/2.0/

Zusammenfassung

Sicherheit in der Informationstechnik ist heutzutage ein wichtiges Teilgebiet der Informatik. Diese Arbeit behandelt einige spezielle Sicherheitsprobleme im Bereich der sogenannten „Drahtlosen Sensornetze“.

Drahtlose Sensornetze sind ein aktuelles Forschungsthema im Gebiet des „Pervasive and Ubiquitous Computing“. Ein Sensornetz besteht aus tausenden von Sensorknoten, die z.B. in Katastrophengebieten, zur Umweltkontrolle oder in Überwachungs- und Alarmanlagen eingesetzt werden können. Ein Sensorknoten besteht aus einer CPU, Speicher, einer Funkeinheit, einer Sensoreinheit und einer Batterie zur Stromversorgung. Nach der Aktivierung bilden die Sensorknoten selbständig ein Ad-Hoc-Funknetzwerk über das Messdaten in Form von Berichten an eine Basisstation, genannt Senke, weitergeleitet werden. In vielen Szenarien werden Berichte nur dann erzeugt, wenn die Messung eines Sensors einen kritischen Wert annimmt. Zusätzlich zu dem eigentlichen Messwert kann der Bericht noch einen Zeit- und eine Ortsangabe des Ereignisses enthalten.

Aufgrund der begrenzten Energieversorgung und aus Kostengründen verfügen Sensorknoten in der Regel nur über eingeschränkte Rechen- und Speicherkapazitäten. Dieser Ressourcenmangel ist maßgebend für die Herausforderungen im Bereich „Sicherheit in Sensornetzen“. Ein Angreifer verfügt in der Regel über mehr Kapazitäten als ein einzelner Sensorknoten. Die Implementierung asymmetrischer Kryptographie ist aufgrund des Ressourcenmangels schwierig, wenn nicht unmöglich. Ausserdem sind Sensorknoten oft physisch angreifbar, da sie in Gebieten ausgelegt werden, zu denen ein potentieller Angreifer Zugang hat. Manipulationssichere Hardware wird in Sensorknoten aus Kostengründen normalerweise nicht verwendet. Das bedeutet, dass ein Angreifer, der physikalischen Zugang zum Sensorknoten hat, Teile des Programms verändern oder geheime kryptographische Schlüssel aus dem Speicher des Knotens auslesen kann. Verschlüsselungs- und Authentifizierungsmechanismen können auf diese Weise umgangen werden. Der Angreifer kann die gewonnenen Schlüssel dazu einsetzen, sich gegenüber dem Netzwerk zu authentifizieren und dann den Netzwerkverkehr abzuhören oder falsche Berichte in das Netz einzuschleusen.

In der vorliegenden Arbeit wird ein Verfahren beschrieben, dass folgende Angriffe behandelt: Erstens versucht ein Angreifer, falsche Nachrichten in das Netz einzuschleusen, um damit einen Fehlalarm auszulösen. Zweitens versucht er, Berichte, die von anderen Sensorknoten erzeugt wurden, zu verfälschen. Drittens ist es sein Ziel, die Energiereserven von weiterleitenden Knoten anzugreifen, indem er eine große Menge von Nachrichten einschleust. Dabei wird davon ausgegangen, dass der Angreifer Zugriff auf die Hardware der Sensorknoten hat und mit einem gewissen Zeitaufwand geheime Schlüssel von einzelnen Knoten extrahieren kann.

Das Ziel des im Rahmen dieser Arbeit entwickelten Systems ist, diese Angriffe zu verhindern bzw. den Aufwand, den der Angreifer betreiben muss, zu maximieren. Die Auswirkung eines physikalischen Angriffs soll auf das Gebiet

im Netz beschränkt bleiben, in dem Sensorknoten kompromittiert wurden. Wird ein größerer Bereich des Netzes kompromittiert, so soll die Leistung des Netzes kontrolliert zurückgehen, d.h. ein Angreifer, der an einer bestimmten Stelle im Netz Knoten kompromittiert hat, soll nur Berichte fälschen können, die sich auf ein Ereignis an der gleichen Stelle beziehen.

Das beschriebene Verfahren basiert auf einer heterogenen Netzstruktur. Zusätzlich zu den ressourcenschwachen „kleinen Knoten“, die den oben genannten Einschränkungen unterworfen sind, wird eine kleine Anzahl „großer Knoten“ eingesetzt, die über eine schnelle CPU, viel Speicher, eine Funkeinheit mit größerer Reichweite und eine unbegrenzte Energieversorgung verfügen. Auf der Basis eines solchen Netzwerks wird ein hybrides Schema zur Nachrichtenfilterung entwickelt, das asymmetrische Kryptographie auf den großen Knoten und symmetrische Algorithmen auf den kleinen Knoten einsetzt. Dieses Schema trägt den Namen „hybrid en-route filtering scheme“ (HEFS).

Um zu verhindern, dass ein kompromittierter Knoten alleine gefälschte Berichte einschleust, werden diese von einer Gruppe mehrerer Sensorknoten gemeinsam erzeugt. Jeder Knoten dieser Gruppe bestätigt den Bericht mithilfe von „Message Authentication Codes“ (MAC) und ein Bericht ist nur dann gültig, wenn eine vorgegebene Anzahl von MACs angehängt wurde. Ein großer Knoten dient als Zwischenstation, prüft die angehängten MACs, signiert den Bericht im Falle einer erfolgreichen Prüfung mit einer asymmetrischen ditigalen Signatur und schickt die Nachricht dann weiter Richtung Senke. Auf dem Weg zur Senke wird die Nachricht sowohl von kleinen als auch von großen Knoten weitergeleitet. Jeder große Knoten, der am Routing beteiligt ist, prüft die digitale Signatur.

Die Sensorknoten, die den Bericht erzeugen, verwenden zwei verschiedene MACs, um die Integrität der Nachricht zu sichern: Die LMAC wird von dem großen Knoten geprüft, der die Nachricht digital signiert. Die SMAC wird zusammen mit der digitalen Signatur von der Senke überprüft, um sicherzustellen, dass kein kompromittierter großer Knoten alleine eine gültige Nachricht erstellen kann. Für den Fall, dass zwar eine gültige Signatur, aber eine falsche SMAC vorliegt, lässt sich die Zahl der verdächtigen Knoten stark einschränken. Über eine Langzeitanalyse solcher „degenerierter Berichte“ versucht die Senke schließlich, kompromittierte Knoten im Netz zu erkennen und zu entfernen.

Die Schlüssel, die für die Erzeugung der MACs verwendet werden, besitzen nur einen eingeschränkten örtlichen Gültigkeitsbereich. Auf diese Weise wird verhindert, dass ein Angreifer die an einer Stelle im Netz gewonnenen Schlüssel benutzt, um gültige Berichte zu erzeugen, die eine andere Ortsangabe enthalten. Weiterhin sorgt die gruppenbasierte Berichterzeugung dafür, dass der Angreifer eine Mindestzahl von Knoten kompromittieren muss, um gefälschte Berichte zu erzeugen.

Wenn ein Angreifer Nachrichten einschleust, um die Energieversorgung von Knoten anzugreifen, ist für den Gesamtenergieverbrauch des Netzes ausschlaggebend, wie viele Knoten eine solche Nachricht weiterleiten. In HEFS wird eine eingeschleuste Nachricht erkannt und herausgefiltert, sobald sie von einem großen Knoten empfangen und überprüft wird. Um die Auswirkungen ei-

nes solchen Angriffs zu ermitteln, wurde HEFS als Proof-Of-Concept in einem Java-basierten Simulator implementiert. Das Einschleusen von Falschnachrichten wurde ebenso simuliert wie die Erzeugung von echten Berichten. In den Simulationen wurde der Energieverbrauch des Sensornetzes gemessen und mit dem Verbrauch eines simulierten Sensornetzes ohne HEFS verglichen. Als Vergleichsnetz diente sowohl ein homogenes Netz ohne große Knoten als auch ein heterogenes Netz.

Die Auswertung zeigt, dass ein Netz mit HEFS dann weniger Energie verbraucht als ein heterogenes Netz ohne HEFS, wenn die Anzahl der eingeschleusten Nachrichten mindestens um zehn Prozent höher ist als die Anzahl der Berichte, die wirklich aufgrund von Messwerten erzeugt wurden. Dieser Auswertung liegt ein Szenario zugrunde, in dem eine Nachricht durchschnittlich 88-mal weitergeleitet wird, bevor sie die Senke erreicht, und in dem hundert große Knoten (und 3900 kleine Knoten) eingesetzt werden. Im Vergleich zum homogenen Netzwerk verbraucht HEFS im gleichen Szenario schon dann weniger Energie, wenn etwa 30 Prozent der Nachrichten gefälscht sind.

Vergleicht man Netzwerke, die kein HEFS einsetzen, untereinander, so stellt man fest, dass der Energieverbrauch allein schon durch den Einsatz großer Knoten gesenkt wird. Dies ist auf die größere Funkreichweite und die angenommene unbegrenzte Energieversorgung dieser Knoten zurückzuführen.

Das beschriebene Verfahren beinhaltet außerdem einen Algorithmus zur Langzeitanalyse degenerierter Berichte. Ein Angriff, der durch den Einsatz eines Analysetools erst möglich wird, besteht darin, gezielt degenerierte Berichte einzuschleusen, die die Senke dazu verleiten, unkompromittierte Knoten aus dem Netzwerk zu entfernen. Der von uns vorgestellte Algorithmus ermöglicht die Erkennung eines kompromittierten großen Knotens anhand von drei degenerierten Berichten und zwingt den Angreifer gleichzeitig, mindestens sieben kleine Knoten zu kompromittieren, um einen großen Knoten aus dem Netzwerk auszuschließen.

Abschließend schlagen wir Forschungsgebiete vor, die sich durch die Entwicklung von HEFS ergeben haben und präsentieren Schwächen und Verbesserungsvorschläge unseres Schemas.

Abstract

The security of information technology is an important area of computer science. This thesis addresses specific security problems of a technology called wireless sensor networks.

Wireless sensor networks are a current research topic in the area of ubiquitous and pervasive computing. A network of thousands of sensor nodes can be deployed in a number of scenarios such as disaster sites, environmental monitoring or surveillance. Each node is equipped with a sensor unit, a low-power radio unit, a low-power CPU, memory, and a battery. After deployment, the motes build a wireless multihop ad-hoc network and forward reports with measurements to a base station called sink.

The main challenge in the area of sensor network security are resource limitations: Energy, computing and memory resources are limited, so that public-key cryptography is difficult to apply. On the other hand, an attacker can often gain physical access to the sensor nodes. An attacker who compromises one or multiple nodes can extract secret keys from their memories and use this data to inject false reports into the network.

The goal of this thesis is to detect and drop such false reports as early as possible. The impact of an attack should be limited to the region of the compromised node. As the attacker compromises more nodes, the network should degrade gracefully but never be completely compromised.

In this thesis, we assume a heterogeneous network structure: A network consists of many small node that are subject to the limitations described above and of a small number of large nodes that have plenty computing, memory and energy resource. We develop a "hybrid en-route filtering scheme" (HEFS) that uses public-key cryptography on the large nodes and symmetric algorithms on the small nodes in order to achieve the presented goals.

In order to evaluate HEFS, we have performed a proof-of-concept implementation in a Java-based simulator. We have simulated the injection of false messages as well as the creation of valid reports, and we have compared the energy consumption to a network without en-route filtering.

The evaluation shows that HEFS can significantly reduce the energy consumption caused by injected messages, and that a heterogeneous network structure can reduce the energy consumption of small nodes, even if no en-route filtering protocol is applied.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The security of information technology is an important area of computer science. In the beginning of computer technology and networking, protocols and systems were often designed without a focus on security. The results are still noticeable today. For example, the mail transfer procotol SMTP was originally designed without support for data integrity, authenticity and confidentiality. Nowadays there are optional extensions of the protocol that allow data encryption and digital signatures, but these extensions are rarely used in practice. This is partly because the setup of a security infrastructure is not trivial, but also because the extensions are optional and not part of the core protocol. It is difficult to add security measures later-on, when compatibility issues have to be considered as well.

The motivation of this thesis is to develop security mechanisms for a new technology before it is deployed on a large-scale. This technology is called "wireless sensor networks". In this chapter, we describe the basic architecture, hardware components, use cases and challenges of wireless sensor networks. Based on this introduction, we then given an overview over the chapters of this thesis.

1.1 Wireless Sensor Networks

A *wireless sensor network* (WSN) is the composition of thousands of small battery-powered sensor nodes called *motes*[1] and a base station called *sink*. Each mote is equipped with a small processor and memory, a radio unit and a sensor unit. The task of a sensor network is to perform measurements of the environment (e.g. temperature, brightness, humidity, seismic stimuli) and collect the results at the sink. Since the radio transmission range of a single sensor may not be powerful enough to reach the sink directly, the motes create an ad-hoc network and forward the results to the sink on a multi-hop route.

wireless sensor network, WSN

motes

sink

A popular mote-type is the MICA2 Mote [5] (see figure 1.1) of the SmartDust project at the University of California in Berkeley. It uses a 16 MHz Atmel ATMega128L processor with 512 kBytes of flash memory for measure-

[1] In this thesis, the words *mote* and *sensor node* are used analogously.

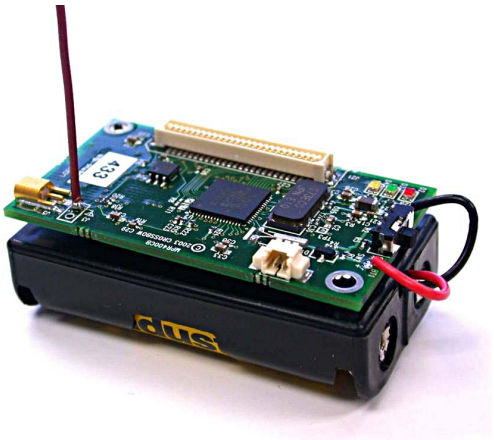


Figure 1.1: MICA2 mote [1]

Figure 1.2: SunSpot mote [2]

ments, 4 kBytes of configuration EEPROM and 128 kBytes of program memory. The power source consists of two AA batteries.

Another recent development is the Java SunSpot™ [4] (see figure 1.2). It uses a 180 MHz, 32-bit ARM920T processor with 512 kBytes RAM and 4 MBytes flash memory. Energy is provided by a 3.6 V rechargeable 750 mAh lithium-ion battery.

1.1.1 Operational scenarios

Originating from military research projects, sensor networks have found their way into the civil world. There is a large variety of scenarios in which sensor networks can be applied or are already in use. These applications include *wildlife monitoring*, *environmental monitoring* (temperature, water pollution, humidity), *medical monitoring* (medical implants measuring heartbeat) and *monitoring disaster sites*.

Another application is the use as surveillance system or the extension of an existing surveillance system: A system called Shot Spotter [7], that is discussed controversially by some parties [9], consists of microphones which may be mounted on wireless sensors nodes. These microphones are deployed in urban areas to triangulate the sound of gun shots. In [29], a sensor network is used as extension to an existing video surveillance system, for example on a university campus. It consists of microphone sensors that pick up loud audio events such as screams and direct the attention of the observing security people to the appropriate cameras.

Finally, an interesting application is the observation of structural integrity in buildings, bridges and other structures [34, 23]. The traditional approach uses expensive *data aquisition systems*, which are connected to seismic, vibration, moisture, temperature and displacement sensors using a lot of wiring. With such a system the setup of wires and sensors can take hours, which means

that an installation aimed for a few-hours measurement is highly uneconomical [34].

Wireless sensor networks in such a scenario are cheaper and easier to install than their wired alternatives. The installation only consists of putting the sensor nodes in the right places and activating the network. The routing paths are created by the network automatically. This makes such short-term measurements economical, but it also reduces the cost of long term installations that run over multiple months or years.

The use of wireless battery-powered motes is not restricted to outdoor scenarios: A fire detector system, that is installed in an old building, can be considerably cheaper if no additional wires have to be installed.

1.1.2 Challenges of Sensor Networks

The main challenge when developing sensor network applications and middleware is the scarcity of resources. The technical data shows that the CPU is slow (16 MHz on a MICA2), the available memory is small (4 kBytes RAM and 128 kb program memory) and the energy supply is limited (max. 3000 mWh when using two AA batteries). There are two reasons for this scarcity:

1. Since a sensor network consists of a large number of motes, each single mote must be cheap. Otherwise, a sensor network would not be economical.

2. Motes are commonly assembled from low-power components. Especially in long-term scenarios, reducing the energy consumption is more important than increasing the computing resources.

The resulting challenge is to design protocols and algorithms that minimize the consumed energy. Table 1.1 summarizes measurements performed by Arvinderpal S. Wander et al. [32]. They compared the energy consumption of message transmissions, symmetric cryptography and public-key cryptography on a MICA2dot mote. This mote is based on the same components as the MICA2 mote. The table shows that radio transmissions consume more energy than symmetric cryptography.

Symmetric Cryptography	AES-128 encryption	1.62 μJ/byte
	AES-128 decryption	2.49 μJ/byte
	SHA-1	5.90 μJ/byte
Radio usage	Energy to transmit	59.20 μJ/byte
	Energy to receive	28.60 μJ/byte
Public-Key-Cryptography	ECDSA-160 sign	22820.00 μJ
	ECDSA-160 verify	45093.00 μJ

Table 1.1: Energy consumption of a MICA2dot [32]

Since most non-cryptographic computations are less expensive than symmetric cryptography, this table shows that message transmission are generally more expensive than computations.

Efficient routing protocols are needed. They must be capable of minimizing transmissions between the nodes and scalable enough to handle a network with thousands of network members. Depending on the scenarios they must also support mobile sensor nodes.

One approach for the reduction of radio transmissions is the *in-network data aggregation*. The goal is to let intermediate nodes perform computations and forward only the aggregated result of multiple incoming messages, in order to reduce the number or transmitted messages.

Another challenge is the handling of node failure: Sensor nodes may stop working because their battery is empty or because they are destroyed by natural (or human) forces. Routing protocol and sensor network applications must be able to deal with such occurences.

The creation of security frameworks for sensor networks presents a number of additional challenges: An attacker is usually not bound to the same resource constraints as the network. Public-key cryptography is difficult to implement in sensor networks because of the resource constraints (see table 1.1). And finally, an attacker may have physical access to sensor nodes and thus be able to extract secret cryptographic keys from their memory.

All problems must be resolved without user interaction because the network acts autonomously in most scenarios.

1.2 Thesis structure

This thesis develops and evaluates a scheme to lower the impact of a specific attack in a heterogeneous sensor network. Chapter 2 describes the security problems and common attacks on sensor networks and points out which attacks are relevant to this thesis. Chapter 3 presents related work and the state-of-the-art in sensor network security. Chapter 4 introduces the notion of heterogenous sensor networks and describes the developed scheme, that we call *hybrid en-route filtering scheme* (HEFS). In order to evaluate the efficiency of HEFS, we have implemented the scheme in a simulated environment. The implementation details are presented in chapter 5. Chapter 6 presents the simulation results and the evaluation of HEFS. Finally chapter 7 draws a conclusion.

Appendix A contains an overview over the variables and function names used in this thesis and a clarification of common terms and mathematical notations.

Chapter 2

Sensor network security

Sensor networks may be deployed in critical scenarios (e.g. when a malfunctioning network results in danger to human life or great financial loss). Such networks must be protected against human intrusion. Consider a sensor network acting as fire warning system in an old building, where exhaustive wiring is too expensive. A potential attacker may want to disable the system or create false alarms, thereby provoking the evacuation of the building. In the first case, life is in danger. In the second case, he may cause a considerable financial damage or at least undermine the trust in the alarm system.

2.1 Classes of attackers

On a technical level, attackers can be distinguished by the amount of resources and by the amount of secret information that is available to the attacker. The first distinction divides adversaries into *laptop class* and *mote class* attackers [20]:

- A *laptop class attacker* has the resources of a larger computer and possibly a larger transmission range than a node. — laptop class attacker
- A *mote class attacker* has modified the program of a sensor node and attacks the network through this mote. This means that his computing resources are as limited as the resources of the other nodes. — mote class attacker

The second distinction separates the attacker into *insider* and *outsider* attackers [36, 35, 38, 37]:

- An *outsider attacker* tries to attack a network solely on the wireless channel. — outsider attacker
- An *insider attacker* gathers physical access to a sensor node and uses the information stored in this node to attack the network. This type of attacker can use secret cryptographical keys of this node to attack the network. — insider attacker

2.2 Types of attacks

In this section, we assume that the network is either unprotected or threatened by an insider attacker. We show common attacks that are discussed in the literature.

Sybil attack In the *sybil attack* [14], an attacker can incoorporate multiple node identities by sniffing the network traffic for valid ID numbers. This can be used to change the outcome of polls or aggregation functions or to perform other attacks.

Selective forwarding attack The *selective forwarding attack* [20] is a typical denial-of-service attack. A node controlled by the attacker[1] participates in the building of a routing tree, meaning that other nodes will try to relay messages over this node. The attacker examines each message and then decides whether to forward or to drop it. If necessary, an ACK reply is sent back, even if the message has been dropped.

Blackhole attack The *blackhole attack* [20] is a simple form of the selective forwarding attack, where messages are dropped without prior examination.

Data alteration attack Instead of dropping messages on a forwarding node, the attacker can also change the header or payload data of forwarded messages. This kind of attack is called a *data alteration attack* [12].

False data injection attack In an unprotected network, the attacker can simply sniff the traffic for a valid node ID and perform a sybil attack to "become" this node. Then he can create messages and inject them into the network in order to provoke a false alarm or in to falsify measurements. The attack is called a false data injection attack [36].

Path-based denial of service attack The path-based denial of service attack (pDoS-attack) [13, 36] is similar to the false data injection attack, but the goal is a different one: By injecting messages on a large scale, the energy of forwarding nodes is consumed. This attack is successful, as long as the intermediate nodes forward injected messages. A detection mechanism at the sink is not a sufficient counter-measure. The pDoS-attack has a multiplied effect compared to other DoS-attacks because the attacker can drain all nodes on the route to the sink by sending messages from only one spot.

Wormhole attack The *wormhole attack* [20] is usually used as preparation for other attacks. In order to preserve energy, most routing protocols try to forward messages along the shortest route to the sink. A laptop class attacker

[1] The node can be compromised but may also be a node introduced to the network by the attacker.

can boost the signal from a distant location to a location near the sink and thereby create a path with fewer hops. The nodes near the starting-point of this wormhole then recognize the attacking node as next hop on the shortest route to the sink and use it to relay messages. The attacker then controls the connections of a whole region and can perform different attacks such as blackhole, selective forwarding or data alteration.

Replay attack As in wired networks, an attacker can simply replay messages that he sniffed from the network for reasons of battery exhaustion or in order to exploit security weaknesses. This attack is called a *replay attack* [12].

Routing loops A more subtle type of attack, that is aimed on battery exhaustion, is the creation of *routing loops* [20]. The attacker influences the creation of the routing tree to provoke the formation of loops. If a message gets caught in such a loop, it is forwarded indefinitely and drains the energy of the forwarding node.

Eavesdropping *Eavesdropping* is basic attack on all kinds of computer networks. The additional problem of wireless sensor network, is that an insider attacker can listen in on the traffic that is routed through a compromised node, even if the communication is encrypted. It is hard to detect such an attack because the attacker does not perform any action that could be detected by other sensor nodes.

2.3 Relevant attacks

The detection and prevention of each of these attacks discussed cited publications. It is difficult to address all attacks at once. Most publications focus on a single attack or on a set of attacks.

This thesis is primarily concerned with the *false data injection attack*, the *data alteration attack* and the *path-based denial of service attack*. In order to secure a sensor network against other attacks, other protocols have to be implemented and integrated as well.

Chapter 3

Related Work

en-route filtering

This thesis develops a mechanism for *en-route filtering* which is a subtopic of sensor network security. In order to implement this mechanism, basic authentication and encryption schemes must be already present. Section 3.1 presents two such schemes. Section 3.2 discusses the advantages and disadvantages of existing en-route filtering schemes. Section 3.3 describes two compression schemes for Message Authentication Codes (MAC). These methods are used in the developed *hybrid en-route filtering scheme* (HEFS) order to decrease the communication overhead. In HEFS, we assume a sensor network that contains a small number of sensor nodes with larger resources. We assume that these nodes are able to perform public-key cryptography. Section 3.4 examines the effect of public-key cryptography on the low-resource nodes of such a network.

3.1 Encryption and authentication

Due to the lack of resources, sensor nodes generally use only symmetric cryptography to encrypt and authenticate communication channels. Messages are encrypted with symmetric algorithms and shared secret keys. Authentication is provided by MACs that are attached to each message.

Symmetric cryptography has a major disadvantage compared to public-key cryptography: Encrypting and decrypting messages requires the knowledge of the same key. The same holds for the creation and verification of a MACs. Consequently, an attacker who has extract a verification key, can also use this key to endorse reports.

This limitation raises the question of how keys should be distributed in a secured network and whether authenticated broadcasts are possible. The following sections present protocols that address this question.

3.1.1 Key distribution

The question of how the keys should be distributed is vital to the security of sensor networks. There are two trivial approaches to this problem:

1. **One key is shared among all nodes in network.** This approach is

insufficient against insider attackers, since the key is immediately known to the attacker once he has compromised one node.

2. **Each node-pair in the network shares pre-distributed pair-wise key.** This method is infeasible on low-resource sensor nodes or at least, does not scale well. On a network of 10000 nodes, every node would have to store 10000 keys of 8 bytes length[1]. Storing 80000 bytes is infeasible on a MICA mote with four kilobyte of RAM.

The second approach assumes that every node has to communicate with every other node in the network. Normally, secure communication only needs to be possible among neighboring nodes and between each node and the sink. The problem of pre-distributed keys is, that prior to the deployment of the network, it may be impossible to determine which nodes will be neighboring nodes post-deployment.

One approach to perform the key distribution post-deployment is described in the *Localized Encryption and Authentication Protocol* (LEAP) [39]. This protocol makes the assumption that it is impossible for an attacker to compromise a node during a pre-defined time interval after deployment. This assumption is reasonable, since the area of deployment can be secured from adversaries during that time. LEAP differentiates between four types of keys: LEAP

1. *Individual keys* are shared between a node and the sink — individual keys
2. *Cluster keys* are among a node and all surounding nodes that are in communication range. These keys are used when the node wants to send an encrypted broadcast message to all neighboring nodes.[2] — cluster keys
3. *Pairwise keys* are shared between two nodes that are in communication range. These keys are used when a node sends a message to one explicit destination node. — pairwise keys
4. The *group key* is shared among all nodes and the base station. — group key

For the initialization of the different key types, LEAP uses a family of pseudo-random functions $f_K(x)$ which have the properties of a MAC-function ($\text{MAC}_K(x)$). $f_K(x)$

Individual keys and the group key The individual key for a node $\mathfrak{a}$ is generated from a master secret K_m by computing $K_{\mathfrak{a}}^{indiv.} = f_{K_m}(ID_{\mathfrak{a}})$. That way, the base station does not need to store all keys, but can compute them from the master secret when needed. The individual keys are loaded into the nodes prior to deployment. The same is done with the group key.

[1] The assumption of 64-bit keys is common in all referenced publications.

[2] Note that, in later sections, the word *cluster* has the meaning of multiple nodes working together to create a report. In LEAP, *every* node is the center of a cluster that consists of all nodes reachable within one hop.

Pairwise keys Prior to deployment, a shared initialization secret K_I is loaded into each sensor node. Each node $\mathbf{a}$ generates a master key $K_{\mathbf{a}} = f_{K_I}(ID_{\mathbf{a}})$. Then it discovers the ID of its neighbors and computes their master secret as well. Discovery is done as follows, considering $\mathbf{b}$ is a neighbor of $\mathbf{a}$:

$$\begin{aligned} \mathbf{a} &\longrightarrow * : \quad ID_{\mathbf{a}}, nonce \\ \mathbf{b} &\longrightarrow \mathbf{a} : \quad ID_{\mathbf{b}}, \mathrm{MAC}_{K_{\mathbf{b}}}(ID_{\mathbf{b}}, nonce) \end{aligned} \tag{3.1}$$

The pairwise key is then computed as $K^{pairw.}_{\mathbf{a},\mathbf{b}} = f_{K_{\mathbf{b}}}(ID_{\mathbf{a}})$. Intermediate results such as $K_{\mathbf{b}}$, $K_{\mathbf{a}}$ and K_I are erased after key setup.

Cluster keys After setting up the pairwise keys, each node generates a random cluster key. It then transmits this key to each one-hop neighbors using the appropriate pairwise key for the encryption of each message.

Key update LEAP proposes the following action for a key update after a compromised node has been detected.

1. Each node creates a new cluster key and sends it to all neighbors except the compromised nodes.

2. The sink generates a new group key and broadcasts it through the network. The broadcast is encrypted with the cluster key of each transmitting node, which means a re-encryption of the message after each hop.

3.1.2 Authenticated broadcasts

A way to circumvent the need for public-key signatures in secure broadcasts has been published by A. Perrig et. al. in SPINS [26]. The protocol is called μTESLA *$\mu TESLA$* and is based on hash-chains. Hash-chains have been first introduced by Leslie Lamport 1981 in [21]:

hash-chain **Definition: 3.1** *A one-way hash-chain is a series of $n+1$ values*

$$S(x) \quad for\ x \in \{0, \ldots, n\} \tag{3.2}$$

that are defined through a cryptographical hash-function $H(x)$ and an initialization value I. The last value of the chain is

$$S(n) \stackrel{def.}{=} I \tag{3.3}$$

The remaining values $S(i)$, $i < n$ of the chain are defined as

$$S(i) \stackrel{def.}{=} H(S(i+1)) = H^{n-i}(I) \tag{3.4}$$

Such a series is called a one-way hash-chain because it is only possible to derive $S(i)$ from $S(j)$ if $i < j$. Even if $S(i)$ is known, it is infeasible to compute $S(i+1)$ as long as the hash function is strong enough. But it is possible to verify whether $S(i+1)$ is correct if $S(i)$ is known. This can be done by verifying

$$S(i) \stackrel{?}{=} H(S(i+1)) \tag{3.5}$$

As a result, only someone, who knows the initialization secret I, is able to provide the whole chain in the order $S(1), S(2), \ldots S(n)$, but anybody knowing the first value $S(0)$ is able to verify the chain.

Lamport's original application of hash-chains was the generation of one-time-passwords: A hash-chain of n values is generated. The verifying party A stores the value $S(0)$ and the authenticating party B stores the end of the chain (or the whole chain). During the authentication process, B sends $b = S(1)$ to A. A verifies that $H(b) \stackrel{?}{=} S(0)$ and allows A to log in. Then B replaces the stored value $S(0)$ by b. For the next log in, A sends the value $S(2)$ to B.

Such a scheme allows B to verify the authenticity of A without knowing the authentication secret.

One-way hash-chains can be used on low resource sensor nodes because no public-key operations are required.

μTESLA μTESLA uses hash-chains to ensure the authenticity of a broadcast message. The sink stores a hash-chain $S(i)$. The first the chain $S(0)$ is stored on each node of the network before deployment. The sink and the nodes maintain a global counter. It represents the index of the first value in the chain that is not known to the sensor nodes. This counter is initialized to $c = 1$ and increased after a specified time interval.

In μTESLA, the values of a hash-chain are used as MAC-keys. The sink attaches a MAC to every broadcast message M, to ensure integrity and authenticity:

$$\text{sink} \longrightarrow **\,^{3}: \; M, c, \text{MAC}_{S(c)}(M) \tag{3.6}$$

The nodes cannot verify the authenticity of the message immediately because the value $S(c)$ is unknown to them. They can however verify that the value c is valid.

The value $S(c)$ (*commitment*) is broadcasted through the network after the counter has been increased. There may be a specified delay after the end of the validity period of $S(c)$ in order to compensate inaccurate system clocks in the sensors. When a sensor receives $S(c)$, it first validates it by checking $H(S(c)) = S(c-1)$ and then verifies the MAC of the message. If the MAC is correct, it re-broadcasts the commitment and accepts the stored message as authentic. Figure 3.1 illustrates the μTESLA protocol.

commitment

Discussion μTESLA can be used to broadcast authenticated messages without the application of public-key cryptography. It is not possible for an attacker to send forged broadcast messages to the nodes.

[3] multi-hop broadcast, see appendix A

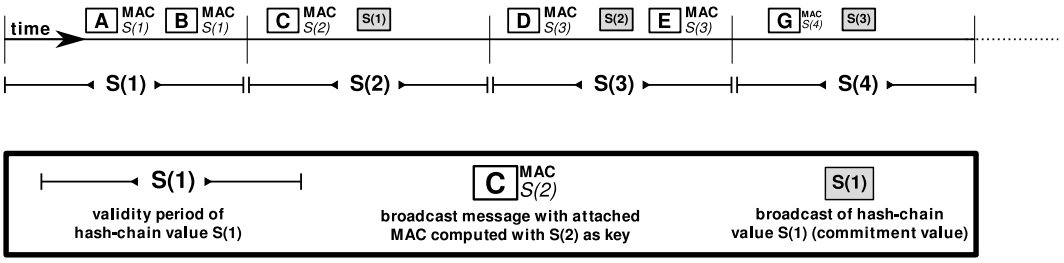


Figure 3.1: Illustration of the μTESLA protocol: The authenticity of the messages A and B can be verified after the broadcast of $S(1)$.

A denial of service attack is possible due to the fact that receiving nodes have to store broadcasts until they receive the commitment. An attacker can send multiple broadcast messages into the network, until the nodes' restricted buffer memory is exhausted. The nodes can then accept no more broadcast messages. An authentic message sent by the sink can be blocked by such an attack.

3.2 False data injection and pDoS-attack

en-route filtering

After the presentation of basic encryption and authentication methods, this section discusses some of the recently published *en-route filtering* schemes. These schemes contain concepts that are also used in the *hybrid en-route filtering scheme* (HEFS).

En-route filtering schemes try to detect and drop an injected messages as early as possible. The goal is to prevent path-based denial of service attacks (pDoS-attacks), false data injection attacks and data alteration attacks. In order to clarify the notation, we use the following terms throughout this thesis:

event
- An *event* is a measurement that has to be reported by the sensor network.

location (event)
- The *location of an event* is the actual location where a stimulus occurs.

report
- A *report* is the message that is sent to the sink in reaction to an event.

location (report)
- The *location of a report* is the location at which the stimulus happened according to the report. In case an attacker forges or modifies a report this location can differ from the location of the event.

point of injection
- The *point of injection* of a message is the location where the message is injected into the network.

loation (key)
- The *location of a key* is the location for which this key is valid. A valid report must have the same location as the keys used to create the attached MACs.

3.2.1 Multiple report authentication and en-route verification

Fan Ye et al. present a scheme for *Statistical en-route filtering of injected false data* (SEF) [36] to address the problem of false data injection as well as the path-based DoS attack. They introduce two basic ideas: SEF

1. **Multiple report authentication:** The report of an event is authenticated by multiple nodes. One node (the *cluster head*) assembles the actual report and neighboring nodes attach MAC signatures after performing a plausibility check. The feasibility of this procedure naturally depends on the application's sensor model. cluster head
2. **En-route verification:** The keys used for the MAC generation are shared by random nodes in the network. With a certain probability, a forwarding node can verify one of the MACs attached to the report and drop the report if it is invalid.

Two parameters can be adjusted in the system:

1. T is the number of MACs that must be attached to a valid report (i.e. the number of nodes agreeing to a report).
2. n is the total number of MAC keys distributed to the nodes[4].

Sequence of operations Before deploying the nodes, n keys are generated and distributed such that each node stores exactly one randomly selected key K_i. Each key is stored on multiple sensor nodes.

Figure 3.2 illustrates this scheme. The highlighted nodes store the same key. When an event is perceived, the following actions are executed:

1. The sensor nodes, which detect the event, choose a cluster head among themselves.
2. The cluster head generates a report R containing the location, type and time of the event. This report is sent to the neighboring nodes.
3. Each neighboring node uses its sensors to verify the plausibilty of the received report. If the report is plausible, it returns $\mathrm{MAC}_{K_i}(R)$ to the cluster head.
4. The cluster head chooses T MACs from the helper nodes and sends
$$\Big(R, i_1, i_2, \ldots, i_T, \mathrm{MAC}_{K_{i_1}}(R), \mathrm{MAC}_{K_{i_2}}(R), \ldots, \mathrm{MAC}_{K_{i_T}}(R)\Big) \tag{3.7}$$
to the sink, where i_2 is the ID of the key K_{i_2}.

Every node en-route to the sink, that node stores the appropriate keys, verifies the correctness of $\mathrm{MAC}_{K_i}(R)$. If the MAC is invalid, the report is dropped. In all other cases, the report is forwarded.

[4]The actual key distribution scheme describe in SEF is more complex: Keys are divided into k categories with $\frac{n}{k}$ keys each. For simplicity, we assume that $k = n$, thus every category contains exactly one key.

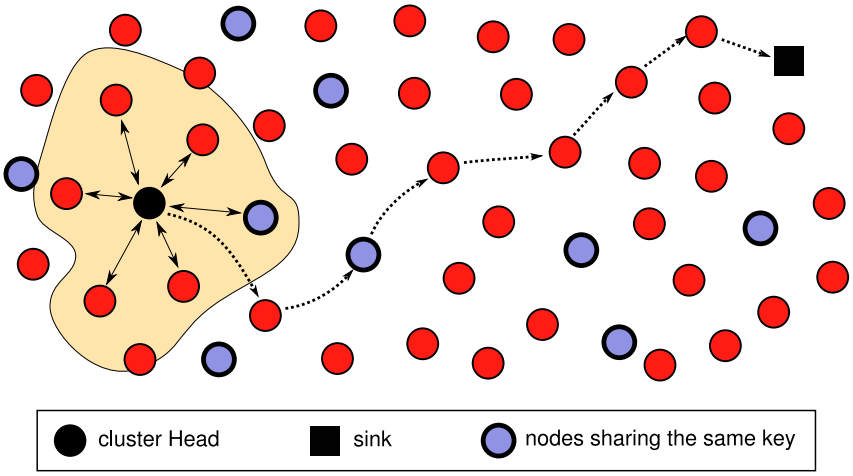


Figure 3.2: Illustration of the statistical en-route filtering scheme

Discussion If the attacker knows $x < T$ of the n keys, he must guess the remaining $T - x$ MACs in order to create a valid report. The probability that a node stores the key belonging to a guessed MAC is $\frac{T-x}{n}$. Assuming that it is impossible to guess a correct MAC, the probability that a node cannot verify the authenticity of the report is

$$p(x) = 1 - \frac{T - x}{n} \tag{3.8}$$

The probability that a forged report travels more than h hops, is then:

$$p_h(x) = \left(1 - \frac{T - x}{n}\right)^h \tag{3.9}$$

For example, if $n = 10$, $T = 5$ and the attacker knows one key, there is a 90 percent probability that an injected report is filtered after four hops. With three compromised keys, the 90 percent probability is at ten hops (see figure 3.3).

The drawback of the scheme is, that once the attacker is in possession of T different keys, he can create arbitrary false reports with arbitrary locations and inject them at any point in the network. He can find these T different keys in the neighborhood of any sensor node because otherwise the scheme would not work. An attacker can find out which keys are stored on which node, by listening in on the radio, locate T nodes with different keys and compromise them. Afterwards, he possesses enough information to generate forged reports with arbitrary locations. In this case, the whole scheme becomes useless.

3.2.2 Location-based keys

location-based keys

In order to fix this problem, Fan Ye et al. published a follow-up scheme [35] that introduces the concept of *location-based keys*. The basic idea is, that the MAC keys are not distributed randomly but are bound to a certain location.

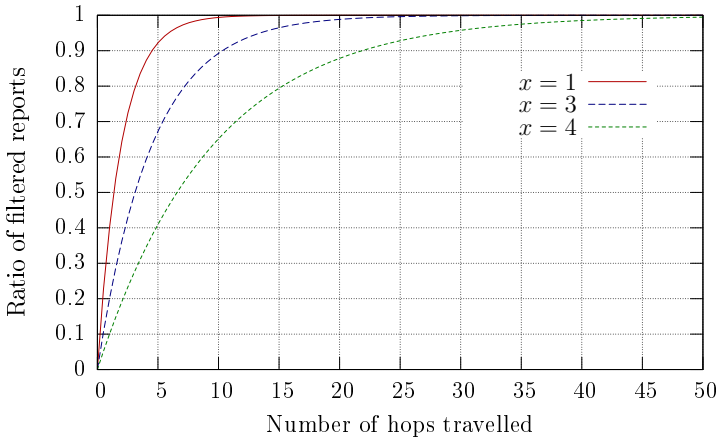


Figure 3.3: Number of dropped reports after h hops for 1, 3 and 4 compromised keys

A report is only valid if the keys used to generate the T MACs are valid for the location of the report. The localization of MAC keys requires a new form of key distribution and a more sophisticated approach to MAC verification. The scheme with location-based keys is based on the following assumptions:

1. The attacker is unable to compromise any nodes within the first few minutes after deployment (similar to LEAP in section 3.1.1)

2. Nodes possess a localization component that allows them to discover their own location. The deployment area is divided in cells as illustrated in figure 3.4.

3. A geographic routing protocol is used in the network. It is assumed that such a protocol provides little deviation of the direct path from event to sink. Respecting this deviation, a node can only receive messages from the *upstream-region*, which is illustrated as gray beam in figure 3.4. This region can be computed from the location of the sink and the location of the node.

upstream-region

Key setup Prior to deployment, a number of initialization secrets K_s^I is generated and each node is pre-loaded with one of them. In the initialization phase, all nodes aquire their location and derive the cell (x, y) they are in. This location is used to compute the *endorsement key*

endorsement key

$$K_{x,y,s} = \text{MAC}_{K_s^I}(x, y) \tag{3.10}$$

for each node. Each node also determines its upstream-region and computes and stores $K_{x,y,s}$ for a random selection of the intersecting cells. These keys are used to verify reports later.

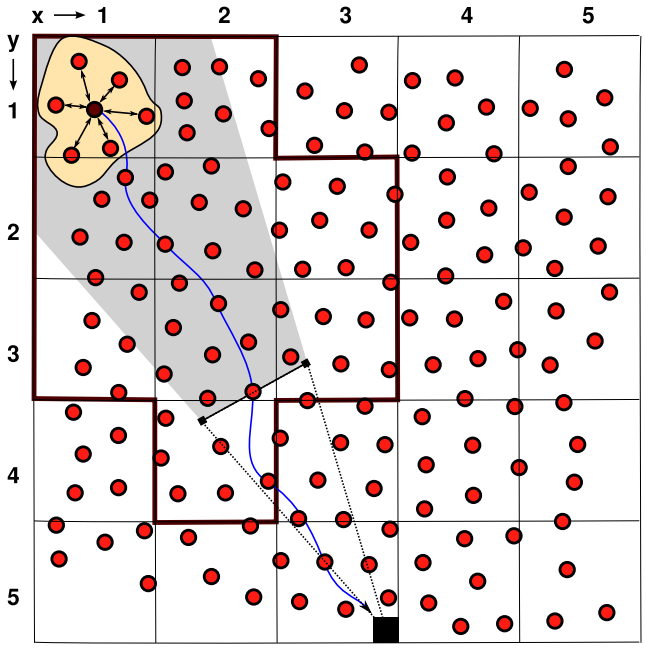


Figure 3.4: System setup with localized keys [35]

Report generation The report generation and en-route verification works similar to section 3.2.1. For the generation of reports, only the localized endorsement key is used. The generated report is

$$\left(R, s_1, s_2, \ldots, s_T, \mathrm{MAC}_{K_{x,y,s_1}}(R), \mathrm{MAC}_{K_{x,y,s_2}}(R), \ldots, \mathrm{MAC}_{K_{x,y,s_T}}(R)\right) \quad (3.11)$$

where R as in section 3.2.1 contains the location information (i.e. the cell) of the event.

Verification Each verifying node performs two checks:

1. It verifies that the location of the report (x, y) is in the upstream region of the node and drops the report if it is not.
2. If the node stores any of the keys $K_{x,y,s_1}, \ldots, K_{x,y,s_T}$ it verifies the corresponding MAC of the report and drops the report if the MAC is invalid.

Discussion The goal of this scheme is, to prevent creating a fake report, without actually compromising T nodes at the location of the report. This goal is achieved partly. For example, if an attacker compromises T nodes at location $(1, 1)$ (see figure 3.4), he can only create false reports from the same location. These reports will only reach the sink if they are injected into any node that has this location as part of its upstream region. Reports injected at other locations (e.g. cell $(1, 5)$) are dropped because then, the upstream-region-check of the forwarding nodes fails. If the attacker tries to use the compromised keys to create a report from the location $(1, 5)$, and a forwarding node stores $K_{1,5,s_i}$, then this node will detect the forgery and drop the message.

The problem with this scheme is that the endorsement keys of a cell (e.g. cell $(1,1)$) are also stored as verification keys on the en-route nodes. An attacker can gather these keys by comprising enough en-route nodes without actually being in cell $(1,1)$.

If the network protects a specially secured area to which an attacker has no physical access, then the attacker can raise false alarms there after compromising enough nodes that have this area in their upstream region.

3.2.3 Using hash-chains

In section 3.1.2 we presented μTESLA as a method for authenticated broadcasts using hash-chains. μTESLA is not suitable for en-route filtering for the following reason:

- En-route filtering means that each node verifies the signature *before* forwarding the message.
- In μTESLA, a node can only verify the message $(M, c, S(c))$ after receiving $S(c)$.
- Consequently, a node has to forward the message after the broadcast of $S(c)$. This is after the end of the validity period of $S(c)$).

When the first verifying node could forward the message, the MAC-keys would not be valid anymore.

Defending against pDoS A different approach to use one-way hash-chains against the path-based DoS attack is presented in [13]. The setup is as follows:

- Each node $\mathbf{a}$ stores a hash-chain $S_{\mathbf{a}}$ and a counter i which is initialized with 0.
- Each intermediate node on the routing path of $\mathbf{a}$ stores $S_{\mathbf{a}}(0)$ as verificator.

Whenever $\mathbf{a}$ creates a message to the sink, it attaches the next element $S_{\mathbf{a}}(i+1)$ of the hash-chain to the report and increases i by one. All forwarding nodes verify the authenticity of $S_{\mathbf{a}}(i+1)$ and store this value as next verificator. This scheme is not designed as a defence against insider attackers. When one node is compromised, it is possible for the attacker to perform a pDoS-attack using the hash-chain of this node.

Multiple chains The *fault localized scheme for false report filtering in sensor networks* [38] is an extension of this scheme presented in [13]: Multiple nodes endorse a report with the next elements of their hash-chains. In addition, each node creates a MAC using an individual key (see 3.1.1). These MACs are attached to the message, in order to enable the sink to verify the report. Hash-chains and individual MAC keys are bound to the location of

the endorsing sensor node in the sense that the sink knows what the location of the sensor node is.

In the verification process, each forwarding node verifies the number of hash-values. If it possesses the previous value of a hash-chain, it also verifies the corresponding hash-value. If it possesses no previous value of any hash-chain in the message, it cannot verify the authenticity of any hash-value. The *fault-localized scheme* does not explicitly address this case, but a similar situation occurs when new nodes are deployed while the network is already running. These nodes also do not store any previous hash-values and simply forward messages without verification.

Discussion Hash-chains are an effective way of limiting the number of messages injected into the network. An attacker cannot inject new messages into the network unless he gathers enough hash-chains by compromising nodes.

One drawback of hash-chains is the trade-off between space and computing time on the generating node. There two extreme approaches of storing a hash-chain on the generating node:

1. Only the last value is stored. Every value is computed on-demand. This method causes a significant overhead of computing time and energy.

2. Every value of the chain is stored. This method uses a lot of memory and is therefore infeasible on small sensor nodes.

There are trade-offs such as storing only every k-th value of the chain. Nevertheless, the generating nodes have to use a significant amount of resources, while the verifying nodes only need to store one hash-value and only have to perform one hash-computation.

Apart from the resource concerns, the *fault localized scheme for false report filtering in sensor networks* eliminates the localized-key problem from section 3.2.2: Intermediate nodes store neither the data necessary to generate a valid hash-value nor the keys to create the MAC that is verified by the sink. Nonetheless, an alteration of the payload data would not be detected by the forwarding nodes because the hash-values have no mathematical relation to the payload of the message. The alteration would be detected at the sink, but still consume energy on the intermediate nodes.

Another problem appears, when the attacker attaches node IDs of a completely different region to the messages. The scheme in section 3.2.2 can drop a report if the location is not found the upstream region of the node. In the *fault localized scheme*, however, every node forwards the message because no intermediate node knows the appropriate hash-values to verify the message. An attacker can use this opportunity to perform a path-based denial of service attack.

3.3 Multiple MAC-Compression

Some of the presented schemes for en-route filtering perform a report endorsement by multiple nodes using Message Authentication Codes. In these schemes, a significant communication overhead is caused by the MACs and the node IDs that are attached to every message. The MACs are the largest part of this overhead: A MAC consists of at least 64 bit while a node ID, even in a large network, is not larger than 16 bit. SEF and the *fault localized scheme* present two different mechanisms to reduce the size of the MAC values.

3.3.1 Bloom-filters

The *bloom-filter* is a probabilistic data-structure introduced 1970 by Burton H. Bloom [10] as a method of efficiently testing the membership of an element in a set. The space-/time-efficiency comes at the price of occasional false-positives. Bloom-filters are commonly used in non-security applications such as spell-checking. SEF adapts this data-structure to test the membership of a MAC value in a set of MAC values, thereby compressing multiple MACs to one value. bloom-filter

First, we will present the general definition of a bloom-filter storing arbitrary elements:

Definition: 3.2 *Let* $C[i], i \in \{0, \ldots, n-1\}$ *be a bit-array of length* n *and*

$$f_i : \{0,1\}^* \mapsto \{0, \ldots, n-1\} \qquad \forall i \in \{1, \ldots, h\} \tag{3.12}$$

be a family of h *hash functions.*

A bloom-filter represents a set S *of elements using the bit-array* $C = \text{bloom}(S)$*. An empty set is represented by a bit-array that consists of zeros:*

$$S = \{\} \overset{\text{def.}}{\Leftrightarrow} \text{bloom}(S) = [0, 0, \ldots, 0] \tag{3.13}$$

With $C = \text{bloom}(S)$*, the formalism for adding elements and testing the membership of an element in the set is:*

$$e \in S \overset{\text{def.}}{\Leftrightarrow} C[f_i(e)] = 1.\ \forall i \in \{1, \ldots, h\} \tag{3.14}$$

Adding an element e to the bloom-filter is equivalent to setting the bits at position $f_i(e)$ to one while leaving the other bits untouched. Membership-testing is done by verifying that all bits at $f_i(e)$ are set to one. Removing elements from a bloom-filter is not possible.

Bloom-filters for MAC compression In order to use bloom-filters to compress multiple MACs, three conditions must be fulfilled:

1. The hash functions f_i must be cryptographically strong.

2. The number of MACs m stored in a bloom-filter, the number of hash-functions h and the bit-length n of the bloom-filter must be preset system parameters. Each verifier must not only perform the membership test for a MAC but also verify that the number of ones in the bloom-filter does not exceed $h \cdot m$.

3. $h \cdot m$ must be significantly smaller than n.

The first condition ensures that the attacker cannot use collisions in the hash functions. Of course, assuming a 64-bit bloom-filter with 10 functions f_i, a collision of one f_i can be found after $65 = 2^6 + 1$ computations of f_i, but $(2^6)^{10} + 1 = 2^{60} + 1$ computations are necessary in order to find a collision of *all* hash functions if the hash functions are cryptographically strong.

The second condition prevents the attacker from using $C = [1, 1, \ldots, 1]$ as valid value for a bloom-filter. The third condition reduces the probability that the attacker can guess the correct bloom-filter value by setting $h \cdot m$ random bits to one.

3.3.2 XOR-cumulation

Another way to compress multiple MACs is to cumulate the MACs using the bitwise XOR-function (denoted $\oplus$). The compressed version of MAC_{k_1}, MAC_{k_2} and MAC_{k_3} is then

$$\text{MAC}_{\oplus} = \text{MAC}_{k_1} \oplus \text{MAC}_{k_2} \oplus \text{MAC}_{k_3} \tag{3.15}$$

The verifying party is assumed to be in possession of all keys used to create the MAC. It can then verify the cumulated value by computing all MACs and applying the $\oplus$-function.

This compression scheme is used in the *fault-localized scheme* [38].

3.3.3 Discussion

Both presented methods can and should be used in order to reduce the size of the tail created by multiple MAC endorsements.

The XOR-cumulated MAC is smaller than the bloom-filter and provides a higher level of security. The verifying party has to know all MAC keys and can only determine if *any* of the MACs are wrong.

The bloom-filter is larger and vulnerable to false-positives, but it also is more flexible: A verifier can, given a list of key IDs verify for every single key, whether it was used in the creation of the bloom-filter or not.

3.4 Public-key cryptography

The main problem of the scheme presented in section 3.2.2 is, that verifying nodes must hold the generation keys. This problem could easily be solved by *public-key cryptography* (PKC). There are several publications that evaluate the feasibility of running PKC algorithms on low-resource sensor nodes, by measuring run-time and energy consumption of the computation. In HEFS, we do not use PKC on low-resource nodes, but we do use such nodes to forward messages signed with public-key signatures (see section 4).

public-key cryptography, PKC

This section therefore primarily focuses on the communication overhead caused by the application of public-key schemes. First, we present mechanisms to distribute the public keys throughout the network. Then, we discuss the length of the keys and signatures in different PKC algorithms.

3.4.1 Distributing public keys

PKC itself does not solve the problem of distributing the public-keys to the nodes. There are three possible alternatives which are discussed in this section:

1. All public keys are preloaded into all nodes.
2. A certificate system is set up.
3. An ID-based public-key scheme is used.

Pre-distribution of public keys This method consumes a large amount of memory on the sensor nodes. In section 3.1.1 we have discussed the infeasibility of preloading pair-wise keys for all communication pairs into the sensor nodes. Pre-distributing all public keys to all nodes would require each node to store the same number of keys. Since PKC requires larger keys than symmetric cryptography, the memory consumption would be even larger.

Certificate Authority Certificates are used in traditional computer security to create a public-key infrastructure. A *certificate authority* (CA) creates signatures for the public keys of all participants in the network. When an entity sends its public key to another entity, it can attach the certificate of the CA in order to prove that the provided public key is authentic. An example of a certificate-based system for sensor networks is TinyPK [33]. The goal of TinyPK is not en-route filtering, but to enable third parties to authenticate themselves to the sensor network. The protocol uses public key operations only to set up symmetric keys for further communication. The transmission of the certificate and the public key has to be split up into multiple packages because of the length of this data.

certificate authority, CA

A certifcate authority could reduce the memory overhead caused by the public keys. Nodes could send their public key towards the sink, so that each en-route node could store exactly the public keys of nodes in its upstream region. The disadvantage is the large communication overhead caused by transmitting the public keys and the certificates.

ID-based signature schemes "Identity-based public-key signature schemes" is a concept that was introduced by Adi Shamir in 1984 [28]. He presented an infrastructure, in which the unique identifier of a participant in the network is also used as his public key. The private key is issued to the participant by a *key generation center* and is computed from a secret global seed value k and the identity of the participant. In Shamir's work the key generation center uses the trapdoor function of the RSA scheme to generate the private key from the public key (i.e. the identifier string) of the participant. The recipient of a message can then use the (global) public key of the key generation center and the identifier string of the sender to verify the authenticity of a message.

key generation center

This approach is very attractive for sensor networks because each sensor node already has a unique identifier that can be used as public key. This identifier does not cause a significant transmission overhead because it is usually not larger than 2 bytes. The memory consumption is even smaller than in the CA-approach: Each node only needs to store the public key of the key-generation center and its own secret key.

In [37], Y. Zhang et al. use an identity-based signature scheme based on elliptic curves for several security applications including the generation of pairwise keys, report-endorsement with multiple nodes and en-route verification. Instead of using a key generation center to generate the keys, the seed value is pre-loaded into every sensor node prior to deployment. Based on the assumption, that no node can be compromised during the first minutes after deployment, each node generates a localized secret key and removes the seed value from its memory afterwards.

We do not define a concrete PKC scheme for HEFS, but ID-based PKC is the most promising approach of the three presented methods.

3.4.2 Key- and signature-length

The length of keys and signatures of PKC is generally larger then symmetric algorithms with the same security level. Table 3.1 is an extract of the "Yearly Report on Algorithms and Keysizes" by the European Network of Excellence in Cryptology [16]. For the size of a symmetric key (first column) it shows the key sizes of equivalent security for RSA, algorithms based on the discrete logarithm problem and elliptic curve cryptography. The size of the signature depends on the size of public and private keys:

- For the signature algorithm of *Rivest, Shamir and Adleman* (RSA) [27] signature algorithm, the signature is the same size as the RSA key (second column of table 3.1).
- The *Digital Signature Algorithm* (DSA) [6], is based on the Discrete Logarithm Problem (DLOG). The size of the signature is twice the size of the subfield (fourth column of table 3.1).
- The *Elliptic Curve Digital Signature Algorithm* (ECDSA) [6], is based on Elliptic Curve Cryptography, the size of the signature is twice the size of the key (last column of table 3.1).

Security (bits)	RSA	DLOG		ECC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

Table 3.1: Key-length equivalence of PKC concepts [16]

ECRYPT proposes a key length equivalent to 80 bits for long-term protection against small organizations. For RSA signatures, this means that the signature is 1248 bits long. DSA and ECDSA only need a length of 320 bits.

The size of the signature can be reduced even further: SFLASH [8] is a signature scheme based on a system called *Hidden Fields Equations* (HFE). It claims to provide the 80-bit security with a signature length of 259 bits. Another scheme called QUARTZ [24] reduces the signature length to 128 bits, with the drawback that the public-key has a size of 71 kBytes.

Another way of reducing the length of the signed message is to apply a signature scheme with message recovery. In such a scheme, the message does not have to be sent along with the signature because it can be reconstructed during the verification process. An ID-based signature scheme with message recovery is presented in [31]. The signature in this scheme is 320 bits long, but a message of 100 bits length can be reconstructed during the verification process.

The implementation in chapter 5 is based on the assumption that such a scheme is used to authenticate messages. Since we only use 64-bit MAC keys, we reduce the size the signature to 256 bits, which provides equivalent security.

3.5 Summary

In homogeneous sensor networks, the capabilities of efficient en-route filtering, especially using localized keys, are limited: Hash-chains cannot guarantee the integrity of a message, MAC keys have to be distributed on the en-route nodes, and public-key cryptography is too energy consuming and to complex to be applied in long-term scenarios on low-resource nodes.

In the next chapter, we will therefore use heterogeneous sensor networks and present a hybrid scheme which uses both, public-key cryptography and symmetric cryptography.

Chapter 4

Concept

The methods presented in the last chapter are based on a homogeneous network structure: All nodes have the same computing, memory and energy resources and all nodes have equally strong radio units.

One of the reasons for the resource limitations of sensor nodes is the fact that nodes have to be cheap: The cost of each node is multiplied by the number of nodes in the network. This thesis shows that there are advantages in deploying a small number of nodes that are more expensive and more powerful than the nodes commonly deployed in the network. If the network is deployed in a building, these nodes can even be attached to the power circuit without creating too much cost.

Deploying such nodes is a trade-off between efficiency and cost. This chapter develops a *hybrid en-route filtering scheme* (HEFS) that tries to take advantage of a heterogeneous network structure.

HEFS

4.1 Assumptions

In this chapter we assume an architecture which employs two types of nodes:

small node

- Many *small nodes* with limited resources as assumed in the previous chapters, such as the MICA2 mote.

large node

- A small number of *large nodes* with a more powerful CPU and more memory such as the SunSPOT. These nodes are also assumed to have a larger radio transmission range and infinite energy resources. We assume that these nodes are capable of performing public-key operations in an acceptable timespan.

The different radio transmission range of a large node may result in uni-directional links, where the large node is able to send a message but the small node is not able to answer in one hop. We assume that the link-layer protocol and the routing protocols are able to handle such uni-directional links. HEFS is additionally based on the following assumptions:

- *Individual*, *pairwise* and *cluster* keys are set up using a technique such as LEAP (see section 3.1.1).

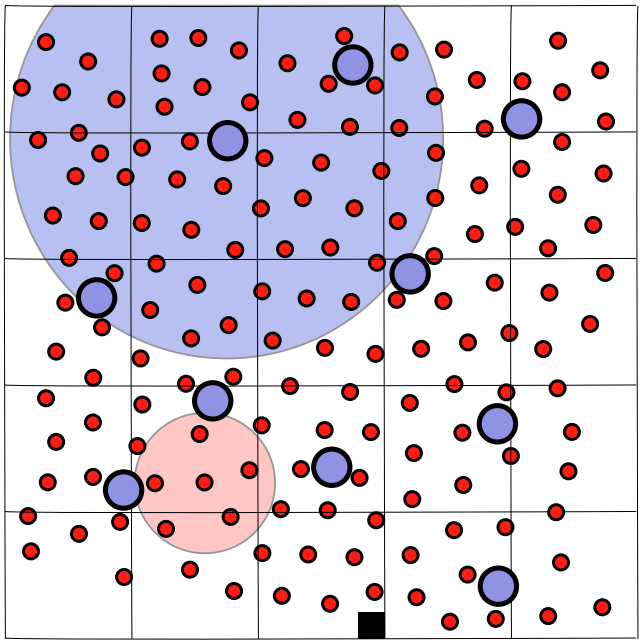


Figure 4.1: Heterogenous network structure with two types of nodes

- The sink is able to perform authenticated broadcasts using a scheme such as μTESLA (see section 3.1.2).
- Small nodes are able to form clusters and elect cluster heads.
- We assume that the attacker has physical access to both large and small nodes.

4.2 Brief overview

The filtering scheme can be divided in two phases.

1. A report is created similarly to the "statistical en-route filtering" scheme desribed in section 3.2.1, but the report is sent to a nearby large node instead of the sink. This large node is called *transition node of the report* R and denoted TN_R.

   transition node TN_R

2. The transition node attaches a public-key signature to the report and sends it to the sink. Large nodes en-route to the sink verify the signature.

We can see that both phases require different routing protocols: The first phase requires a protocol that sends messages towards the transition node. We call this protocol *transition routing protocol* (TRP). In the second phase, the routing protocol must send messages towards the sink. We call this protocol *sink routing protocol* (SRP). The SRP must prefer large nodes for the route to the sink. We assume that both routing protocols already exist. The development of such protocols is not in the scope of this thesis.

TRP

SRP

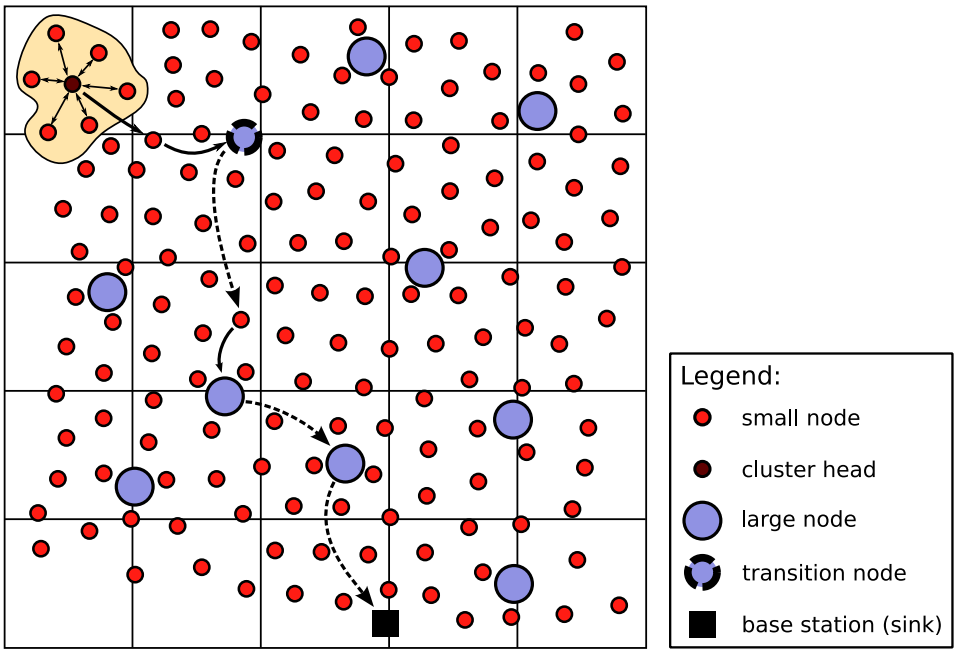


Figure 4.2: Overview over the hybrid filtering scheme

4.3 Keys and notation

Apart from the keys presented in LEAP, we employ three different classes of keys:

LBI-key, $K^{lbi}_{\mathbf{a}}$

- *Location-based individual keys (LBI-keys)* are symmetric keys shared between each small node and the sink. These keys are bound to a location as in the scheme described in section 3.2.2. The purpose of these keys is to endorse a report with a MAC that is verified by the sink. These keys are denoted $K^{lbi}_{\mathbf{a}}$ for a node $\mathbf{a}$

LE-key, $K^{le}_{\mathbf{a},\mathbf{A}}$

- *Local endorsement keys (LE-keys)* are symmetric keys shared between each small node and nearby large nodes. These keys are also bound to a location. Their purpose is to endorse a report with a MAC that is verified by the transition node. The LE-key, that is shared by a small node $\mathbf{a}$ and a large node $\mathbf{A}$, is denoted $K^{le}_{\mathbf{a},\mathbf{A}}$.

En-route key pair, $\kappa^{sec}_{\mathbf{A}}$, $\kappa^{pub}_{\mathbf{A}}$

- *En-route key pairs* are public-private key pairs. Each large node $\mathbf{A}$ stores the secret key of an en-route pair ($\kappa^{sec}_{\mathbf{A}}$) and is able to retrieve the public key ($\kappa^{pub}_{\mathbf{A}}$) of every other large node in the network using one of the methods discussed in section 3.4.1. The sink has access to all public keys as well. $\kappa^{sec}_{\mathbf{A}}$ is used by the transition node to sign reports and $\kappa^{pub}_{\mathbf{A}}$ allows the en-route nodes and the sink to verify the signature.

In the rest of the chapter, the following notation is used:

$\mathrm{MAC}_K(M)$

- $\mathrm{MAC}_K(M)$ denotes a message authentication code computed from message M with the key K.

- $\langle M \rangle_{\mathbf{A}}$ denotes a message M that has been signed with the private key $\kappa_{\mathbf{A}}^{sec}$. $\langle M \rangle_{\mathbf{A}}$
- $ID_{\mathfrak{a}}$ *and* $ID_{\mathbf{A}}$ denote the identifier string of node $\mathfrak{a}$ and $\mathbf{A}$ respectively. This is generally a binary number long enough to provide a unique identity for every node. $ID_{\mathfrak{a}}$, $ID_{\mathbf{A}}$

The following system parameters can be adjusted:

- L is the number of large nodes that every small nodes sets up connections to (i.e. the number of LE-keys stored by each small node) L
- T is the number of small nodes needed to endorse a report. T

4.4 Key establishment

As in LEAP, we assume that after deployment of the network there is a timespan in which no attacker can compromise any nodes. During this timespan, *individual keys*, *pairwise keys* and *cluster keys* are established as described in LEAP (see section 3.1.1). For the setup of LBI-keys and LE-keys, we reuse LEAP's global initialization secret K_I.

LBI-keys For the computation of the LBI-keys, we use a cryptographical hash-function in MAC-mode with the master key K_I as MAC key. During the initialization phase, every small node computes

$$K_{\mathfrak{a}}^{lbi} = \mathrm{MAC}_{K_I}(ID_{\mathfrak{a}}, location) \tag{4.1}$$

When the sink receives a report $(ID_{\mathfrak{a}}, R, location)$ with a MAC attached. It can compute the appropriate LBI-key on-the-fly and then verify the report.

LE-keys In order to set up the LE-keys, each large node $\mathbf{A}$ broadcasts a HELO message with the same transmission power as the small nodes:

$$\mathbf{A} \longrightarrow ** : (ID_{\mathbf{A}}, h_{\mathbf{A}}) \tag{4.2}$$

The value of $h_{\mathbf{A}}$ is initially zero and is increased by every forwarding node. Then the large node computes and stores

$$K_{\mathbf{A}}^{init} = \mathrm{MAC}_{K_I}(ID_{\mathbf{A}}) \tag{4.3}$$

When a small node receives a HELO from large node $\mathbf{A}$, it stores and retransmits the message

- if it has stored less than L HELOs so far or
- if it has stored L HELOs already, but $h_{\mathbf{A}}$ is smaller than the hop count of one of the stored HELOs. If this is the case, it looks for the stored entry with the largest hop count and replaces it with the current HELO message.

After this procedure, each small node stores HELO messages from the nearest L large nodes. For each of the stored messages, the small node $\mathbf{a}$ first computes $K_{\mathbf{A}}^{init}$ as in equation 4.3 and then

$$K_{\mathbf{a},\mathbf{A}}^{le} = \mathrm{MAC}_{K_{\mathbf{A}}^{init}}(ID_{\mathbf{a}}, location) \tag{4.4}$$

After the initialization phase, the small nodes remove K_I and all $K_{\mathbf{X}}^{init}$ securely from their memory. The large nodes only remove K_I and keep their $K_{\mathbf{X}}^{init}$. When a large node $\mathbf{A}$ verifies the MAC-signature of a message, it can compute $K_{\mathbf{a},\mathbf{A}}^{le}$ on-the-fly as long as $ID_{\mathbf{a}}$ and the location record is sent along.

Note that an attacker who compromises $\mathbf{A}$ can generate valid $K_{\mathbf{A},\mathbf{x}}^{le}$ for all small nodes $\mathbf{x}$. The knowledge of these keys is however irrelevant because LE-keys are pairwise keys between one small node and one large node. The compromised large node is the only node which uses this key to verify MACs.

The setup of LE-keys can be combined with the creation of the routing tree of the transition routing protocol (TRP). We also assume that the TRP only routes messages to large nodes for which an LE-key exists. This limits the impact of pDoS-attacks on this protocol by limiting the distance of the furthest target node.

Discrete locations We have assumed so far, that each small node can determine its own location. The location information mainly depends on the scenario and the sensor model. For example, in the forest-fire-detection scenario, x and y coordinates would be an appropriate representation. A structural health monitoring system would have a z coordinate as well. It might also have a completely different system, that is based on the structure of the building's carrier plates. The fire-alarm of a building might have location information based on rooms and hallways.

It is difficult to select an appropriate level of detail for the representation of locations: On the one hand, the sensor network application may require accurate location information. On the other hand, a higher level of detail increases the memory consumption caused by the localized keys. When a cluster of sensor nodes endorses a report, each sensor must store keys that are valid for the report's location. If multiple locations are within the sensor range of a node, then the node must generate keys for all of them. If the location information is too detailed, the sensor node has to store more keys. This means that less memory is available for the actual sensor network application.

application-, key-accuray, loc_{app}, loc_{key}

We call the location accuracy needed for the application *application-accuracy* (loc_{app}) and the accuracy needed for localized keys *key-accuracy* (loc_{app}) and introduce a unique mapping

$$lm : loc_{app} \longmapsto loc_{key} \tag{4.5}$$

that maps a location record with application-accuracy to a location record with key-accuracy. The report only contains a location record with application-accuracy. When a node uses a localized key, it applies the mapping function to the location record in order to compute the key-accuracy record.

In this section, we assume that a node can only endorse reports at a single location and that the mapping lm is the identity function.

4.5 Report generation

The generation of a report is split into two major phases that we call *local generation* and *transition*.

Local generation When an event occurs, we assume that a number of small sensor nodes can verify the measurement. We also assume that the nodes have elected cluster heads among themselves. The local generation process is initiated by a cluster head that measures the event:

1. The *cluster head* generates a report based on the measurements and sends it, encrypted with its cluster key, to the broadcast address: cluster head, CH

$$CH \longrightarrow * : \ R = (location, value, \text{timestamp}) \tag{4.6}$$

2. Each neighboring node $\mathbf{b}_x$ receiving the report performs a plausibility check with its own sensors and returns

$$\mathbf{b}_x \longrightarrow CH : \ SM_x, \mathcal{TN}_{\mathbf{b}_x} \tag{4.7}$$

where $\mathcal{TN}_{\mathbf{b}_x} \stackrel{def.}{=} ID_{\mathbf{A}_1}, ID_{\mathbf{A}_2}, \ldots$ is the set of potential transition nodes for which $\mathbf{b}_x$ store LE-keys and $SM_x \stackrel{def.}{=} \text{MAC}_{K^{lbi}_{\mathbf{b}_x}}(R)$. $\mathcal{TN}_{\mathbf{b}_x}$, SM_x

3. The cluster head chooses a transition node $\mathbf{A}$ that is found in at least T sets $\mathcal{TN}_{\mathbf{b}_x}$. Then it chooses T *helper nodes* $\mathbf{b}_i$ with $\mathbf{A} \in \mathcal{TN}_{\mathbf{b}_i}$, compresses their MACs into a bloom-filter, and answers: helper node

$$CH \longrightarrow \{\mathbf{b}_1 \ldots \mathbf{b}_T\} : \ SMAC \ = \ \text{bloom}(\{SM_1, \ldots, SM_T\}) \tag{4.8}$$

4. Each helper node $\mathbf{b}_i$ checks whether $\text{MAC}_{K^{lbi}_{\mathbf{b}_i}}(R) \in SMAC$ and then returns

$$\mathbf{b}_i \longrightarrow CH : \ LM_i = \ \text{MAC}_{K^{le}_{\mathbf{b}_i,\mathbf{A}}}(R, SMAC) \tag{4.9}$$

to the cluster head.

5. The cluster head computes $LMAC = \text{LM}_1 \oplus \ldots \oplus \text{LM}_T$ and sends the whole report to the transition node $\mathbf{A}$:

$$CH \longrightarrow \mathbf{A} : \ (R, ID_{\mathbf{b}_1}, \ldots, ID_{\mathbf{b}_T}, SMAC, LMAC) \tag{4.10}$$

Transition When the transition node $\mathbf{A}$ receives the message, it generates the LE-keys $K^{le}_{\mathbf{b}_1,\mathbf{A}}, \ldots, K^{le}_{\mathbf{b}_T}$ from the node IDs $ID_{\mathbf{b}_1}, \ldots, ID_{\mathbf{b}_T}$, the location of R and its initialization secret. $K^{init}_{\mathbf{A}}$ (see section 4.4). Then it verifies that

$$LMAC \stackrel{?}{=} \bigoplus_{i=1}^{T} \text{MAC}_{K^{le}_{\mathbf{b}_i,\mathbf{A}}}(R, SMAC) \tag{4.11}$$

If the LMAC doesn't match, the report is dropped because it is assumed to have been modified or injected. If the LMAC matches, the transition node computes a public-key signature of $(R, ID_{\mathbf{b}_1}, \ldots, ID_{\mathbf{b}_T}, SMAC)$ using its private key $\kappa^{sec}_{\mathbf{A}}$ and sends

$$\langle R, ID_{\mathbf{b}_1}, \ldots, ID_{\mathbf{b}_T}, SMAC \rangle_{\mathbf{A}} \tag{4.12}$$

to the sink.

4.6 Report verification

The verification of the report is done in multiple places:

En-route verification One assumption about the routing protocol is that it favors large nodes en-route to the sink. Every large node that forwards the report verifies the authenticity of the public-key signature using $\kappa_{\mathbf{A}}^{pub}$. If the signature is wrong, the report is dropped.

Sink verification The sink also verifies the public-key signature and drops the report if it is wrong. Additionally, the sink computes the keys $K_{\mathbf{b}_1}^{lbi}, \ldots, K_{\mathbf{b}_T}^{lbi}$ from $ID_{\mathbf{b}_1}, \ldots, ID_{\mathbf{b}_T}$, the location of R and the initialization secret K_I. For each key, it verifies

$$\forall i \in \{1, \ldots, T\} : \quad \mathrm{MAC}_{K_{\mathbf{b}_i}^{lbi}}(R) \stackrel{?}{\in} SMAC \tag{4.13}$$

If this condition is true, the report is accepted. If not, the report is rejected and further actions are taken.

4.7 Immediate action

degenerated report

We call a report with a valid public-key signature and an invalid SMAC a *degenerated report*. Such a report can be created by the network due to the following reasons:

1. **The transition node has been compromised.** This allows the attacker to create signed reports, but without the correct LBI-keys of a cluster, he cannot create a correct SMAC. Nevertheless, he can still launch a pDoS-attack to drain energy from the forwarding nodes.

2. **Some nodes of the endorsing cluster nodes are compromised.** The above assumption alone would enable the attacker to frame an innocent large node **A** by compromising a single small node. We call such an attempt a *frameup attack*. The attacker can perform the following actions during the local report generation process (see section 4.5) to achieve his goal:

frameup attack

   - In step 2 the compromised node returns a wrong (or arbitrary) value for SM_i. As a result, the cluster head would then compute an invalid SMAC.
   - In step 4 the compromised node computes $\mathrm{MAC}_{K_{\mathbf{b}_i,\mathbf{A}}^{le}}(R, SMAC)$ correctly and returns it to the cluster head although $SM_i \notin SMAC$.

   The transition node would then receive a report with a correct $LMAC$ and an incorrect $SMAC$. Since it cannot verify the $SMAC$, it attaches a valid public-key signature.

It is impossible to determine the exact node that is responsible for the degenerated report. However, since the bloom filter allows a membership check of distinct elements, we can determine the set $\mathcal{G}_R$ of small nodes that might be guilty of forging the *SMAC* of report R as $\mathcal{G}_R$

$$\mathcal{G}_R = \left\{ \mathfrak{a} \,.\, ID_{\mathfrak{a}} \in \{ID_{\mathfrak{b}_1}, \ldots, ID_{\mathfrak{b}_T}\} \wedge \mathrm{MAC}_{K^{lbi}_{\mathfrak{a}}}(R) \notin SMAC \right\} \tag{4.14}$$

The set $\mathcal{G}_R$ contains the IDs of all nodes whose SM_i could not be found in *SMAC*. Those MACs were presumably forged. We also define the set $\overline{\mathcal{G}_R}$ $\overline{\mathcal{G}_R}$

$$\overline{\mathcal{G}_R} \stackrel{def.}{=} \{ID_{\mathfrak{b}_1}, \ldots, ID_{\mathfrak{b}_T}\} \setminus \mathcal{G}_R \tag{4.15}$$

An exact identification of the compromised node is impossible at this moment, but the attacker definitely knows the keys:

$$K^{le}_{\mathfrak{x},TN_R} \text{ such that } \mathfrak{x} \in \mathcal{G}_R \tag{4.16}$$

If the attacker has compromised TN_R, he can generate $K^{le}_{\mathfrak{x},TN_R}$ for all small nodes $\mathfrak{x}$. If he has compromised all small nodes $\mathfrak{x} \in \mathcal{G}_R$, he can extract the keys from these nodes. Thus the immediate reaction of the sink to a degenerated report is to broadcast a revocation instruction for these LE-keys. Afterwards TN_R refuses to act as a transition node for any node in $\mathcal{G}_R$ and all nodes $\mathfrak{a} \in \mathcal{G}_R$ remove $K^{le}_{\mathfrak{a},TN_R}$ from their memory.

4.8 Long-term analysis

In order to pinpoint the responsible nodes, the sink stores an *evidence record*. evidence record

$$\mathcal{E}_R = (TN_R, \mathcal{G}_R, \overline{\mathcal{G}_R}) \tag{4.17}$$

for each degenerated report it receives. From the set of all evidence records, the sink derives a scoring $score_{\mathfrak{a}}$ for each node $\mathfrak{a}$. If the $score_{\mathfrak{a}}$ exceeds a treshold $score^{max}$ the sink sends an authenticated broadcast message through the network ordering every node to isolated $\mathfrak{a}$ from the network.

Computing the score A simple way to compute the score of a node $\mathfrak{a}$ (or $\mathbf{A}$) is the *static scoring scheme*. In this scheme, the score is exactly the number of evidences $\mathcal{E}_R$ with $\mathfrak{a} \in \mathcal{G}_R$ for small nodes or $\mathbf{A} = TN_R$ for large nodes. static scoring scheme

A more sophisticated approach is the *dynamic scoring scheme* which also examines the number of false SMAC-components in a report R. With this information, we can determine the effort that an attacker has to make to create R. The nodes involved in actions with a lower effort receive a higher score increment: dynamic scoring scheme

Suppose $\mathcal{G}_R$ contains g elements. An attacker has to perform one of the following actions to create such a degenerated report R:

1. Compromise the transition node TN_R and modify the SMAC of an existing report such that g components are falsified. This means that the attacker must wait until a real report is generated and the compromised node is chosen as the transition node.

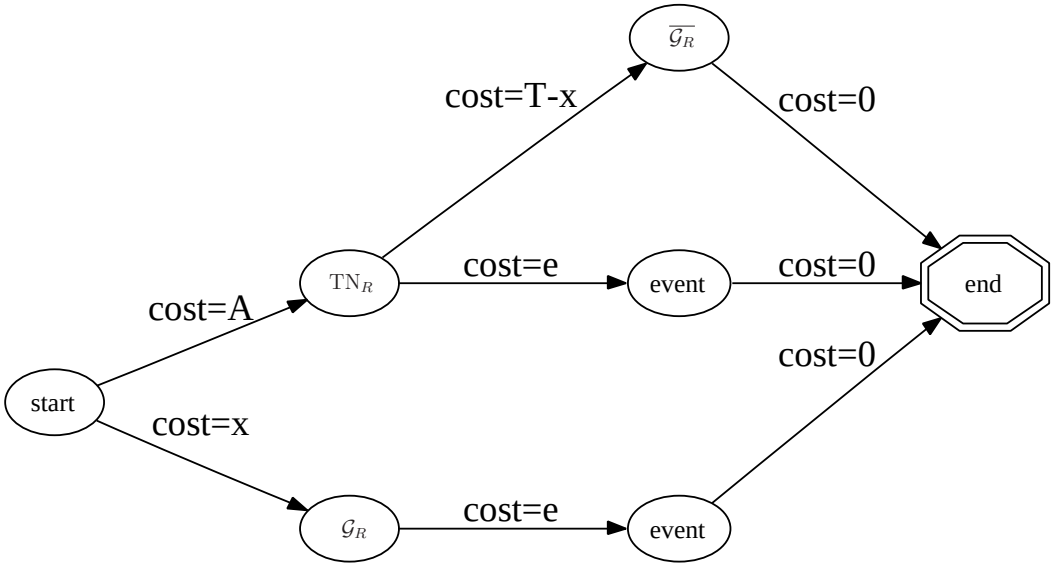


Figure 4.3: Actions leading to a degenerated report

2. Compromise TN_R and $T-g$ small nodes ($\overline{\mathcal{G}_R}$). The attacker can then use the stolen LBI-keys to generate $T-g$ correct SMAC-components and choose arbitrary values for the remaining ones. He can also generate the signature using the secret key $\kappa^{sec}_{TN_R}$.

3. Compromise g small nodes ($\mathcal{G}_R$) and falsify the SMAC-components of these nodes. This is only possible if a report is being generated by a cluster containing the compromised nodes. Of course, if the cluster head is compromised, the attacker can choose other compromised nodes as helper nodes.

The effort of each method can be composed from cost-values for the basic actions: Compromising one small node (cost= 1), compromising one large
A node (cost= A) and waiting for an event (cost= e). The parameters A and e
e can be adjusted to reflect the situation in the real deployment scenario. For example, A can be increased if large nodes are not easily accessible.

Figure 4.3 illustrates how these actions lead to a degenerated report. We can now model the effort that an attacker has to make:

$\text{eff}(g,N)$ **Definition: 4.1** *Let* $\text{eff}(g,N)$ *be the minimal effort to create a degenerated report with* g *false SMAC-components, based on compromising a set* N *of sensor nodes.* N *can be either* $\{TN_R\}$, $\mathcal{G}_R$ *or* $\overline{\mathcal{G}_R}$. *In figure 4.3, the value of* $\text{eff}(g,N)$ *is the length the shortest path from "start" to "end" via node* N.

Based on this definition, $\text{eff}(g, N)$ resolves to:

$$\begin{aligned} \text{eff}(g, \mathcal{G}_R) &= g + e && (4.18) \\ \text{eff}(g, \overline{\mathcal{G}_R}) &= A + T - g && (4.19) \\ \text{eff}(g, \{TN_R\}) &= A + \min(e, T - g) && (4.20) \end{aligned}$$

In order to give high scores to nodes that are involved in a sequence of actions with low effort, we define a score increment $score^+_N(g)$ for $N \in \{TN_R, \mathcal{G}_R, \overline{\mathcal{G}_R}\}$: $score^+_N(g)$

$$\begin{aligned} score^+_{\mathbf{a}}(g) &= \frac{1}{\text{eff}(g, \mathcal{G}_R)} = \frac{1}{g + e} \quad \forall \mathbf{a} \in \mathcal{G}_R && (4.21) \\ score^+_{\mathbf{b}}(g) &= \frac{1}{\text{eff}(g, \overline{\mathcal{G}_R})} = \frac{1}{A + T - g} \quad \forall \mathbf{b} \in \overline{\mathcal{G}_R} && (4.22) \\ score^+_{\mathbf{A}}(g) &= \frac{1}{\text{eff}(g, \{TN_R\})} = \frac{1}{A + \min(e, T - g)} \quad \text{for } \mathbf{A} = TN_R && (4.23) \end{aligned}$$

The score of a node can then be computed as the sum of $score^+(g)$ over all evidences $\mathcal{E}_R$. A node $\mathbf{a}$ is removed from the network if

$$score_{\mathbf{a}} > score^{max} \tag{4.24}$$

The threshold $score^{max}$ should be adjusted such that at least two evidences are necessary for a node to be removed from the network.

Chapter 5

Implementation

In order to evaluate the scheme presented in chapter 4, we have performed simulations on a sensor network simulator.

Various sensor network simulators already exist, so we first attempted to implement the simulation on an existing platform. Section 5.1 describes the process of deciding which platform to use.

Section 5.2 gives details about the implementation of our own simulator. Section 5.3 describes the implementation of HEFS in the simulator. Chapter 5.4 describes the protocol that is compared to HEFS in the evaluation. Finally, section 5.5 presents the configurations that were used to gather the statistical results presented in the evaluation.

5.1 Platform decision process

A number of assumptions and required components were defined in chapter 4. Some of these prerequisites and assumptions must be reflected in the simulation while others are only relevant in a real deployment scenario. For example, LEAP provides authenticity and data confidentiality between neighboring nodes, and the μTESLA protocol allows authenticated broadcasts. For the evaluation of HEFS, it is adequate to simulate the overhead caused by these protocols, but it is not necessary to simulate the protocol in detail. It is easier to modify the attacker's behavior to *not* attempt attacks that would be prevented by LEAP or μTESLA.

This section highlights the guidelines that were used in the platform decision process and briefly evaluates two simulation platforms according to these guidelines.

Heterogeneous network The assumed network structure in chapter 4 contains different types of sensor nodes. Such a network structure must be supported by the simulator:

1. Sensor nodes must be able to have different properties concerning the battery model, the radio model, CPU resources and memory.

2. Sensor nodes must be able to run different applications.

The second item is particularly important because it would be impossible to implement different functions for small nodes and large nodes otherwise.

Routing protocols HEFS uses two different routing protocols with special properties:

- The **sink routing protocol (SRP)** performs a routing towards the sink. It prefers large nodes for routing and uses their larger radio transmission range to create shortcut paths.

- The **transition routing protocol (TRP)** performs a routing from a small node towards the L nearest large nodes.

A basic implementation of these protocols is needed for the evaluation of the scheme, but it is not necessary to use a sophisticated protocol. Simulated nodes do not fail and we assume that compromised nodes do not attempt to perform black hole or selective forwarding attacks.

Crytpographic operations Cryptographic operations are used when message authentication codes (MAC) and public-key signatures are attached to reports and verified. Attacks on the cryptographic algorithms are not the primary focus of this thesis, so the actual algorithms are not important. Even so, it is important that false MACs are not accidently accepted. For this goal, it is beneficial to use real MAC algorithms. Since the implementation of MAC algorithms is not the primary goal of this thesis, the existence of a cryptographic library is a requirement.

The algorithm for public-key signatures can be replaced by a dummy algorithm. The algorithms presented in section 3.4.2 are relatively new, so there are no implementations in standard cryptographic libraries. Using existing algorithms with a larger signature length would distort the statistical results. As a work-around we use symmetric algorithms instead and let the simulated attacker behave as if they were asymmetric. The signature can then be padded to the length of the public-key signature proposed for the scheme.

Gathering statistics The goal of the simulation is to collect data about the number of transmitted bytes and messages and about the energy consumption of sensor nodes. The simulation platform must provide mechanisms to collect these data.

Development tools Implementing the simulation is only one of the goals of this thesis. In order to reduce the time spent on this goal, the quality of the development and debugging tools is relevant for deciding which simlution platform to use.

Advantages of existing frameworks An alternative to using an existing platform is writing a new simluation framework from scratch. In this case all features can be implemented specificly for HEFS. It is important to point out the advantages of using an existing framework:

- **Reuse of code:** A simulation framework already provides a code base for simulations. Physical-layer and link-layer protocols may already be available as well as other general mechanisms. This may shorten the time needed for development.

- **Comparability and credibility:** Simulations generally do not model every aspect of reality. In order to implement a simulation framework, decision have to be made about what is exactly simulated and what is not. By using an existing, commonly known framework, it is clear to the reader, which decisions have been made. It is also clear, that the "reduced reality" of the simulation has not been adapted to the simulated protocol. Finally, the simulation results of different protocols are more comparable if both simulations have been run in the same framework.

5.1.1 Evaluated Frameworks

Two simulation frameworks were considered for the implementation of HEFS.

JSim JSim [30] is a simulation framework developed at the the Distributed Realtime Computing Laboratory [3] of the Ohio State University. It is written in Java and offers a component-based framework for different kinds of simulations. The sensor network extension provides implementations of the lower networking layer protocols (Physical layer, Link layer) based on the IEEE802.11 standard and the routing protocols *Ad-hoc On-demand Distance Vector Routing* (AODV) [25] and *Dynamic Source Routing* (DSR) [19]. An energy model for batteries and power line is included. Mobile sensor nodes are supported and real sensor nodes can be attached in to a running realtime simulation.

We did not implement HEFS in JSIM for the following reasons: AODV and DSR are routing protocols for Mobile Ad-Hoc Networks. They are optimized to handle frequent changes in the routing table caused by the movement of mobile nodes, which is out of scope of this thesis. On the other hand, no distinction is made between different node types. This means that these protocols cannot be used for the sink routing protocol without adaption.

Furthermore, a complete simulation of all network layers according to the IEEE802.11 standard is unnecessary in our evaluation scenario. We presumed that implementing the scheme in JSim would take considerably longer than implementing a new simulation platform.

TOSSIM The TOSSIM [22] framework is a part of the TinyOS [18] distribution. Its goal is to test real TinyOS applications in a simulated environment.

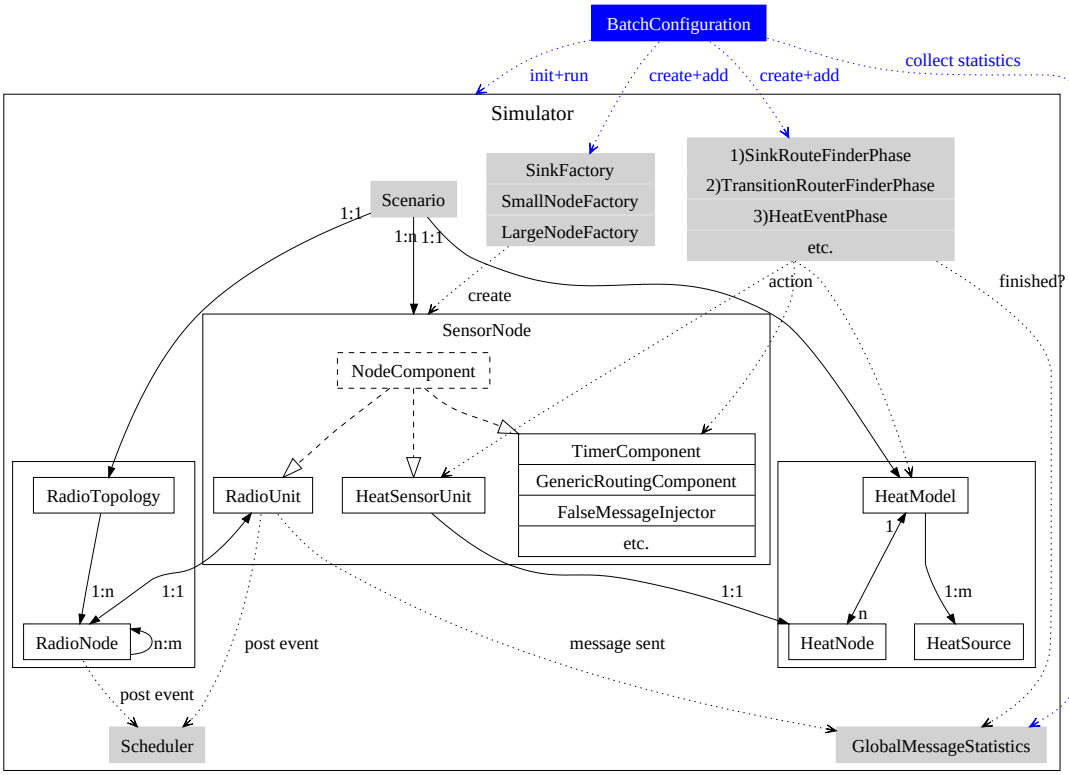


Figure 5.1: Overview over the implementation of the simulator

TinyOS applications are written in the component-based programming language `nesc` [15]. Components for two basic routing protocols (Dissemination and Collection) are available, but they do not have the properties needed for a heterogeneous network. The radio layer must be pre-computed and a connectivity graph is given to the simulator at the beginning of the simulation.

TOSSIM does not support multiple sensor node applications in one simulation, which makes it unsuitable for modelling a heterogeneous network. Another point against choosing TOSSIM was the lack of development tools. Implementing the scheme in `nesc` would have taken longer than intended.

5.2 The simulator

Due to these problems, we decided to implement a new simulation framework in Java. Section 5.2.1 describes the implementation of the simulation framework. Section 5.2.2 presents details of the prerequisite protocols.

5.2.1 The simulation framework

Figure 5.1 provides an overview over the components used in the simulation. The `Simulator` box represents the `Simulator` object that runs one simulation. Gray boxes are objects that are directly referenced by the `Simulator`. Solid lines denote connections between the components of the scenario model. The

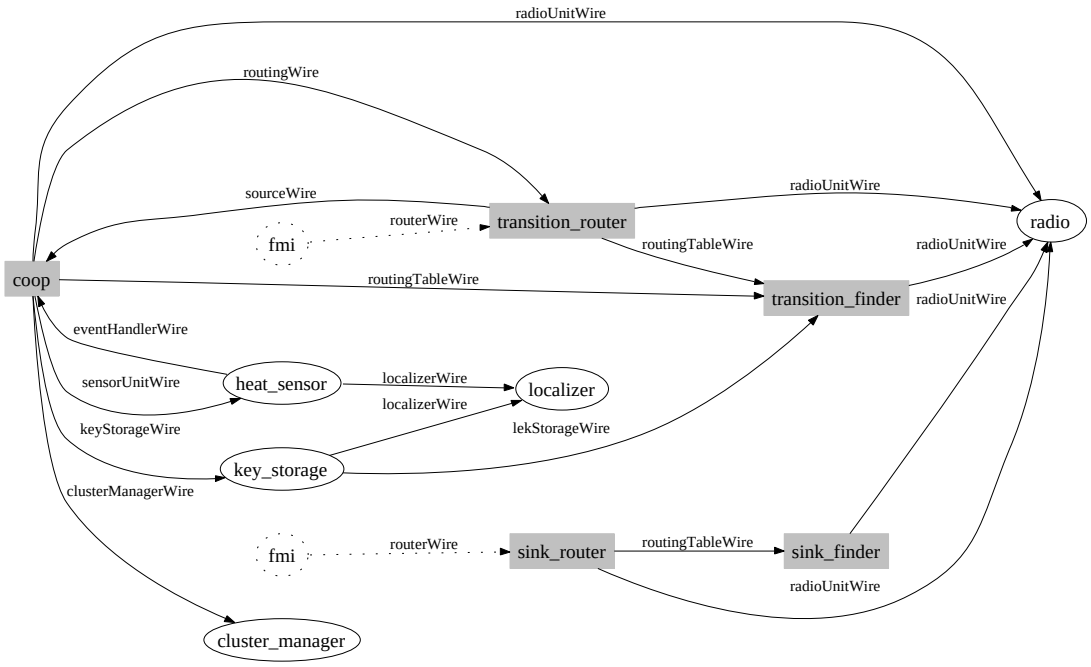


Figure 5.2: Node components and wires of a small node.

labels `1:1`, `1:n` and `n:m` denote one-to-one, one-to-many and many-to-many relations respectively. Dotted lines show function calls, and dashed lines indicate class-inheritance. The parts of this figure are presented in detail below.

Sensor nodes and node components Node components are the lowest abstraction level of the simulation framework. A component encapsulates certain functions of the sensor node. Each component is implemented in a subclass of `NodeComponent` and can be connected to other components. The connection is called a *wire* and is realized through instance variables that match the pattern `*Wire`. Figure 5.2 shows the components and the wiring of a small node in a heterogeneous network. There are implicit wires from the RadioUnit to the components marked as gray boxes. These wires have been omitted for the sake of the clarity. The `TimerComponent` has been ommitted for the same purpose. The false message injector (FMI) is optional and can be connected either to the `sink_router` or to the `transition_router`.

wire

Note that all components are software components. The concept of TinyOS and `nesc` defines a software architecture that is similar to the composition of hardware-chips (i.e. components and subcomponents that are connected by wires). We use this concept in our own simulator.

Creation of sensor nodes Different types of sensor nodes can be created by composing different components. This creation process is performed by a *node factory* (i.e. `SmallNodeFactory`, `LargeNodeFactory`, `SinkFactory`) using the following steps:

node factory

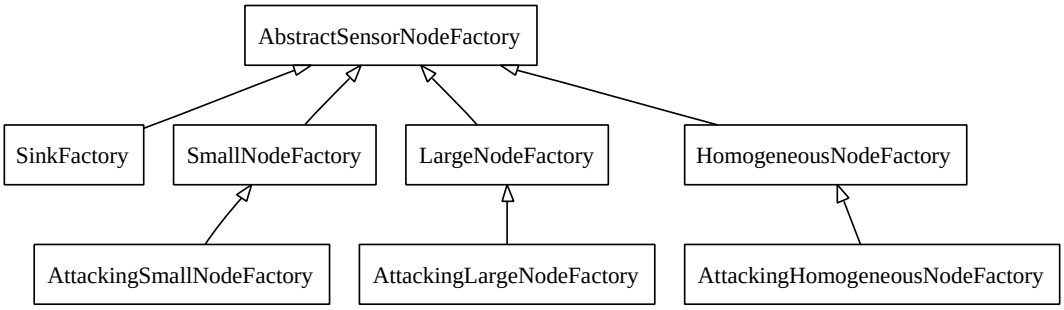


Figure 5.3: Class hierarchy of the node factories.

Field:	destination	type	payload
bytes:	2 bytes	1 byte	

Figure 5.4: Basic message structure of all radio messages

1. Create a `SensorNode` object.
2. Create all node components and add them to the `SensorNode`
3. Connect the wires of each component to the appropriate target components.
4. Verify that all wires of all components are connected.

Part of the wiring is done automatically by the `SensorNode` class: Components, that implement the Java interface `MessageHandler` or `MessageSnooper` are automatically wired to the radio unit. In figure 5.2 these components are shown in gray boxes.

Figure 5.3 shows the inheritance structure of the implemented factories. The classes `LargeNodeFactory` and `SmallNodeFactory` are used in the heterogeneous network. The `HomogeneousNodeFactory` creates nodes for the comparative protocol (see section 5.4). The `Attacking*` factories add a *false message injector* (FMI) component to the nodes. This unit is used by the attacker to perform pDoS-attacks.

false message injector, FMI

Radio model The radio model of the simulation is a graph of `RadioNode` objects. Each `RadioNode` represents "the air around a sensor node". The `RadioNode` is connected to the radio unit of the sensor node and to `RadioNodes` that are in radio range (see figure 5.5).

Messages that are sent over the radio model inherit the abstract class `Message`, which defines the basic structure of the message header (see figure 5.4).

The first field (`destination`) is either the ID of the destination node or the broadcast address (`DEST_BROADCAST = -1`). The second field (`type`) is the message type. It determines the structure and size of the message.

In order to simulate the message overhead caused by LEAP (see section 3.1.1), we add 8 byte to every message to represent the attached message

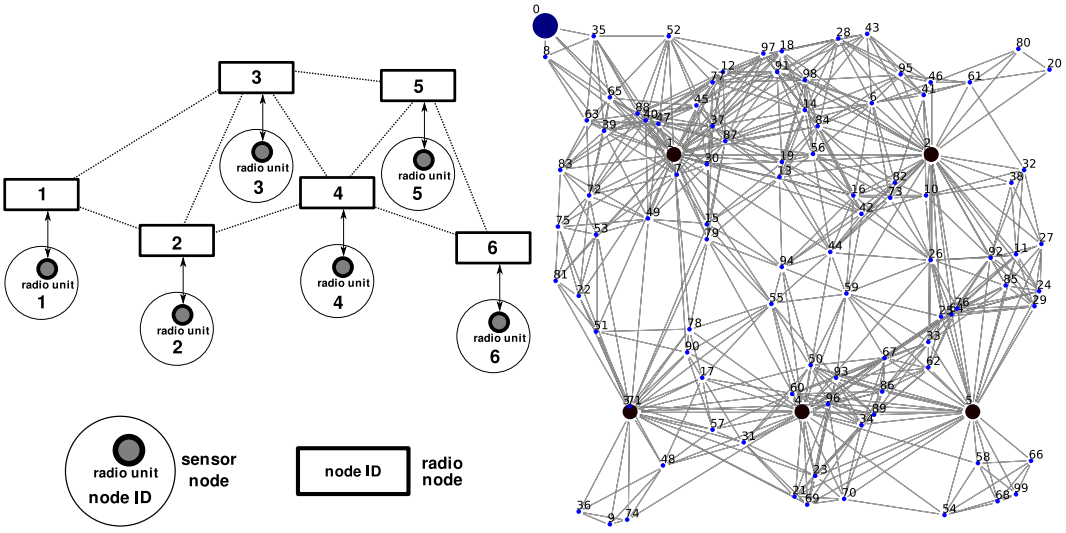


Figure 5.5: Topology network of `RadioNode`s

authentication code. Every message must also contain the ID of the sending node because the receiving node must be able to choose the correct MAC-key for the verification.

The following example illustrates the process of sending a message: We assume that in figure 5.5, node 4 has just received a message from node 6 which should be routed to node 1. For this purpose, the routing component (e.g. "transition_router") creates a new message with the payload of the received message and sends it to node 2 as the next hop. Node 4 now performs the following steps:

1. The routing component creates the new message object of the correct type and sets 2 as destination address.

2. It passes the message to the radio unit through the `radioUnitWire` (see figure 5.2).

3. The radio unit performs some additional action according to the link-layer protocol presented in section 5.2.2. It then passes the message to the connected `RadioNode`.

4. The `RadioNode` notifies the adjacent `RadioNode` objects (2, 3, 5, 6) of the beginning of the transmission. Each of the receiving nodes increases a counter of pending transmissions. If other transmissions are already going on, all transmissions are discarded and the receiving radio units are notified of the collision.

5. The sending `RadioNode` posts a scheduler event that is executed when the message transmission is over. The time of this event is determined from the message size.

6. When the event is executed, the sending `RadioNode` notifies the adjacent nodes of the end of the transmission and delivers the message object. The receiving `RadioNode` objects decrease their counter of pending transmissions and pass the message to the radio unit of the connected sensor node as long as no collision has taken place.

7. The radio unit of each receiving node determines whether it is the intended recipient of the message and delivers the message to the appropriate node components.

Node components that process incoming messages must implement either the interface `MessageHandler` or `MessageSnooper`. The implemented method `handledType()` returns the message type that can be handled by this component. The radio unit maintains a table that links the message type to the appropriate node component. A `MessageSnooper` receives all messages of its specified type. A `MessageHandler` receives a message only if the node ID matches the destination address or the destination address is the broadcast address.

In order to simulate transmissions with different ranges (for large nodes and small nodes), each sensor can send data either in a normal transmission mode or in a boost mode. The boost range of a node can be different from the radio transmission range. The sink routing protocol expects large nodes to have twice the transmission range of small nodes if is is sending in boost mode[1].

The connections of the radio topology are loaded from a definition file (see appendic C) and not computed during the simulation. As a result, mobile sensor nodes cannot be simulated with this framework.

Sensor model The developed scheme makes the assumption that sensor nodes are able to verify the measurements of nearby nodes. This assumption must be reflected in the simulated sensor model. In the first approach, a simplified forest-fire scenario was implemented. The sensor component (`HeatSensorUnit`) measures the temperature at the current location.

In order to seperate the sensor model from the sensor component, a `HeatNode` is attached to the `HeatSensorUnit` of each sensor node. When the sensor unit performs a measurement, it queries the `HeatNode`, which in turn asks the `HeatModel` to compute the temperature for its location. The `HeatModel` can be turned off and on by scheduler events. The `HeatSensorUnit` periodically measures the temperature and compares it to a threshold value. If the temperature is too high, it uses its `eventHandlerWire` (see figure 5.2) to invoke the creation of a report.

The temperature for a given location is computed from an *ambient heat value* $heat_0$, a *decrement factor* dec and a number of *heat sources*

ambient heat value, $heat_0$

decrement factor, dec

heat sources

$$s_i = (x_i, y_i, heat_i) \tag{5.1}$$

[1]It is possible to adapt the sink routing protocol (SRP) to a different boost range, but in the current implementation, we use a boost range that is twice the normal range.

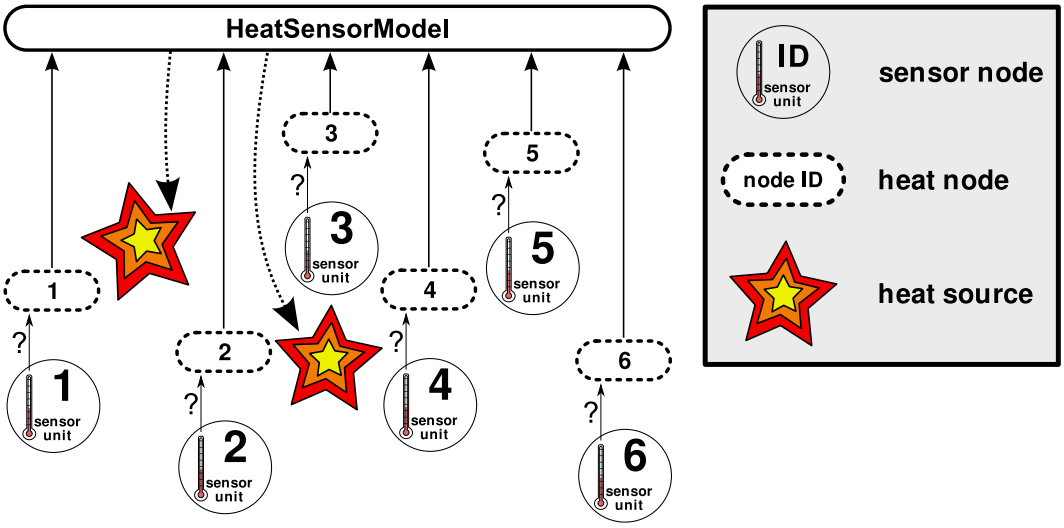


Figure 5.6: Structure of the sensor model

$t_i(x, y)$

where (x_i, y_i) is the location and $heat_i$ is the temperature of the source s_i. The temperature $t_i(x, y)$ that s_i causes at the location (x, y) depends on the distance to the source:

$$t_i(x, y) = \max(0, heat_i - dec \cdot \sqrt{(x - x_i)^2 + (y - y_i)^2}) \tag{5.2}$$

Given the heat sources $s_1, s_2, \ldots, s_n$, the final temperature $t(x, y)$ at location (x, y) is defined as

$$t(x, y) = heat_0 + \max\{t_i(x, y)\ ,\ i \in \{1 \ldots n\}\} \tag{5.3}$$

verification distance, V_d

When the sensor unit is asked to verify a measurement (as helper node) that has occured at a location (x, y), it first computes the distance to this location. If the distance is below a predefined *verification distance* V_d and the current temperature is above the critical threshold, it returns a positive result.

This model can be used in collaboration with a visualization tool to verify the correctness of the simulation. The problem however is that it is not possible to specify the exact number of reports that are created and injected into the network. In simulations, where the goal is to determine the impact of injected false messages onto the total number of transmitted bytes, it is necessary to specify the exact number of valid and false injected message. For this purpose, we have implemented a second model, the `BurnRomeHeatModel`:

In this model the temperature at any location is higher than the threshold value all the time. The sensor units are not performing periodic measurements, but they are activated at the beginning of a simulation phase (`HeatEvent-Phase`). For every activation, a single report is generated.

Scheduler and simulation phases The central entity of a simulation run is the *scheduler*. It maintains an event queue of events such as "message received by node", "timer fired in a node component" sorted by the event's time. The

scheduler

event time is specified in an abstract time unit, the *simulation cycle*. The simulator repeatedly takes the first event out of the scheduler's event queue, sets the appropriate simulation time and then runs the event. The execution causes further events to be posted to the schedulers queue. simulation cycle

The simulation is run in a single thread. Each event is processed without preemption and within a single simulation cycle. In the current implementation, events are only posted when a node component waits for a number of simulation cycles (e.g. a `TimerComponent`) or when a message is transmitted. It would be more precise to also post events that simulate the delay caused by expensive computations, but this would slow down the simulator and make the program more complex. We make the simplification, that all computations are performed instantly in order to increase the simulation speed.

A simulation is divided into a sequence of *simulation phases*. Each phase is defined by an initial action and a condidition to determine its end. The simulator verifies the condition after each execution of an event. A new phase is started, when the condition is true or when the scheduler's event queue is empty. In most cases the condition is of the form: "No message of type X has been sent during the last Y simulation cycles". Some examples of simulation phases are shown in table 5.1. simulation phase

Phase-Type	**Initial stimulus**	**Finalization condition**
`SinkRoute-FinderPhase`	The sink starts a route-finding broadcast	No broadcast messages have been sent for X simulation cycles
`HeatEvent-Phase`	A heat sensor is activated and measures a critical temperature	No routed message with the target "sink" has been sent for X simulation cycles
`PDoSAttack-Phase`	The attacker injects a false message into the network	No routed message with the target "sink" has been sent for X simulation cycles

Table 5.1: Examples of simulation phases

Gathering statistics The main goal of the simulations is gathering statistics about transmitted messages. The foundation of these statistics is the message counting component `GlobalMessageStatistics`. One instance of this class is attached to all radio units in the simulation. When a message is transmitted, the radio unit calls the method `messageSent(Message)` of the `GlobalMessageStatistics` instance, which increases the counter of transmitted messages and bytes for the appropriate message type. As a secondary task, this class keeps track of the time of the last transmission of each message type. This information is needed to verify the finalization condition of simulation phases. Similarly a method `messageReceived(Message)` is called every time a radio unit delivers a message to a `MessageHandler`-component. In the end, the transmitted and received bytes are merged into the total energy consumption of all radio units using the values from table 1.1.

For the computation of statistics, the `GlobalMessageStatistics` instance maintains a history of message counters. The current counter is increased by

the `messageSent(Message)` and `messageReceived(Message)` method. The `nextLog()` method can be called during the simulation to save the counter and initialize a new one. This is usually done in a dedicated simulation phase (`NextLogPhase`) which can be configured to discard or store the current counter. From the set of counters, the class can then compute the minimum, the maximum, the median, the average (mean) and the standard deviation.

These statistics are stored in a record and archived as well. The statistic records of multiple simulation runs are then merged into the final results. The merging is done by taking the average values of the set of statistic records.

batch configurations

Batch configurations The package `org.knappi.wsn.sim.batch` contains *batch configurations*. These classes provide the initial `main(String[] args)` methods that run different simulations. The classes in this package have to following purpose:

1. They build a simulation with a specific configuration.

2. They run multiple simulations and write statistical results into a file.

The first task consists of

1. specifying the data directory from which layout and radio-topology information is read,

2. specifying a heat model,

3. creating instances of node factories and passing them to the simulator,

4. adding simulation phases to the simulator, and

5. calling the `init()` method of the simulator. This last step uses the factories and the information from the data files to create the sensor nodes of the scenario. It also connects the sensor nodes to the radio model and the sensor model.

The abstract class `AbstractBatchConfiguration` provides functionality to run a given task on multiple data directories. Its subclass `AbstractStatisticConfiguration` adds functionality to collect statistical data beyond the scope of one simulation run and writes the output to a GNUplot-readable file. Batch configurations usually inherit one of these classes.

5.2.2 Implementation of required protocols

In order to simulate HEFS, a link-layer protocol and two routing protocols had to be implemented.

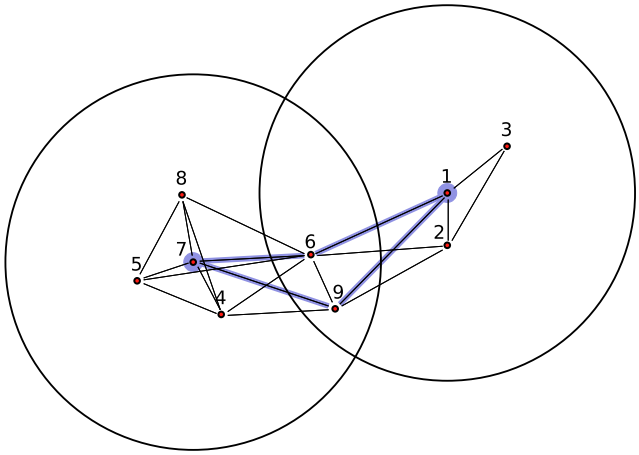


Figure 5.7: Hidden terminal problem

Medium Access Control The link-layer protocol is implemented in the radio unit. It uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) to organize the message transfer [11]:

In wired networks, a participant can listen at the bus while sending data. That way, it can detect message collisions. In wireless networks, the transmitted signal is much stronger than signals received from other locations, so a listen-while-send is not possible. CSMA/CA proposes a listen-before-send: Before sending a message, the sensor node listens to the wireless channel. If the channel is free, it stops listening and starts sending. If the channel is busy, it waits for a random period of time before attempting to send again. The upper boundary for the random number is increased exponentially after each attempt. In the simulation, the "listening" to the channel occurs in no time and there is no gap between "listening" and "sending". Nonetheless, collisions can occur in the case of a *hidden terminal* problem, which is illustrated in figure 5.7: If the nodes 1 and 7 transmit a message at the same time, both messages collide at the nodes 6 and 9. These nodes cannot receive any of the messages while 1 and 7 are not able to detect the collision.

This problem could be solved by sending an ACK message back to the sender once the sending is complete. We decided not to implement such a solution because

1. it would make the protocol significantly more complex,
2. the sending of ACK messages requires an expensive message transmission, and
3. message collisions generally occur only in certain high level protocols. These protocols can deal with the problem in a more flexible way than the link layer protocol.

Message routing and payload data In different scenarios, different payload data needs to be transmitted while the same routing protocol is used

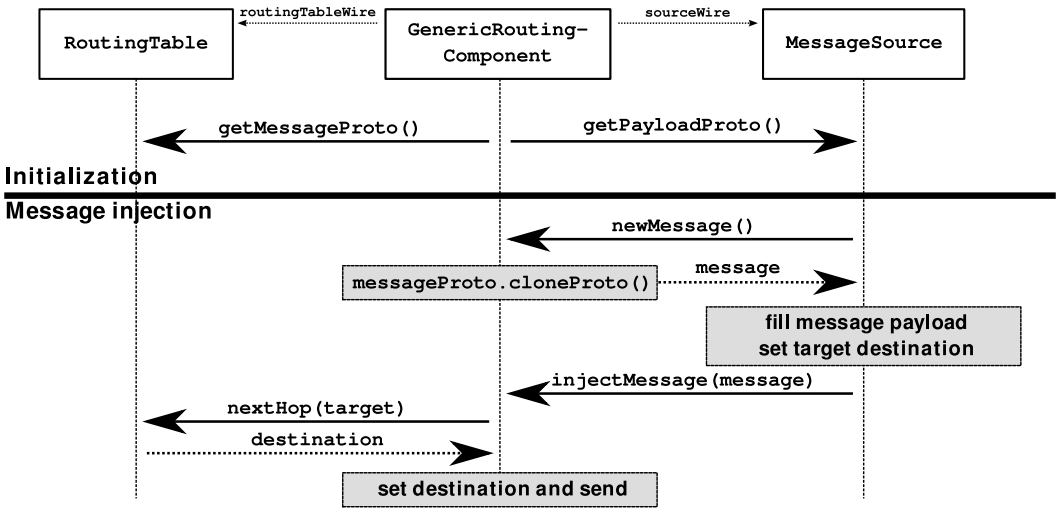


Figure 5.8: Relation of routing component, routing table and message source

to forward data. For example, the payload data of HEFS includes multiple Message Authentication Codes (MAC) and a public-key signature (see section 5.3). The comparative scenario of a homogeneous network only includes a single MAC and no public-key signature (see section 5.4). Yet, the same routing protocol is used for both messages. In order to enable code-reuse in the simulation, the payload data has been separated from the routing data.

In the design of the simulated routing components, the payload is specified by the component that injects messages into the router (i.e. the `MessageSource`) while the message type is declared in the implementation of the `RoutingTable`. Figure 5.8 illustrates the interaction of `RoutingTable`, `GenericRoutingComponent` and `MessageSource` during the initialization and during the message injection.

Sink routing protocol One of the two implemented `RoutingTable` components is the *sink route finder*. This protocol uses a gradient based approach to forward messages from any node in the network to the sink: During the `SinkRouteFinderPhase`, the sink floods the network with a broadcast message that contains a hop-counter and the one-hop sender-address. Each node re-broadcasts this package and updates its routing table.

In order to support the requirement that large nodes be able to use their extended radio range, the protocol is modified to transmit not only the node ID of the sending node, but also a list of nodes that have re-transmitted this message before. The structure of this message is displayed in figure 5.9. The field `lasthops[0]` is the sender of the message, `lasthops[1]` the previous sender and so on.

The routing table consists of the following fields:

- `my.hops`: This field contains the number of hops to the sink.
- `my.nexthops[]`: This array contains the IDs of the next hops on the

route. `nexthops[0]` is the ID current node, `nexthops[1]` the next hop, and so on.

In the simulations, the extended range of large nodes is twice the normal range of small nodes. Still, the broadcast messages are transmitted with the normal range only.

When a small node receives a broadcast message `msg` it updates the routing table and re-broadcast the message as shown in algorithm 5.1.

Algorithm 5.1: SRP routing table update on small nodes

```
if msg.counter < my.hops then
    my.hops = msg.counter + 1;
    for i=0 to msg.nexthops.length-1 do
        my.nexthops[i+1] = msg.lasthops[i];
    end
    my.nexthops[0]= my.nodeId;
    newMsg = new BroadcastMessage();
    newMsg.lasthops = my.nexthops;
    newMsg.counter = my.hops;
    send(newMsg);
end
```

The behavior of a large node receiving the broadcast depends on whether the last hop (`lasthop[0]`) is a small node or a large nodes. If the last hop was a small node and the boost-range for large nodes is twice the normal radio range then this node can be omitted in the routing (see algorithm 5.2).

Since the counters transmitted by large nodes are generally smaller than counters transmitted by small nodes, the route is more likely to use a large node. Figure 5.10 displays the route of a message originating in node 14. On the left side, large nodes are treated as small nodes. The route from node 14 does not use any large node as forwarding node. On the right side large nodes are used as described above and use their extended range. This protocol uses the large nodes 1 and 4 as forwarding nodes.

.

Transition routing protocol The second routing protocol is used to route messages from the cluster head to the transition node. The route establishment happens in combination with the creation of LE-Keys as described in section 4.4. The structure of this broadcast message is shown in figure 5.11.

This routing protocol is only implemented on small nodes. The processing of the message is similar to the sink routing protocol: All counters are increased

Field:	`dest`	`type`	`counter`	`lasthops[0...(x − 1)]`
bytes:	2	1	2 bytes	$2 \cdot x$ bytes

Figure 5.9: Structure of a `AbstractSinkRouteFinder` broadcast message

Algorithm 5.2: SPR routing table update on large nodes

```
if msg.counter < my.hops then
    if msg.lasthops[0] is a large node then
        my.hops = msg.counter;
        for i=0 to msg.lasthops.length-1 do
            my.nexthops[i+1] = msg.lasthops[i];
        end
    else
        my.hops = msg.counter - 1;
        for i=0 to msg.lasthops.length do
            my.nexthops[i] = msg.lasthops[i];
        end
    end
    my.nexthops[0]= my.nodeId;
    newMsg = new BroadcastMessage();
    newMsg.lasthops = my.nexthops;
    newMsg.counter = my.hops;
    send(newMsg);
end
```

before the update. The routes are updated for the shortest L large nodes encountered so far. If the routing table has changed, the new table is re-broadcasted.

Both protocols are rather fragile concerning node failures. Backup routes are not stored. If a node fails, the complete sub-tree that has this node as root is cut off from the sink (or the transition node). In a real deployment scenario, these protocols would have to be more sophisticated, but this topic is beyond the scope of this thesis.

5.3 Implementation of HEFS

The classes for the local cooperative report generation are found in the package `org.knappi.wsn.sim.nodecomponents.coop`. The component consist of five subcomponents that are managed and encapsulated by the `LocalCooperationComponent`. Each of the subcomponents represents one state of the generation protocol from section 4.5. The components are enumerated by the order, in which they are activated. Components that are active when the node is in cluster-head mode have odd numbers. Helper node components have even numbers. A cluster manager can be attached to the `LocalCooperationComponent` in order to implement different clustering behaviors.

Clustering algorithm A trivial clustering algorithm has been implemented in the class `AdHocClusterManager`: The first node that attempts to become cluster head is elected. When a sensor event occurs, every node waits a random period of time. Two possibilities can occur then:

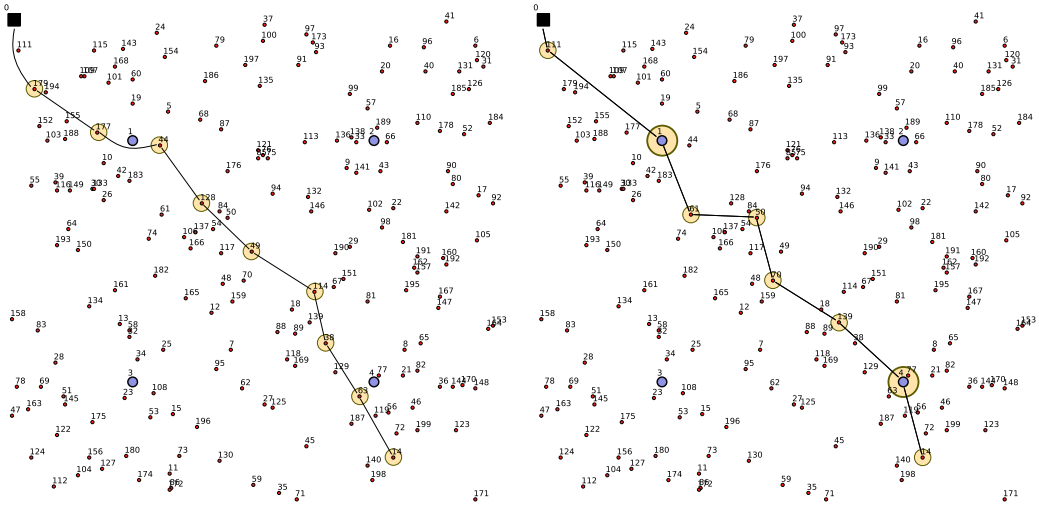


Figure 5.10: Routing path of a message from node 14 to the sink and a router without (left) and with (right) preference of large nodes

Field:	`dest`	`type`	`sender`	`id`$[0\ldots(L-1)]$	`counter`$[0\ldots(L-1)]$
bytes:	2	1	2 bytes	$2 \cdot L$ bytes	$2 \cdot L$ bytes

Figure 5.11: Structure of a `AbstractSinkRouteFinder` broadcast message

1. The waiting period finishes before anything else happens: Then the node switches to cluster-head mode and sends a `COOP1StimulusMessage` to the neighboring nodes.

2. A `COOP1StimulusMessage` message is received before the waiting period is finished. In this case, the node switches to helper-node mode and ignores the end of the waiting period.

The cluster manager maintains the current mode of operation, which can be: *undefined*, *cluster-head*, *helper-node*. The report-generation components can ask the cluster manager to switch the mode and the cluster manager can approve or refuse the request. The `AdHocClusterManager` always approves the request if it is in an undefined state or if it is already in the requested state, but other implementations are possible, too.

The state of the report generation process is kept in the cluster head, while the helper nodes are stateless. This fact is reflected in the cluster manager: The helper-node mode is automatically reset to the undefined mode after a specified waiting period, but the cluster-head mode must be explicitely reset by the generation component.

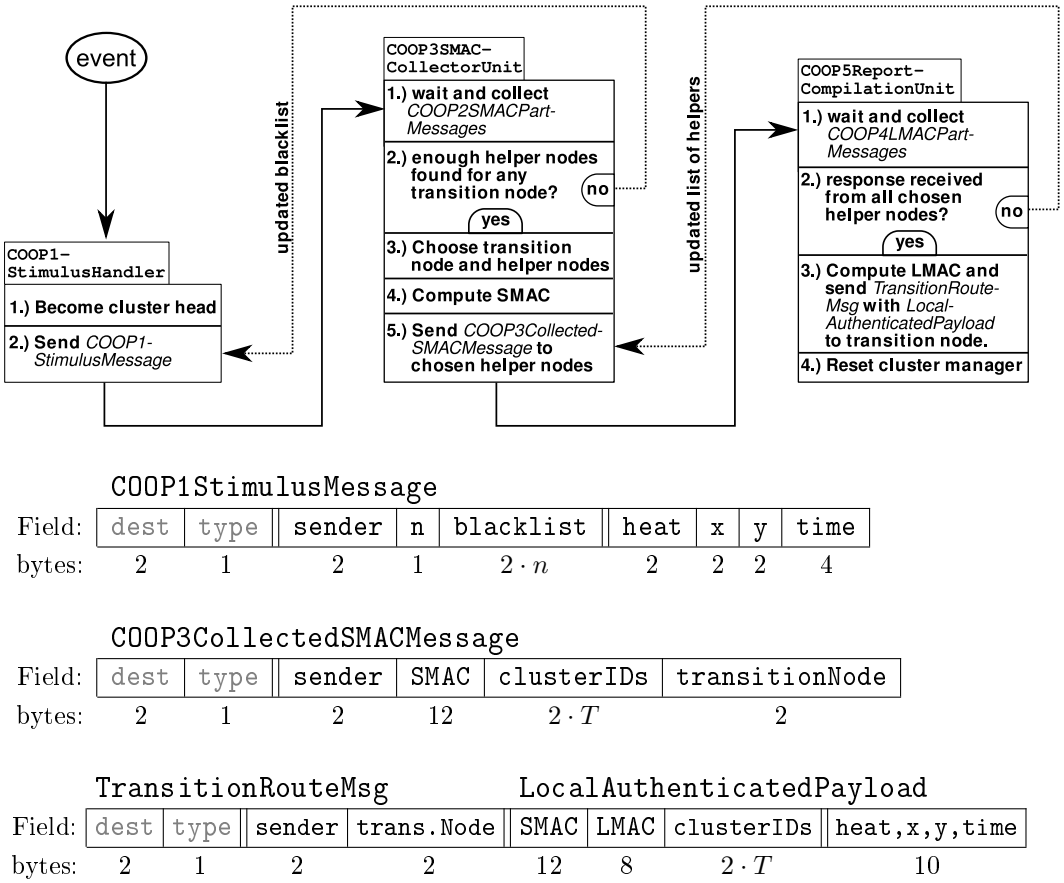


Figure 5.12: State transitions and message structures of the report generation component, when in cluster-head mode

Localization and key creation The key generation is done by the `AbstractKeyStorage` class and its subclasses as described in section 4.4: The deployment area of the network is divided into cells. Locations are specified by an (x, y)-pair, where x and y are 2-byte values. The cell of a location (x, y) is computed as

$$\text{cell}_x(x, y) = \left\lfloor \frac{x}{C} \right\rfloor \tag{5.4}$$

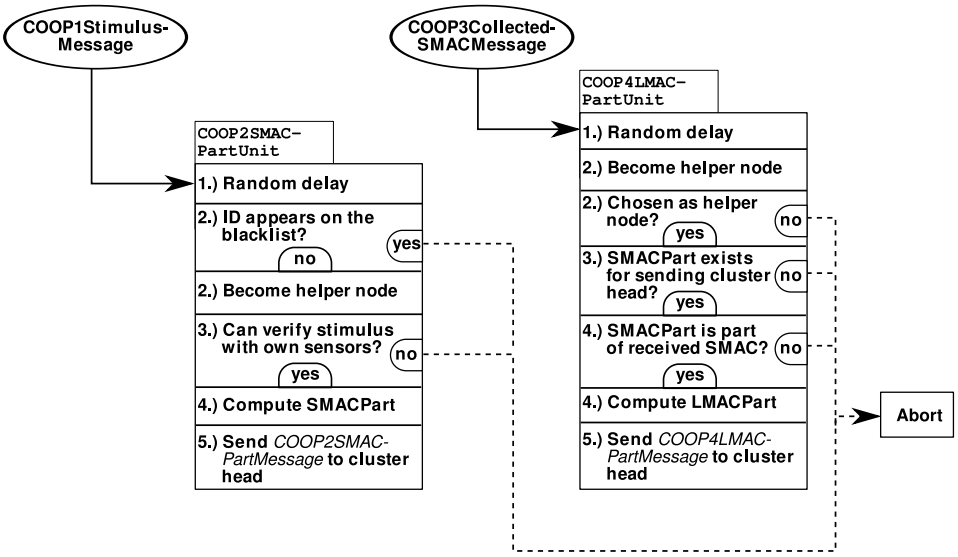
$$\text{cell}_y(x, y) = \left\lfloor \frac{y}{C} \right\rfloor \tag{5.5}$$

where C is the pre-defined cell size. Each small node computes LE-Keys and LBI-Keys for each cell which lies within its verification distance V_d.

Each large nodes computes and stores $K_{\mathbf{A}}^{init}$ as described in chapter 4.4.

Local generation Figure 5.12 gives an overview over the interaction between the cluster-head components and the structure of the sent messages. The optimal sequence of actions of the cluster head is:

1. Switch to cluster-head mode.
2. Send `COOP1StimulusMessage` to neighboring nodes.



`COOP2SMACPartMessage` (helper node)

Field:	dest	type	sender	transitionNodes	smacPart
bytes:	2	1	2	$2 \cdot L$	8

`COOP4LMACPartMessage` (helper node)

Field:	dest	type	sender	lmacPart
bytes:	2	1	2	8

Figure 5.13: State transitions and message structures of the report generation component, when in helper-node mode

3. Wait and collect `COOP2SMACPartMessages` from neighboring nodes.

4. Choose a transition node and helper nodes, compute the SMAC and send `COOP3CollectedSMACMessage` to the helpers (specified in the `clusterIDs` field).

5. Wait and collect `COOP4LMACPartMessages` from helpers.

6. Compute LMAC and send a `TransitionRouteMsg` with a `LocalAuthenticatedPayload` to the transition node.

7. Reset the cluster manager.

In the simulation, this sequence may be disturbed by message collisions: If the cluster head does not receive enough answers in the steps 3 and 5, it cannot proceed with the next action. Instead, it has to resend the previous message and collect the answers again. The `blacklist` field in the `COOP1StimulusMesage` reduces the probability of collisions after resending the message. It contains a list of all node IDs of which an answer has already been received. Blacklisted helpers do not send a reply to the message anymore. Similarly, nodes are blacklisted in step 5 by removing their ID from the `clusterIDs` field.

Figure 5.13 presents an overview over the component interaction, when the node is acting as helper node. In the first implementation of the report-generation component, the helper-node mode had a similar structure as the cluster-head mode: When a `COOP1StimulusMessage` was received, the helper-node mode was activated. After sending the `COOP2SMACPartMessage` the component awaited the `COOP3CollectedSMACMessage` message from the same cluster head. Finally, after sending the `COOP4LMACPartMessage`, the cluster manager was reset to an undefined mode again. This behavior caused a number of problems:

- Every helper node was able to support only one cluster head. When an event was generated on a larger area, the cluster heads began competing for helpers. The worst effect of this flaw was the following: A cluster head **a** at the border of the event could not file a report because not enough helpers could confirm the measurement. Another cluster head **b** was unable to create a report because **a** had sent its broadcast message first, thereby binding some potential helper nodes. In this case, no report was filed at all.

- Helper nodes could not answer to repeated broadcasts because when the broadcast (`COOP1StimulusMessage` or `COOP3CollectedSMACMessage`) had been sent for the second time, the state transition had already taken place.

- Some helper nodes did not reset their cluster managers because they never received the `COOP3CollectedSMACMessage`. The effect was that those nodes were essentially lost for the network. They could not become cluster head by themselves and did not answer to `COOP1StimulusMessages` anymore.

- When a helper node returned to the undefined state before the initial waiting period was over, it became a cluster head and filed a new report for the same event. As a result more reports than necessary were sent.

In order to resolve these problems, the state transitions have been removed. In the current implementation, the cluster manager is switched to helper-node mode when a `COOP2SMACPartMessage` or `COOP4LMACPartMessage` is received and it switches back to *undefined* automatically after a defined waiting time. Both helper-mode components can be active independently of each other, but the `COOP4LMACUnit` only answers to the cluster head, for which SMAC-part has been stored before.

Multiple SMAC-parts can be stored, so that multiple cluster heads can be served at the same time. The waiting period before the reset prevents the helper node from acting as cluster head for the same event and ensures that no helper node gets stuck in its role as helper.

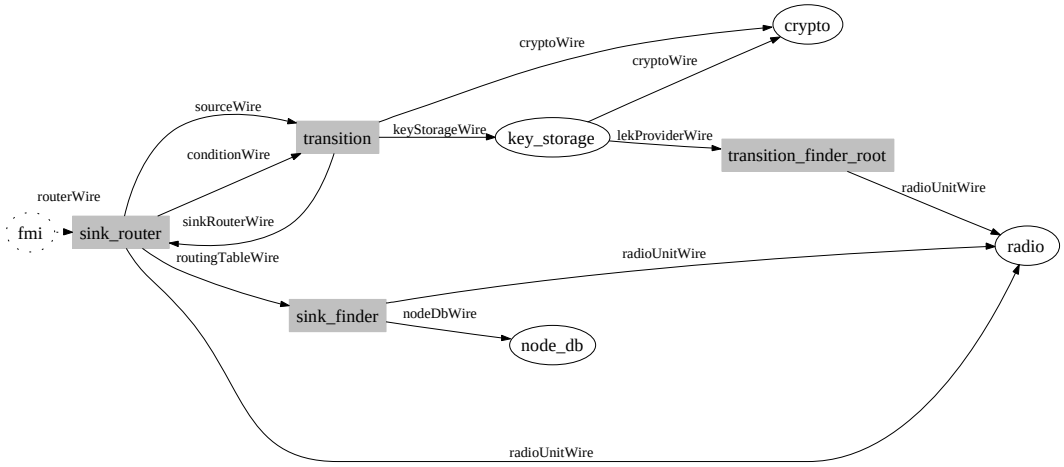


Figure 5.14: Node components and wires of a large node. The `TimerComponent` has been ommitted for the sake of the clarity.

`SinkRouteMsg` `PKSigPayload`

`dest`	`type`	`sender`	`trans.Node`	`SMAC`	`signature`	`clusterIDs`	`heat,x,y,time`
2	1	2	2	12	32	$2 \cdot T$	10

Figure 5.15: Structure of a `SinkRouteMsg` with `PKSigPayload`

Transition and the public-key signature The components and wires of a large node are illustrated in figure 5.14. When a `LocalAuthenticatedPayload` reaches the transition node, it is handled by the `TransitionComponent`. This component asks the key storage component to generate the LE-keys for all attached IDs and the cell of the location specified in the report. After verifying all MACs, it creates its "spublic-key signature" for the report. In section 5.1, we have explained that the implementation of a real public-key algorithm is not necessary in the simulation. Instead, the "public-key signature" is a MAC, that is computed with the transition node's ID as key. The remaining part of the signature is padded with *zeros* up to a length of `PK_SIG_LENGTH = 32` bytes (256 bits). After computing the signature, the transition node sends a `SinkRouteMsg` with a `PKSigPayload` (see figure 5.15).

Note that `SinkRouteMsg` does not define an extra target-field on top of the default header because there is only one possible target destination (i.e. the sink) for this message type.

Verification In small nodes, the class `GenericRoutingComponent` is responsible for forwarding messages in both protocols. Large nodes use the subclass `ConditionalRoutingComponent` for the SRP. This component has an additional wire, the `conditionWire`. Before routing a message, it calls the `verifyCondition(payload)`-method of the `conditionWire`. The message is only

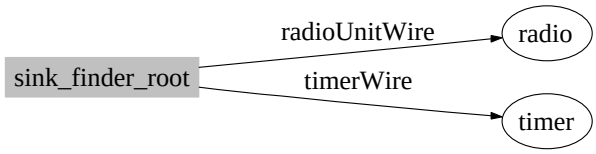


Figure 5.16: Components and wires of the sink.

forwarded if the return value is true. The `conditionWire` is connected to the `TransitionComponent` which performs the verification of the payload's signature.

We have omitted the verification at the sink and the long-term analysis. In the simulation, we assume that an attacker does not forge the public-key signature, even if he compromises a large node. The long-term analysis is evaluated theoretically in chapter 6. That is why the implementation of the sink only consists of three components (see figure 5.16). The sink initiates the broadcast that initializes the sink routing protocol. Other functions have been omitted.

5.4 Comparative protocol

The goal of the simulation is to compare the performance of HEFS to the perfomance of an "ordinary" sensor network. This section describes the implementation of the "ordinary" protocol.

The ordinary sensor network is a homogeneous network. Network layout and the radio topology are loaded from the same files as in the en-route filtering simulation, but the factories for both large and small nodes are of the type `HomogeneousNodeFactory`. This means that only one node type is generated (except from the sink). The components and wiring of this node type are displayed in figure 5.17.

The protocol assumes a cluster based report generation but does not simulate the communication in detail. Instead, when a sensor node measures a critical temperature, it sends a message directly to the sink. The structure of this message is shown in figure 5.18.

The `clusterIDs` field contains the least-significant byte of the ID of the node that created the report and is padded with zeros to simulate the correct length of the message. The `MAC`-field is reserved for a cluster-created MAC but contains only zeros because the current implementation does not perform a sink verification. We also assume, as in all messages that an additional 64-bit MAC is attached by the LEAP-framework. The `sender` field is the ID of the last sender which is needed by LEAP.

If we assume that the cluster uses LBI-keys for the report generation, then the protocol has the same properties as HEFS, except that it cannot filter reports en-route. Therefore, the protocol is an appropriate comparison when measuring the effectiveness of the en-route filtering.

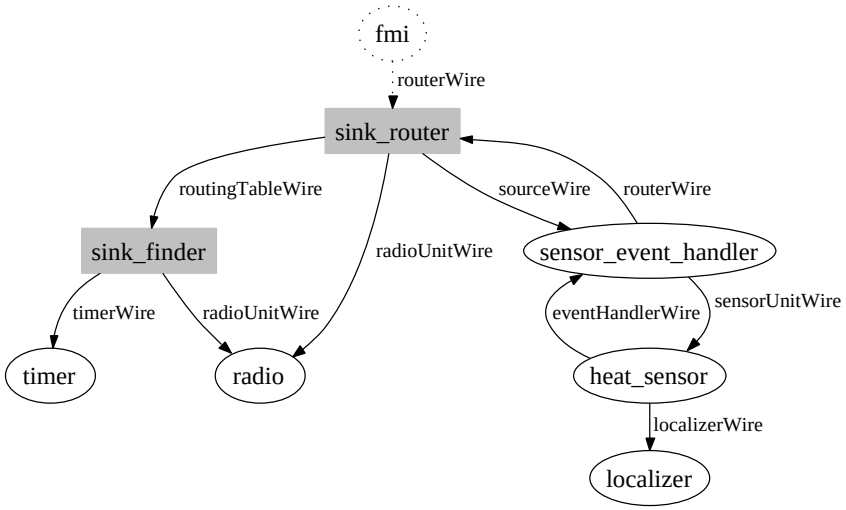


Figure 5.17: Wiring of a node in the comparative scenario.

	SinkRouteMsg			SimpleAuthenticatedPayload		
Field:	dest	type	sender	clusterIDs	MAC	heat,x,y,time
bytes:	2	1	2	$2 \cdot T$	8	10

Figure 5.18: Message structure of the comparative protocol

Boosted configuration The comparison to a homogeneous network does not respect the higher cost of a heterogeneous network. In order to compare HEFS to a protocol that is based on the same hardware, we have implemented the above protocol in a *boosted configuration*. The `HomogeneousNodeFactory` can be configured to create small and large nodes. The only difference between the nodes, is that

boosted configuration

1. the energy consumption of large nodes is ignored in the simulation results, and

2. the large nodes perform algorithm 5.2 to update their routing table.

The result is a protocol that has the message structure shown in figure 5.18, but saves energy by using large nodes to routes message to the sink. This protocol does not perform an en-route verification.

5.5 Implemented configurations

Three batch configurations have been implemented for the statistical evaluation of HEFS. The goal of the first configuration is to determine a reasonable number of large nodes. The second configuration simulates the injection of transition routing protocol (TRP) messages for different values of L. The

third configuration assumes a percentage of injected traffic and measures the energy consumption of the network.

All configurations expect a set of scenario directories as input and measure the traffic caused by injecting messages into

1. a homogeneous network,
2. the comparative scenario in the boosted configuration, and
3. a heterogeneous network that uses HEFS. The injection of SRP messages and TRP messages is simulated separately. TRP messages are sent towards the furthest of the L target nodes in the routing table.

Each configuration writes the simulation results to a file that can be read by GNUplot. The structure of one line is equal for all configurations and shown in figure 5.19. The meaning of the first column is different in each configuration.

1	2	3	4	5	6	7	8	9	10	11
	Hops per message					μJoules per message				
x	min	max	median	avg.	std.Dev.	min	max	median	avg.	std.Dev.

Figure 5.19: Format of the batch configurations' output file

5.5.1 Number of large nodes

The `AbstractPDoSBytesConfiguration` expects that the input contains scenario definitions with different numbers of large nodes. It runs a simulation with the phases displayed in algorithm 5.3 for each scenario.

At the end of each simulation statistics are computed about the number of transmissions and the energy consumption of TRP- and SRP-messages. Since only one message has been injected between two re-initializations of the message counters, the result of one simulation are statistics about the number of hops (the energy consumption) per message. This statistic record is stored along with the number of large nodes defined in the scenario file[2].

In some cases, a simulation error occurs during the simulation. This may be the case, when the sink is not in range of any node in the network. If an error occurs, the statistic record of the simulation is probably corrupted and therefore discarded.

If multiple records have the same number of large nodes, all of them are saved. In the end, the average of these records is computed. The resulting output file has one line for each number of large nodes. The first column of the output x is the number of large nodes. This file is then used to display the graphs in figure 6.2.

[2]A homogeneous network does not contain any large nodes but the scenario files contain this information nevertheless.

Algorithm 5.3: Simulation phases of the `AbstractPDoSBytesConfiguration`

```
run SinkRouteFinderPhase ;                    /* Initialize SRP */
if heterogeneous network then
    /* Initialize TRP and LE- and LBI-keys.                    */
    run TransitionRouteFinderPhase;
end
/* Discard current message counters and initialize new ones.
   */
run NextLogPhase(discard);
for i=1 to 100 do
    /* Inject a false message into a random node.              */
    run PDoSAttackPhase;
    /* Save current message counters and initialize new ones.
       */
    run NextLogPhase(store);
end
```

5.5.2 Number of LE-Keys

The `TNodesConfiguration` is almost identical to the `AbstractPDoSConfiguration`. The difference is, that it expects scenarios to have all the same number of large nodes. It runs all scenarios with values for

$$L \in \{1, \ldots, 30\} \tag{5.6}$$

in order to measure the impact of the L-parameter (number of transition nodes per small node) on pDoS-attacks. We assume that the attacker injects messages towards the large node with the maximal hop count and this distance depends on the value of L. The injection of sink routing messages is also performed as a reference value, even though the L parameter should not have any influence on the sink routing protocol. The output file contains one line for each value of L, which is the value of the first column. The graphs in figure 6.4 are based on the output of this batch configuration.

5.5.3 Ratio of injected messages

The `AbstractTrafficTestConfiguration` also expects scenarios with the same number of large nodes. The sequence of simulation phases for each simulation is described in algorithm 5.4.

We assume that an injected message causes more traffic in the comparative protocol than in HEFS because HEFS drops the message at the first large node. On the other hand, a *valid message* that was created in reaction to a real sensor event causes more traffic in HEFS: It is not filtered and the attached MACs and signatures make every single message bigger than in the comparative scenario.

valid message

Algorithm 5.4: Simulation phases of `AbstractTrafficTestConfiguration`

```
run SinkRouteFinderPhase;
if heterogeneous network then
    run TransitionRouteFinderPhase;
end
run(NextLogPhase(discard);
for n=0; to 50 do
    reset GlobalMessageStatistics;
    for i=1 to 10 do run(HeatEventPhase);
    for i=1; to n do run(PDoSAttackPhase);
    run NextLogPhase(store);
    compute statistics;
    addStats(n/10.0, stats) ;                          /* β = n/10 */
    /* Results are divided by 10.0 before output                  */
end
```

The goal of this configuration is to measure the total traffic caused for different values of

$$\beta = \frac{f}{v} \tag{5.7}$$

where f is the number of injected false reports and v is the number of valid reports. The output file has one line for each value of β, which is written in the first column of the file. The values in columns 2 to 11 refer to the total number of message transmissions (or the energy consumption) during one heat event and β injected messages.

Chapter 6

Evaluation

This chapter evaluates the *hybrid en-route filtering scheme* HEFS. Section 6.1 provides a security analysis of HEFS. Section 6.2 analyzes the dependencies between the parameters of HEFS and the security and efficiency goals. Section 6.3 evaluates the energy consumption of HEFS and the resilience against pDoS-attacks. Section 6.4 evaluates the security long-term-analysis tool and the energy consumption of injected degenerated reports. Section 6.5 analyzes the memory overhead caused by HEFS and the underlying protocols, and section 6.6 examines the impact of failing large nodes on the protocol.

6.1 Security analysis

HEFS was developed to reduce the impact of *false data injection attacks*, *data alteration attacks*, and *path-based denial of service attacks*. A more detailed distinction of the attacks is necessary in order to evaluate the effect of these attacks because **(a)** a pDoS-attack can be started in different ways and **(b)** there must be a distinction between comprising large nodes and small nodes. The section analyzes the opportunities an adversary has to attack the network.

Path-based DoS attacks Only large nodes verify the signature of SRP- and TRP-messages. Thus an injected or altered message is forwarded until it reaches the first large node on the route. A measure of the efficiency of HEFS is the number of hops travelled by such a message and the total energy consumed by all radio units in the network.

The maximal hop count of an injected message depends on the number of large nodes and is evaluated in section 6.2.4. An analysis of the overall energy consumption is presented in section 6.3.

The impact of pDoS-attacks on the TRP is also influenced by the parameter L. The TRP performs routing to the L nearest large nodes. An attacker, who injects TRP messages, causes the largest damage by injecting a message towards the furthest of these nodes. The impact of the L parameter is analyzed in section 6.2.5.

Injection of degenerated reports If an attacker has compromised a large node and less than T small nodes, he can generate reports with a valid signature but he cannot create a correct SMAC. Such a degenerated report remains undetected until it reaches the sink and thus causes more traffic than the previously described injections. HEFS reacts by isolating compromised nodes from the network. The impact of such an attack is analyzed in section 6.4.

Exploit weaknesses in the long term analysis The statistical analysis tool of the sink computes a score for each node that is involved in the creation of a degenerated report. When the score exceeds a pre-defined threshold, the node is removed from the network. The rules of this tool are assumed to be available to the attacker. After compromising a number of nodes, the attacker can actively try to mislead the statistical analysis, thus removing innocent nodes from the network. The effort and impact of such frameup attacks is evaluated in section 6.4.

Removal of large nodes The heterogeneous routing structure of HEFS requires that small nodes send their data to a nearby large node. After the initialization phase, each small node maintains shared keys with L large nodes. If all L potential transition nodes of a small node $\mathbf{a}$ fail or are isolated by the sink, node $\mathbf{a}$ is isolated as well because a redeployment of nodes is not intended in HEFS. The more large nodes lost, the bigger the number of small nodes unable to participate in the creation process. In section 6.6 we analyze the impact of the simple case that exactly L large nodes are failing.

Compromisation of L small nodes in the same region An attacker, who compromises L small nodes in the same location (key-accuracy, see page 28), can inject false reports under a number of conditions:

1. The location of the report must match the location of the keys that are stored in the compromised nodes.

2. If the attacker has not compromised any large nodes, the injected message must be sent to a transition node for which the compromised nodes store LE-keys (i.e. a large node in the vicinity of the compromised nodes).

3. If the attacker has compromised a large node and enough small nodes in the same location, he can use the LBI-keys of the small nodes and the signature of the large node to inject false reports anywhere into the network. Chapter 7 proposes a method to restrict the point of injection.

Within these limitations, the attacker can perform pDoS-attacks as well as false data injection attacks. A purely software-based solution cannot prevent this kind of attack. The attacker can achieve the same result by leaving the node-application intact and injecting false stimuli into the sensor unit of the compromised nodes.

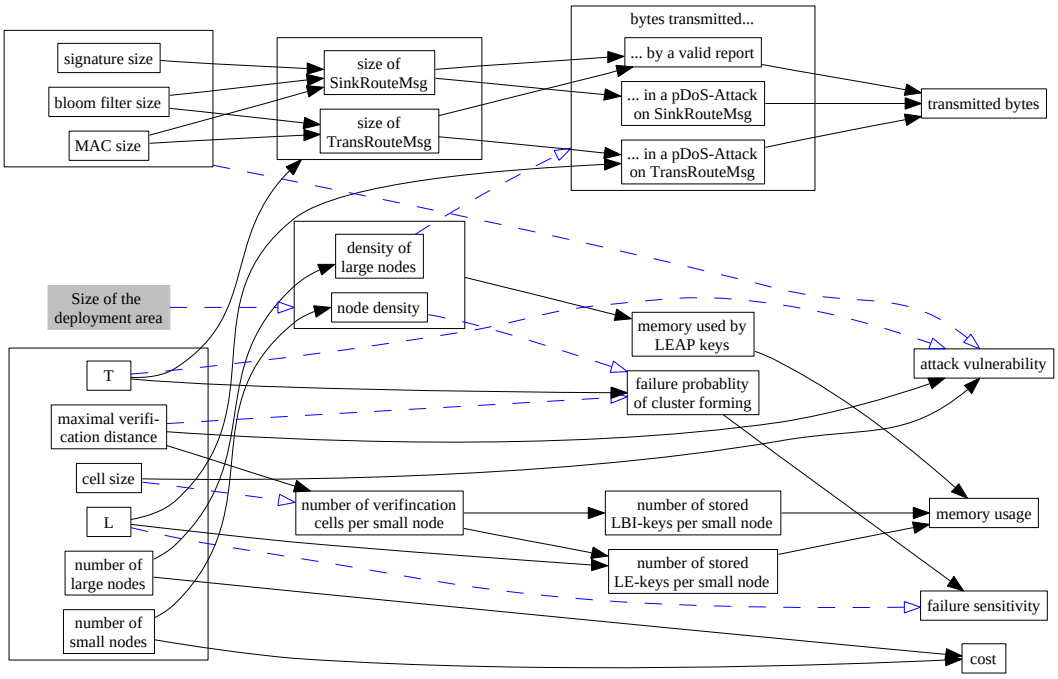


Figure 6.1: Dependencies between parameters and efficiency goals. The left column shows parameters and the gray box is assumed to be constant. The right column contains minimization goals. Each box can be seen as a value such as "large" or "small". Dotted blue lines indicate that increasing the source value causes a decrease in the target value. Solid black lines indicate that increasing the source value causes an increase in the target value

Denial-of-service attacks En-route filtering generally has the drawback that the sink is not notified about data alterations. Instead, a data alteration attack has the same effect as a black hole attack: The altered message disappears on the way to the sink. Attacks of this type are not in the scope of this thesis.

6.2 Parameter adjustment

The efficiency of HEFS and its resource overhead depend on a variety of adjustable parameters. Figure 6.1 provides an overview of the parameters and the effect of parameter modification. This section describes the illustrated dependencies in detail.

Some of the dependencies have been evaluated empirically in a simulation. In these simulations, the sensor nodes are deployed **(a)** in an area with the shape of a square and **(b)** in an area with the shape of a prolate rectangle. The exact parameters are shown in table 6.1.

In order to provide comparability with other en-route filtering schemes, the size of the unauthenticated payload data (temperature, x, y, time) has been expanded to 24 bytes. In the *statistical en-route filtering scheme* (SEF) [36] the

	prolate shape	quadratic shape
area size	4000×200	2000×2000
radio range	50 (boost 100)	50 (boost 100)
verif.dist.	60	60
total nodes	2000	6000
large nodes	50 and 100	100 and 200
T	5	5
L	4	4

Table 6.1: Parameters used in the simulations

same size is used to evaluate the energy consumption. The size of a transition route message is then 61 bytes (see figure 5.12) and the size of a sink route message is 85 bytes (see figure 5.15)

6.2.1 Cluster size T

The parameter T defines how many nodes have to endorse a valid report. Modifying the parameter has positive and negative effects:

1. A larger T makes the network more robust because it forces the attacker to compromise more nodes before he can inject false messages without being detected.

2. A larger T also bears the danger that the cluster head does not find enough helper nodes to create a report.

3. A larger T increases the message size because the ID of each node has to be transmitted in the message.

The significance of this parameter cannot be evaluated in a generic simulation and the increase of the message size is only relevant for large values of T. With a total message size of 61 or 85 bytes, the increase of T by one causes less then 5 percent additional traffic. For the simulation we arbitrarily set $T = 5$.

6.2.2 Cryptographic parameters

The size of SMAC, LMAC and the public-key signature is a modifiable parameter of HEFS. A larger key size increases the security against brute-force attacks but it also increases the message size and thus the network traffic. In all our evaluations we used a MAC size of 8 bytes and a bloom-filter size (SMAC) of 12 bytes. We did not modify the parameters because neither cryptoanalytical nor brute-force attacks are in the scope of this thesis.

As shown in section 3.4.1 there are multiple ways of distributing public keys. The ID-based approach is the most promising for an en-route filtering scheme like HEFS because no public-keys have to be transmitted. The public-key is the node ID of the transition node. This ID is already part of each sink route message. This greatly reduces the traffic induced on small nodes.

In our simulations we used a signature size of 32 bytes (256 bits). Elliptic curve signatures of this size provide a security equivalent to an 8 byte (64 bit) MAC (see section 3.1).

6.2.3 Number of small nodes

A modification of the number of deployed small nodes has several implications. On the one hand, the hardware cost of the network increases linearly with the number of small nodes. On the other hand, a higher node density implies that each node has more direct neighbors, which has two consequences:

1. The memory consumption increases because LEAP needs to store more pair-wise keys and cluster keys.

2. A low node density bears the risk that the cluster head cannot find enough helper nodes to create a report. This probability decreases with an increasing node density.

The number of LEAP keys can be limited manually in order to save memory but that also increases the risk of not being able to find enough helper nodes.

6.2.4 Number of large nodes

The number of large nodes also influences the hardware costs and the memory consumption caused by LEAP-keys because large nodes communicate with small nodes.

On the other hand, the result of a high density of large node is a short distance from any small node to the nearest large node. This reduces the impact of pDoS-attacks on both routing protocols. Figure 6.2 shows the travelled hops of an injected message in simulations with 0 to 200 large nodes (see section 5.5). Four different network configurations have been simulated:

1. A homogeneous network, where all large nodes are acting as homogeneous nodes (see section 5.4).

2. A heterogeneous network that does not filter injected messages. In this *boosted configuration*, the energy consumption of large nodes is ignored in the measurement. The SRP favors large nodes en-route to the sink and large nodes use their extended radio range when transmitting messages.

boosted configuration

3. A heterogeneous network that uses HEFS to filter injected messages and an attacker who injects SRP messages.

4. A heterogeneous network that uses HEFS to filter injected messages and an attacker who injects TRP messages.

The figure shows that the average hop count of an injected message in a homogeneous network is $h_0 \approx 47$ in a prolate scenario and $h_0 \approx 30$ in a quadratic

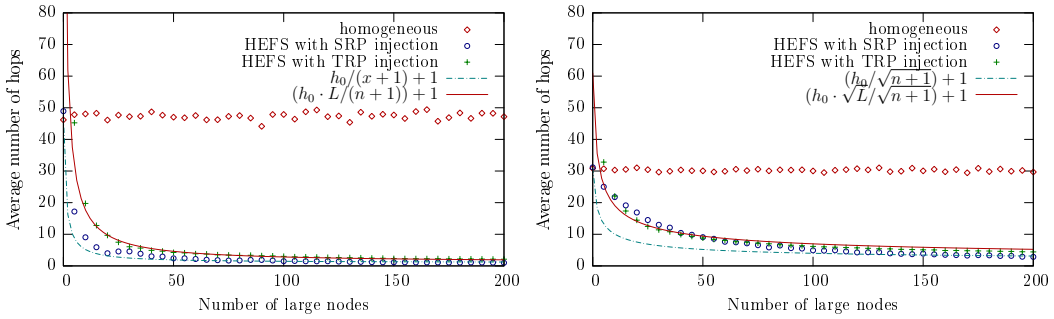


Figure 6.2: Average hops of one injected false message in the prolate scenario (left) and in the quadratic scenario (right)

scenario without large nodes. When n large nodes are present, the hop count of an injected sink route message can be approximated by

$$h_{SRP}(n) \approx \frac{h_0}{c} + 1 \approx \frac{h_0}{\frac{n+1}{r}} + 1 \tag{6.1}$$

if all large nodes are arranged in c columns and r rows and $c > r$. In the prolate scenario, large nodes are arranged in one, two or three lines, depending on n. In the quadratic scenario, equation 6.1 changes into

$$h_{SRP}(n) \approx \frac{h_0}{\sqrt{n+1}} + 1 \tag{6.2}$$

because then $r \approx \sqrt{n}$.

The difference between the measured hops and the approximation formula is caused by message injections that are not routed via the nearest large node in direction to the sink. This happens because the SRP is primarily targetted on minimizing the transmissions by small nodes. The route via a large node is only used if it is no more than two hops longer than the shortest alternative route around the large node.

The behavior of TRP messages is similar but additionally depends on the L-parameter. This relation is discussed in the next section.

Figure 6.3 shows the total energy consumption per message injection. This figure respects the size of each transmitted message and the energy consumed by sending and receiving one byte (see table 1.1). The energy consumption of an injected message in HEFS must be significantly lower than in the homogeneous and in the boosted scenario because the difference must compensate the MAC and signature overhead of valid reports in HEFS.

Note that the number of deployed small nodes does not affect the hop count of an injected message. The only relevant factors are the size and shape of the deployment area, the radio range of the nodes, and the number of large nodes.

6.2.5 Number of LE-keys L

The L parameter defines how many LE-Keys are stored by each small node. It also defines the number of potential transition nodes for which a node can

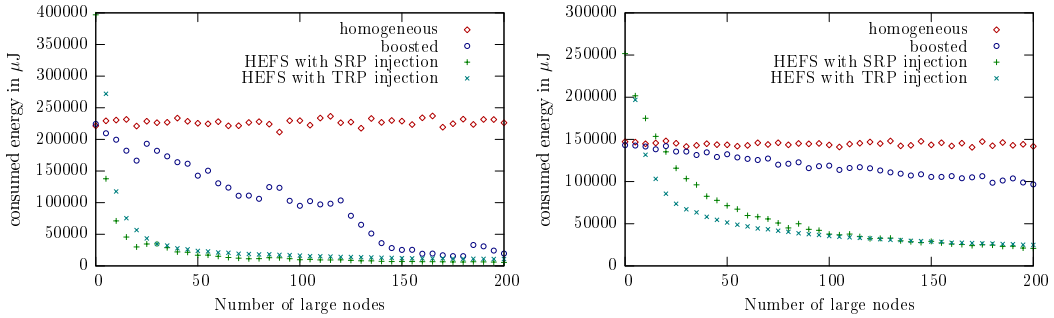


Figure 6.3: Average energy consumption by one injected false message for different numbers of large nodes in the prolate scenario (left) and in the quadratic scenario (right)

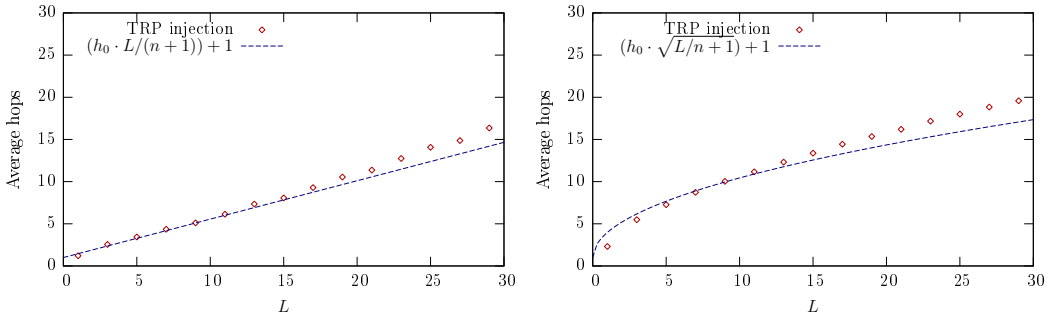


Figure 6.4: Average hops of one injected TRP message for different values of L in the prolate scenario (left) and in the quadratic scenario (right)

endorse reports and the number of targets in the TRP.

Increasing the parameter has a positive effect on the network stability: Small nodes have multiple potential transition nodes if some large nodes should fail. The negative effects of a large L are the memory consumption (8 bytes per LE-key) and the effect on the TRP: a compromised node causes the biggest damage by sending a message to the furthest target nodes. The distance to this node increases with an increasing L. The relation between L and the hop count of an injected TRP message is displayed in figure 6.4. For low L, the average number of hops of an injected message can be approximated by

$$h_{TRP}(n, L) \approx h_0 \cdot \frac{L}{n+1} + 1 \tag{6.3}$$

in the prolate scenario and by

$$h_{TRP}(n, L) \approx h_0 \cdot \sqrt{\frac{L}{n+1}} + 1 \tag{6.4}$$

in the quadratic scenario and it can be expected that for a cubic scenario the approximation function has the form

$$h_{TRP}(n, L) \approx h_0 \cdot \sqrt[3]{\frac{L}{n+1}} + 1 \tag{6.5}$$

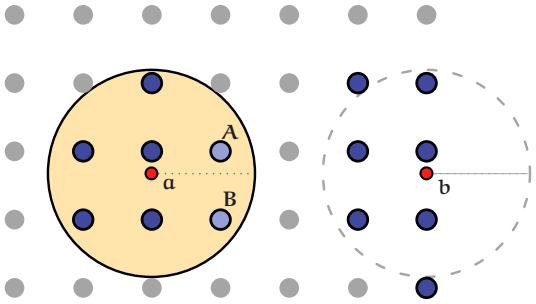


Figure 6.5: Simplified illustration of $\mathcal{TN}_{\mathbf{a}}$ and $\mathcal{TN}_{\mathbf{b}}$ for $L = 7$

Thus the impact of an increasing L on the hop-count is lower if the dimension of the scenario model is higher.

For large L, we can observe a deviation of the approximation function from the measurements. This deviation is caused by the influence of small nodes at the border of the deployment area which is illustrated in figure 6.5:

The figure shows a simplified view on a heterogeneous sensor network. The large nodes are represented by large circles. The small nodes, with the exception of $\mathbf{a}$ and $\mathbf{b}$, are hidden in this figure. The potential transition nodes $\mathcal{TN}_{\mathbf{a}}$ and $\mathcal{TN}_{\mathbf{b}}$ are highlighted. It can been seen that the maximal distance of a small node $\mathbf{a}$ to a large node $\mathbf{X} \in \mathcal{TN}_{\mathbf{a}}$ is larger than the equivalent distance for $\mathbf{b}$. The reason is that $\mathbf{b}$ lies within a zone close to the border of the deployment area. Transition nodes equivalent to $\mathbf{A}$ and $\mathbf{B}$ cannot be found in the vicinity of $\mathbf{b}$.

This zone grows with an increasing L. Since the simulation injects messages into random nodes, a higher number of simulated message injections start in a node within this zone. The approximation function assumes a small node in the middle of the deployment area. This explains the deviation in figure 6.4.

Consequently, an attacker causes more damage by injecting transition route messages at the border of the deployment area.

6.2.6 Cell size and verification distance

The location in *key-accuracy* is less exact than the location in *application-accuracy*. Depending on the scenario, it might be possible to specify how accurate a key-location is. In the simulation, the location of a key is as accurate as the size of the cells that are used for the transformation (see page 50).

An attacker who has compromised T or more small nodes can inject false reports from the area for which the keys are valid. A small cell size (or a high key-accuray) means that the validity area is small.

A related parameter is the *maximal verification distance*. Generally, a helper node must be able to verify measurements from neighboring nodes. If such a verification is possible over a large range, it can be limited by reducing the maximal verification distance. The parameter specifies the maximal distance between a helper node's location and the event that is to be endorsed. If the distance is larger than the cell size, each node has to store keys for multiple

locations. This means, that

- the memory consumption of HEFS increases.
- an attacker gains valid keys for multiple locations by compromising a single node.

On the other hand, if the value is too small, the network is unable to form clusters for the report generation.

In reality the parameters *cell size* and *verification distance* do not need to be simple numeric values but can be handled flexibly according to the deployment scenario. For example, we can distinguish between different security levels: High-security areas might be protected against intrusion while low-security areas are not. In this case sensor nodes in low-security areas should not store localized keys for a high-security area, even if they are able to verify measurements in this region.

6.3 Energy consumption of HEFS

We do not use a battery model to measure the energy consumption of HEFS, but instead examine the most energy-consuming actions of a sensor node:

- radio transmission and reception,
- cryptographic operations.

We only evaluate the energy consumption of small nodes because we assume that large nodes have an unlimited supply of energy.

6.3.1 Radio transmission and reception

The energy consumption caused by radio transmission and reception has been measured with the `AbstractTrafficTestConfiguration` (see section 5.5). Figure 6.6 displays the results of this simulation: The x-axis shows the normalized amount of injected network traffic β. The y-axis displays the average energy consumption caused by the creation of one valid report and the injection of β false messages. The graphs show the energy consumption of a homogeneous network, the boosted configuration, HEFS with the injection of TRP messages and HEFS with the injection of SRP messages.

We can see that the energy consumption of HEFS for $\beta = 0$ is higher than in the unfiltered scenarios but increases more slowly when β increases.

The high value for $\beta = 0$ can be explained by the larger message sizes, the cluster-based report generation, and the fact that every message is first sent to a nearby large node that does not necessarily lie near the route to the sink. In combination with an equal hop count for valid reports, this results in a higher total energy consumption. The slower increase for $\beta > 0$ can be explained by the fact that an injected message travels fewer hops and thus consumes less energy in HEFS. This can also be seen in figure 6.3.

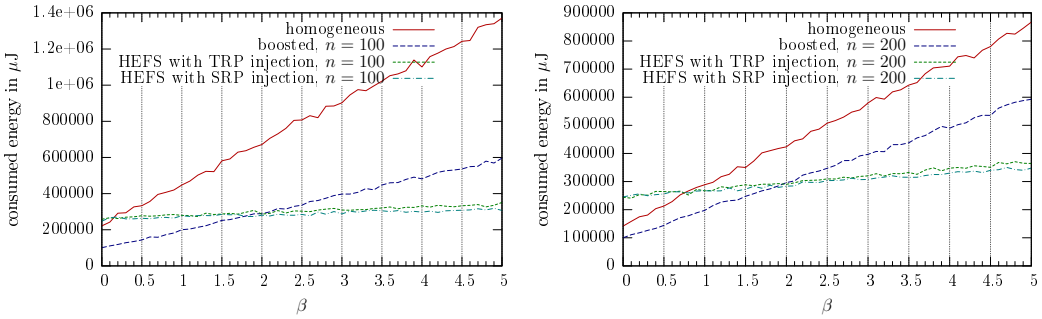


Figure 6.6: Average energy consumption of one valid report and β injected messages in the prolate scenario (left) and in the quadratic scenario (right)

Figure 6.6 also shows that the boosted configuration consumes a signficantly smaller amount of energy than the homogeneous network. There are two reasons for this observation:

- The extended radio range of large nodes reduces the travelled hops of a message to the sink.
- The energy consumed by large nodes is ignored in the measurements.

Another consequence is that the energy consumption of HEFS in the prolate scenario with 100 large nodes for $\beta = 0$ is only marginally larger than the energy consumption of the homogeneous scenario. These observations indicate that HEFS should not only be compared to a homogeneous network but also to the boosted configuration. For this comparison we define the values β'_h and β'_b:

Definition: 6.1 *In comparison with an unfiltered scenario, let β' be the amount of injected messages so that HEFS consumes less energy than the comparative scheme if and only if $\beta > \beta'$. Let β'_h be the value of β' in comparison with the homogeneous network and β'_b be the value of β' in comparison with the boosted configuration.*

β'_h
β'_b

Table 6.2 shows the values β'_b and β'_h that result from different simulation scenarios. Appendix B shows the graphs that lead to these values. Table 6.2 shows that increasing the number of large nodes only reduces β'_b down to a certain limit as long as the size of the deployment area remains equal. At some point, β'_b even increases with the number of large nodes. The reason is, that the efficiency of the boosted configuration also increases with n. If the number of large nodes is so high that a route from any large node to the sink uses only other large nodes then the boosted scenario performs better than HEFS for the following reasons:

- A message, which has reached a large node, does not consume any energy anymore thus the filtering becomes obsolete.

area size (nodes)	β'_b (β'_h)				h_0
	$n = 50$	$n = 100$	$n = 150$	$n = 200$	
2000×2000 (6000)	3.1 (2.3)	2.2 (1.75)	2.0 (1.0)	2.0 (0.75)	30
2000×200 (1000)	**4.0 (0.8)**	$\gg 5$ (0.0)	$\gg 5$ (<0)	$\gg 5$ (<0)	23
4000×200 (2000)	1.6 (0.7)	**2.0 (0.2)**	$\gg 5$ (<0)	$\gg 5$ (<0)	46
6000×200 (3000)	1.3 (0.7)	1.5 (0.1)	**1.5 (0.0)**	2.5 (<0)	71
8000×200 (4000)	1.1 (0.7)	1.1 (0.3)	1.2 (0.0)	**1.4 (<0)**	88

n = number of large nodes

Table 6.2: β'_b and h_0 for different scenario configurations

- The messages sent to the large node in the boosted scenario are smaller than in HEFS.
- The simulation of the boosted scenario does not involve the cluster based report creation which causes an additional transmission overhead.

The bold entries of table 6.2 show that HEFS performs better if the area is larger but the density of large nodes does not change. In scenarios where the average homogeneous hop count h_0 is larger, the hop count of an injected message relative to h_0 is smaller.

SEF [36] is evaluated with $h_0 = 100$. The resulting value β' is approximately 1. It is difficult to compare HEFS to SEF because of the different network structure. Compared to a homogeneous network, HEFS performs better than SEF, but we have to consider that higher hardware costs are involved. We can assume that HEFS achieves $\beta'_b \approx 1$, as well when $h_0 = 100$: Table 6.2 shows a value of $\beta'_b \approx 1.1$ with $h_0 \approx 88$ (in the scenario 8000×200 (4000) with 100 large nodes).

6.3.2 Cryptographic operations

In other schemes [36, 35, 37], every forwarding sensor node is involved in the verification of MACs or signatures. HEFS uses cryptographic operations only during report generation and on large nodes. As a result, small nodes do not perform any cryptographic operations in reaction to an injected false report.

The cluster based generation of a report includes the following cryptographic actions on each helper node (see figure 5.13:

1. Creation of the SMAC-part (step 2). This computation is not only done by the T helper nodes but also by other nodes that have received the `COOP1StimulusMessage` from the cluster head and were able to verify the report with their sensors.
2. Verification of the SMAC (step 4). This means one computation of all bloom-filter hash-functions.
3. Creation of the LMAC-part (step 4).

The cluster head additionally performs the creation of the SMAC out of the SMAC-parts. This involves one computation of all bloom-filter hash-functions. Given h bloom-filter hash-functions and k nodes that have received the `COOP1-StimulusMessage`, the number of symmetric cryptographic operations caused by one report generation is

$$z = k + T \cdot (h + 2) \text{ MAC computations} \tag{6.6}$$

Assuming that $k = 10$, $T = 5$ and $h = 16$, then z resolves to $z = 10 + 5 \cdot (16 + 1) = 95$. According to [17], one MAC computation of a 29-byte packet consumes 49,92 μJ if the RC5 algorithm is used in CBC-MAC mode. This leads to the following estimate for the energy consumed by cryptographic operations:

$$W_{\text{crypt}} \approx 49,92 \cdot z \frac{\mu\text{J}}{\text{report}} = 4742,4 \frac{\mu\text{J}}{\text{report}} \tag{6.7}$$

Note that this is estimate is probably too high because it can be expected that each bloom-filter hash-function consumes less energy than the computation of a MAC.

Nevertheless, the estimate shows that the energy consumed by cryptographic operations is insignificant compared to the energy consumption of the radio unit, which lies in the order of magnitude of $10^6 \mu$J.

6.4 Long-Term-Analysis

The attacks considered in the previous sections of this chapter were launched from small nodes only. In this section, we assume that an attacker has compromised one large node **A** and up to T small nodes in order to inject messages. The knowledge of $\kappa_{\mathbf{a}}^{sec}$ allows the attacker to create a degenerated report that is not filtered en-route but only detected by the sink. Such an action can have two goals:

1. It can be a path-based denial of service attack. In this case the sink excludes attacking nodes from the network after a number of injected messages, so that the impact is limited to a constant amount of traffic. We estimate the energy consumption of such an attack from **(1)** the average number of hops in a boosted scenario, **(2)** the size of a SRP message (85 bytes), and **(3)** the number of injections that are possible before the sink excludes an attacking node.

2. It can be a frameup attack: The attacker compromises a number of nodes and makes sure that multiple degenerated reports are created in a way such that the score of one innocent node **A** is increased in each report. When the score of **A** exceeds the $score^{max}$, **A** will be isolated from the network.

Table 6.3 shows the hop count of a boosted configuration in different simulation scenarios. Only messages transmitted by small nodes were counted as *hops* in this table.

area size (nodes)	n	avg.hops	avg. neighbors
4000×200 (2000)	50	30.4	18
	100	20.3	18
2000×2000 (6000)	100	24.8	12
	200	20.4	12

Table 6.3: Average SRP hop-count and number of neighbors

Field:	dest	type	TN_R	$\mathcal{G}_R[0\ldots(T-1)]$	μTESLA MAC
bytes:	2	1	2	$2 \cdot T$	8

Figure 6.7: Assumed structure of the revocation broadcast for LE-keys

In the following evaluation, we consider the following energy-consuming actions, exemplarily computed for the prolate scenario with 100 large nodes:

- the actual injected message: Assuming that the attacker can inject k messages, before the compromised node is isolated, the energy consumption of these messages is

$$W_{msg} = k \cdot 20.3 \cdot 85 \text{ bytes } \cdot (59.20 + 28.60) \frac{\mu\text{J}}{\text{byte}} = k \cdot 140480.00 \mu\text{J} \quad (6.8)$$

  counting the 20.3 hops per injection, 85 bytes per message, 59.20 μJ/byte for each transmission and 28.60 μJ/byte for each reception.

- the revocation broadcast of the LE-keys used in a degenerated report (see section 4.7). This message is sent after each degenerated report. We assume that this message is transmitted once by each node and that each transmission is received by all its neighbors. The message is structured as shown in figure 6.7. The size for $T = 5$ is 23 bytes. With 18 neighbors per node, 2000 nodes and 1000 large nodes (see table 6.3, the result is a total energy consumption of the small nodes of

$$W_{lek} = 23 \cdot ((2000 - 100) \cdot (59.20 + 18 \cdot 28.60))\,\mu\text{J} = 25083800.00 \mu\text{J} \quad (6.9)$$

  μTESLA also performs a commitement broadcast, containing the next element of the hash-chain:

$$W_{com} = 11 \cdot ((2000 - 100) \cdot (59.20 + 18 \cdot 28.60))\,\mu\text{J} = 11996600.00 \mu\text{J} \quad (6.10)$$

- the revocation of the compromised node. We assume that this information is attached to the last LE-key revocation, increasing its message size by 2 bytes:

$$W_{fin} = 25 \cdot ((2000 - 100) \cdot (59.20 + 18 \cdot 28.60))\,\mu\text{J} = 25083800.00 \mu\text{J} \quad (6.11)$$

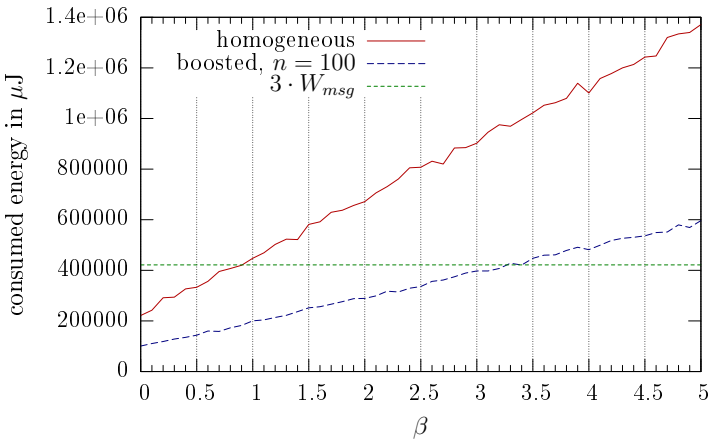


Figure 6.8: W_{msg} and energy consumption of injected traffic in the prolate scenario

The energy consumption caused by one compromised large node is then

$$W = k \cdot W_{msg} + \underbrace{(k-1) \cdot (W_{lek} + W_{com})}_{\text{LE-key revocation}} + \underbrace{W_{fin} + W_{com}}_{\text{last LE-key and node revocation}} \tag{6.12}$$

The energy consumption of the injected message W_{msg} is acceptable given that the compromised node is removed after detection: Figure 6.8 shows the energy consumption of three injected messages compared to the energy consumption of β injected messages and one authentic report in the homogeneous and boosted configuration. Three injected degenerated reports correspond to $\beta = 1$ in the homogeneous network and $\beta = 3.3$ in the boosted configuration.

The energy consumption of the revocation broadcast is much higher because the message is re-broadcasted by every node in the network. Chapter 7 suggests a possible solution to this problem.

The number of potentially injected reports k depends on the applied scoring system.

Static scoring The static scoring system computes the score of a small node $\mathbf{a}$ as the number of evidence records $\mathcal{E}_R$ for which $\mathbf{a} \in \mathcal{G}_R$. The score of a large node $\mathbf{A}$ is the number of evidence records for which $\mathbf{A} = TN_R$. This scoring function leads to

$$k = \lfloor score^{max} \rfloor + 1 \tag{6.13}$$

If we assume $score^{max} = 2$, this means that the attacker is able to send three degenerated reports before detection. Thus a low $score^{max}$ reduces the energy consumptions but also makes the system vulnerable to frameup attacks:

If an attacker tries to isolate a large node $\mathbf{A}$ from the network, he must repeat the following actions k times ($k = \lfloor score^{max} \rfloor + 1$):

1. compromise one small node and

g			1	2	3	4	5
$score^{+}_{\mathcal{G}_R}(g)$	$=$	$\frac{1}{x+g}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$
$score^{+}_{\overline{\mathcal{G}_R}}(g)$	$=$	$\frac{1}{A+T-g}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$
$score^{+}_{TN_R}(g)$	$=$	$\frac{1}{A+\min(e,T-g)}$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$

$A = 2 \qquad e = 3$

Table 6.4: Score increments for a report with g false SMAC-components

	attack strategies				
compromised large nodes	1				
compromised small nodes	0	1	2	3	4
injections before detection	3	4	5	6	6

Table 6.5: Maximal injections of degenerated reports

2. wait for the opportunity to falsify the SMAC of a report with $\mathbf{A}$ as transition node.

$\mathbf{A}$ will then be isolated by the sink because its score exceeds $score^{max}$. Every compromised small node can increase the score of $\mathbf{A}$ only once because the LE-keys shared by TN_R and $\mathcal{G}_R$ are invalidated when the sink receives the degenerated report. Nevertheless, every compromised small node can degenerate k reports before being removed from the network. Thus with k compromised small nodes, the attacker can ensure the removal of k large nodes from the network. For $score^{max} = 2$, this means that the attacker can remove three large nodes by compromising three small nodes.

An attacker might also try to isolate small nodes from the network by sending degenerated reports from a compromised large node. In this case, he needs to compromise k large nodes to frame T small nodes. Before the compromised large nodes are detected, he can then remove $T \cdot k$ small nodes. For $score^{max} = 2$ and $T = 5$, the attacker can remove 15 small nodes by compromising three large nodes.

Dynamic scoring For the dynamic scoring scheme, we choose the parameters $A = 2$ and $e = 3$. The resulting scores for $\mathcal{G}_R$, $\overline{\mathcal{G}_R}$ and TN_R are shown in table 6.4. We furthermore assume that $score^{max} = 1$.

With this configuration, an attacker can send $k = 3$ messages, after compromising only one large node. If he additionally compromises $x < T$ small nodes, k increases because he can then create messages with x correct SMAC-components and $T - x$ false ones. Table 6.5 shows the number of possible message injections k after compromising one large node and x small nodes before the compromised large node is detected.

The effort to perform a sucessful frameup attack is significantly larger when the dynamic scoring scheme is applied. First, we assume that an attacker

	attack strategies				
compromised large nodes	1				
compromised small nodes per report	1	2	3	4	5
necessary reports	7	6	5	4	3
total compromised small nodes	7	12	15	16	15
"reusability" (isolated large nodes)	5	6	7	8	9

Table 6.6: Necessary node compromise to frame large nodes

	attack strategies				
compromised small nodes	0	1	2	3	4
necessary reports	9	8	7	6	5
total compromised large nodes	9	8	7	6	5
innocent small nodes	5	4	3	2	1
"reusability"	3	4	5	6	6
total isolated (innocent) small nodes	15	16	15	12	6

Table 6.7: Necessary node compromises to frame small nodes

compromises $x < T$ small nodes in order to increase the score of a large node $\mathbf{A}$ beyond $score^{max}$. Without a compromised large node, the attacker cannot inject messages directly, but has to wait until the compromised small nodes are involved in the creation of a report. Table 6.6 shows different attack strategies for framing a large node. The following example explains the values in the first column:

The attacker uses one compromised small node to falsify one SMAC-component for each report (first row). The score of $\mathbf{A}$ is increased by $\frac{1}{6}$, which means that seven such reports are necessary to remove $\mathbf{A}$ from the network (second row). Every compromised small node can only participate in one such report because of the LE-key revocation. This means, that the attacker has to compromise seven nodes to successfully frame $\mathbf{A}$. The score of each compromised node is increased by $\frac{1}{4}$ when the report reaches the sink. This means that each node can be "(re)used" for frameup attacks against other large nodes up to five times before it is removed from the network (last row).

Table 6.6 shows that this example is the most efficient attack strategy, which requires compromising seven small nodes, but as a result five large nodes can be removed from the network.

An attacker can also try to frame small nodes after compromising one large node. Table 6.7 evaluates the strategies for this approach. The values in the second column result from the following example: The attacker compromises one large node and one small node (first row). The keys stored in these nodes enable the attacker to inject degenerated reports with one correct and $T-1 = 4$ false SMAC-components, which results in $score^{+}_{\mathcal{G}_R}(4) = \frac{1}{7}$ and $score^{+}_{TN_R}(4) = \frac{1}{3}$. Thus he has to create 8 such reports for a successful frameup attack (second row). Again, each large node can only be used for one such report, so 8 large

Key description	number	estimate
LEAP cluster keys	d	10
LEAP pair-wise keys	d	10
μTESLA verifier	1	1
LBI-Keys	c	$1 \ldots 4$
LE-Keys	$c \cdot L$	$4 \ldots 16$
	total number of keys	$26 \ldots 41$
	memory consumption in bytes ($8\frac{\text{bytes}}{\text{key}}$)	$208 \ldots 321$

Table 6.8: Memory consumption of HEFS-keys. The third columns expects values for $d = 10$, $c = 1 \ldots 4$, $L = 4$

nodes must be compromised (third row) to frame the $T - 1 = 4$ innocent small nodes (fourth row) involved in the report. He can use every large node for 4 reports (fifth row) removing altogether $4 \cdot 4 = 16$ small nodes from the network (last row).

This kind of frameup attack is only feasible if L is larger than the total number of compromised large nodes. In the evaluated configurations, compromising the $L = 4$ potential transition nodes of a small node $\mathbf{a}$ is sufficient to isolate $\mathbf{a}$ from the network.

In conclusion, the dynamic scoring system makes frameup attacks difficult even if the threshold $score^{max}$ is low. In the static scoring system, the number of injected reports before detection k is equal to the number of nodes that are needed for a successful frameup attack. In the evaluated configuration, the dynamic scoring system forces the attacker to compromise seven nodes in a configuration that provides $k = 3$.

6.5 Memory usage

The data that needs to be stored in a sensor node's memory can be separated into static and dynamic data. Static data includes data that needs to be stored *all the time* such as cryptographic keys. Dynamic data includes temporary data such as message buffers.

Static data Table 6.8 summarizes the memory consumption of cryptographic keys in HEFS. This includes the keys used by LEAP and μTESLA. Cluster and pairwise LEAP-keys are necessary for the local report generation and secure routing. The LEAP group key and the LEAP individual key can be omitted here: The group key is not used in any part of the scheme, assuming that broadcasts are re-encrypted with cluster keys after each hop. The individual key is not needed because the LBI-key has the same function in HEFS. The number of needed LEAP keys depends on the node density d in the network, which is the number of one-hop communication partners of a node. For the estimation of consumed memory, we assume that $d = 10$. Generally, it is

important that d is significantly larger than T in order to allow a cluster-based report generation, even if some neighbors fail.

The number of localized keys depends on the cell size and the verification range: For each cell that is within verification range, a key must be generated. In the table, c is the number of cells in verification range. For the estimate, we assume that the edge of a cell is larger than twice the verification distance. This means that $c \in \{1, 2, 3, 4\}$, depending on whether the node is placed in the middle, at the edge or in the corner of a cell.

Other sources of memory consumption are the routing tables. In HEFS, two routing protocols must be implemented on each small node, which means that multiple routing tables must be stored in the memory. In the simulator implementation, the routing table of the TRP consumes $6 \cdot L$ bytes. The SRP uses about 8 byte. In a real deployment, we can assume that these components consume more memory in order to provide a higher robustness against node failure.

Dynamic data Dynamic data does not occupy memory all the time but only temporarily. The consumption sometimes depends on the number of events and number of messages that the network processes at the same time.

- In the μTESLA protocol, every broadcast message must be stored until the commitment value is received by the node. HEFS uses μTESLA to broadcast node revocation messages. It is important, that these messages are not ignored because of full message buffers.

- In step two of the local report generation, helper nodes supply their SMAC-part (SM_i) to the cluster head. In step four, they must verify that $SM_i \in$SMAC. Thus they must store SM_i in the memory. If a node assists multiple cluster heads in the report generation, it may have to store multiple SM_i values at the same time, each of which consists of 8 bytes.

- When a node acts as cluster head in the local report generation, it collects messages from neighboring nodes in steps 3 and 5. The number of messages (and thus the number of stored bytes b) is limited by the number of communication neighbors d. In step 3, at most d payloads of a `COOP2SMACPartMessage` have to be stored. Assuming $T = 5$, $d = 10$ and $L = 4$, the memory consumption in the worst case is

$$d \cdot 2 + 2 \cdot L + 8 = 10 \cdot 18 = 180 \text{ bytes} \tag{6.14}$$

  In step 5, T payloads of a `COOP4LMACPartMessage` have to be stored, which resolves to

$$T \cdot 2 + 8 = 5 \cdot 10 = 50 \text{ bytes} \tag{6.15}$$

  At this time, the messages stored in step 3 can be removed from memory. Thus about 180 bytes have to be temporarily available to a cluster head during the report generation.

occurences	routing table			
63	44	45	54	55
63	7	8	17	18
64	4	5	14	15
65	51	52	61	62
66	87	88	97	98
68	3	4	13	14
68	61	62	71	72
72	85	86	95	96
80	81	82	91	92
86	89	90	99	100
104	1	2	11	12

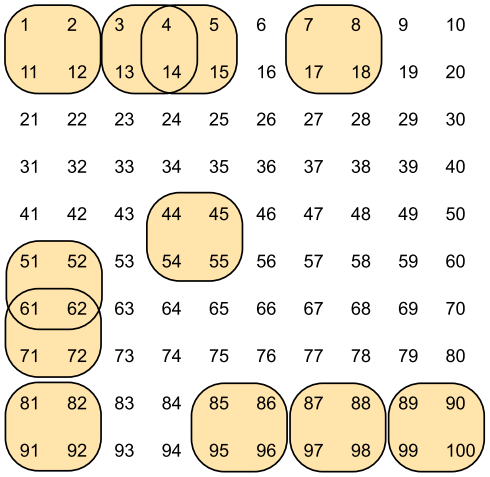


Figure 6.9: Most numerous TRP routing targets and their location

Summary If we assume an available memory size of 4 kBytes then the total memory consumption of $180 + 321 = 501$ bytes is acceptable for a security framework. Still, a reduction of the overhead should be subject to future work. We can abandon the LEAP cluster keys if the cluster head sends the `COOP1StimulusMessage` and `COOP3CollectedSMACMessage` individually to each neighbor. This would, however, consume more energy due to message transmissions.

6.6 Failing large nodes

The TRP stores routes from each small node $\mathbf{a}$ to the L nearest large nodes. When these L large nodes fail or are removed by the sink, the node $\mathbf{a}$ is practically isolated from the network:

- It cannot be used as a helper node or as cluster head to create a report.
- It cannot forward any TRP messages.

The only thing it can *can* do, independently of the TRP routing table, is forward SRP messages.

An attacker might take advantage of this fact and try to disable parts of the network by compromising or destroying large nodes. In this section, we assume that the attacker destroys exactly L large nodes to isolate a maximal number of nodes.

In order to evaluate the impact, we have analyzed the routing tables of the TRP in all small nodes of a scenario. In the context of this section, a *routing table* is a quadrupel $(ID_{\mathbf{A}_1}, ID_{\mathbf{A}_2}, ID_{\mathbf{A}_3}, ID_{\mathbf{A}_4})$ of routing targets. *The number of occurences of a routing table* is the number of small nodes that have exactly the same targets in their table. Figure 6.9 shows the most numerous TRP routing tables in a quadratic scenario with $n = 100$ large nodes. We can see that the attacker could isolate 104 small nodes by destroying the large nodes

routing table

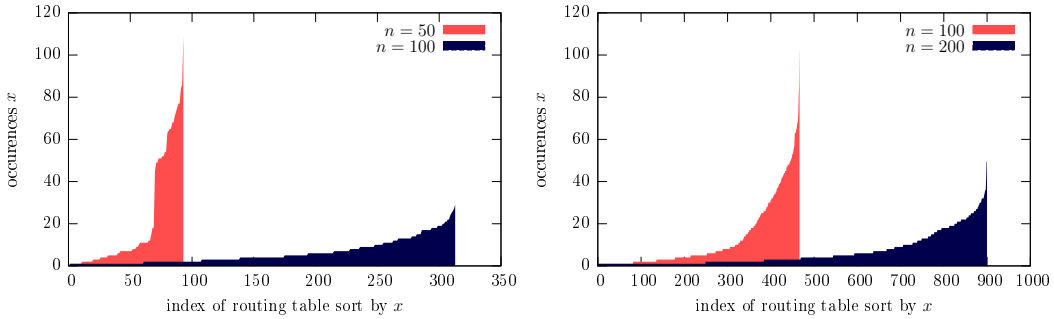


Figure 6.10: Number of small nodes with the same potential transition nodes in the prolate scenario (left) and the quadratic scenario (right)

1, 2, 11 and 12. These are the nodes in the upper left corner of the deployment area. We can also see that the ten most numerous routing tables all contain a square of large nodes at the border of the deployment area.

In figure 6.10, the routing tables have been sorted by their number of occurences x. The x-axis shows the index of the routing table sorted by x and the y-axis shows the number of occurences.

We can see, that a larger number of large nodes results in more diversity in the routing tables, so that the destruction of large nodes has a lower impact.

Chapter 7

Conclusion & Future Work

In this thesis, we have developed a *hybrid en-route filtering scheme* (HEFS) for heterogeneous sensor networks. The term "heterogeneous" refers to the fact that different types of sensor nodes with different resources are used in the network. HEFS is a "hybrid" scheme because it employs public-key cryptography as well as symmetric algorithms.

HEFS limits the impact of pDoS-attacks to a small area in the network. The localization of endorsement keys ensures that false data injection attacks are limited to the area of the compromised nodes: the attacker cannot create reports of events in a certain region unless he has compromised enough nodes in that particular part of the network. The use of public-key cryptography on large nodes ensures that endorsement keys are not distributed on verifying nodes. The public key signature also enables the sink to identify and remove copromised nodes from the network. The result is a graceful degradation in the sense that the compromised parts of the network degrade while the rest of the network remains intact.

We have implemented a sensor network simulation framework in Java and used this framework to develop a proof-of-concept implementation of HEFS.

The evaluation has shown that HEFS can significantly reduce the impact of pDoS-attacks on a sensor network. We have also noticed that the sink routing protocol reduces the network traffic significantly, even if HEFS is not applied.

Future work

Some aspects of HEFS and the underlying protocols have not been investigated in this thesis:

Applications of the transition routing protocol In future works, the transition routing protocol could be extended to work as a basis for other routing protocols. The small nodes only need one protocol implementation and large nodes can mantain multiple routing tables on top of the TRP. The following applications could be build upon the TRP:

- **a more sophisticated version of the sink routing protocol:** The SRP in this thesis has the problem that large nodes are avoided if they

are too far off-the-route. A sink routing protocol built on the basis of the TRP would always send messages via the next large node in direction of the sink. Additionally, such a protocol could more easily find alternative paths in case of node failure because large nodes can maintain flexible and large routing tables while small nodes only forward TRP messages.

- **a protocol for sink broadcasts:** One of the major energy consumers of HEFS is the revocation broadcast sent by the sink in reaction to degenerated reports. On the basis of the TRP, the broadcast message could first be sent to large nodes in the region of the keys or nodes that are due to revocation. Theses target nodes could then broadcast the message in their region.

  Such a protocol could prevent that an attacker injects revocation broadcasts with the goal of consuming the energy of all small nodes in the network. μTESLA cannot prevent such an attack because the message cannot be verified while the broadcast takes place.

Memory consumption of LEAP A large part of HEFS' memory consumption is caused by the pairwise keys and cluster keys needed by LEAP. Since HEFS already provides authentication through the SMAC and LMAC, it should be investigated, whether some or all keys of the LEAP-framework can be omitted.

Location-binding for public-key signatures With the current version of HEFS, the following problem still remains:

An attacker who compromises a large node and L small nodes can perform a pDoS-attack with valid SRP-messages. He can even inject messages into every part of the network. Yanchao Zhang et al. present an ID-based scheme with localized private keys [37]. This location information becomes part of the signature and can be verified by en-route nodes.

Such a scheme would allow verifying large nodes to drop messages that do not originate in their upstream area. It would also allow the sink to perform a plausibility test by comparing the location of the report to the location of the signing large node.

Cryptographic schemes There cryptographic signature schemes, such as SFLASH [8] and QUARTZ [24], that have been developed explicitely to reduce the length of the digital signature. QUARTZ claims to provide security equivalent to a 80-bit symmetric key with a signature length of 128 bits. The drawback of this scheme is the size of the public key (71 kBytes). Nevertheless, such schemes are worth being investigated for the use in heterogeneous sensor network because the small signature size reduces the energy consumption of small nodes.

Re-deployment of nodes HEFS makes the assumption that the attacker cannot compromise nodes during the initialization phase. The re-deployment

of nodes is difficult because the secrets used to compute shared keys are deleted at the end of the initialization phase. The re-creation of shared keys is an important topic in two cases:

- If a small node has lost all potential transition nodes due to node failure or infiltration, it should be able to negotiate new LE-keys with other large nodes.
- If a new large node is introduced into the network, LE-keys must be setup with nearby small nodes.

Security analysis of bloom-filters Bloom-filters have been optimized and evaluated in many non-security applications, but the use in security applications is relatively new. We have not found an analysis of the security properties of bloom-filters so far. This should be done, before bloom-filters are really used to compress multiple MACs.

Composition of published concepts Most of the publications (and this thesis as well) reduce their scope to a small number of problems in the area of sensor network security. This approach is certainly beneficial for the analysis of problem details, but it also hides the problems that arise in the composition of different concepts. A general problem is, that the memory and transmission overhead of multiple combined schemes adds up, which reduces the memory available to the application.

A large part of future work in the whole area of sensor networks consists of combining concepts and resolving incompatibilities. With regard to this thesis, this means the integration of methods to prevent denial-of-service attacks such as the *blackhole attack*.

Appendix A

Notation

Message transmissions

$\mathbf{a} \longrightarrow \mathbf{b} : \text{msg}$	Node $\mathbf{a}$ sends msg to node $\mathbf{b}$
$\mathbf{a} \longrightarrow * : \text{msg}$	Node $\mathbf{a}$ sends a message to the broadcast address. Such a message is received by all one-hop neighbors, but not retransmitted to other nodes.
$\mathbf{a} \longrightarrow ** : \text{msg}$	Node $\mathbf{a}$ sends a multi-hop broadcast message. The message is a sent to the broadcast address and retransmitted by every receiving node. This mechanism is also called *flooding*.

Sensor nodes

$\mathbf{a}$, $\mathbf{b}$, $\mathbf{x}$	Small nodes
$\mathbf{A}$, $\mathbf{B}$	Large nodes
CH	Cluster head
TN_R	Transition node of a report R
$\mathcal{G}_R$	Small nodes with invalid SMAC-component
$\overline{\mathcal{G}_R}$	Small nodes with valid SMAC-component
$\mathcal{TN}_{\mathbf{a}}$	potential transition nodes of $\mathbf{a}$

Signatures, MACs and keys

$\langle D \rangle_{\mathbf{A}}$	data D with an attached public-key signature computed with the private key$\kappa^{sec}_{\mathbf{A}}$.
$\text{MAC}_K(D)$	MAC computed over data D with key K
MAC_K	MAC computed over implicit data with key K
$K^{lbi}_{\mathbf{a}}$	Location-based indiviual key of node $\mathbf{a}$
$K^{le}_{\mathbf{a},\mathbf{A}}$	Local endorsement key of $\mathbf{a}$ shared with $\mathbf{A}$
$\kappa^{pub}_{\mathbf{a}}, \kappa^{sec}_{\mathbf{A}}$	Public-private key pair of $\mathbf{A}$

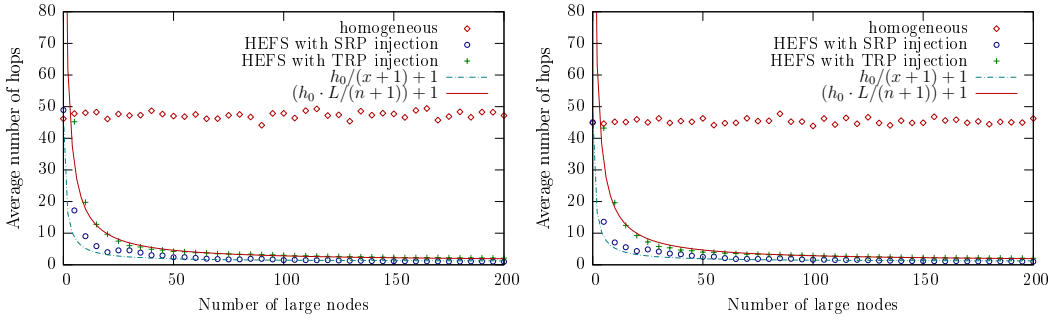
Hop counters (chapter 6)

h_0	Average hop count in a homogenous network
h_{SRP}	Average hop count of a SRP message before filtering
h_{TRP}	Average hop count of a TRP message before filtering

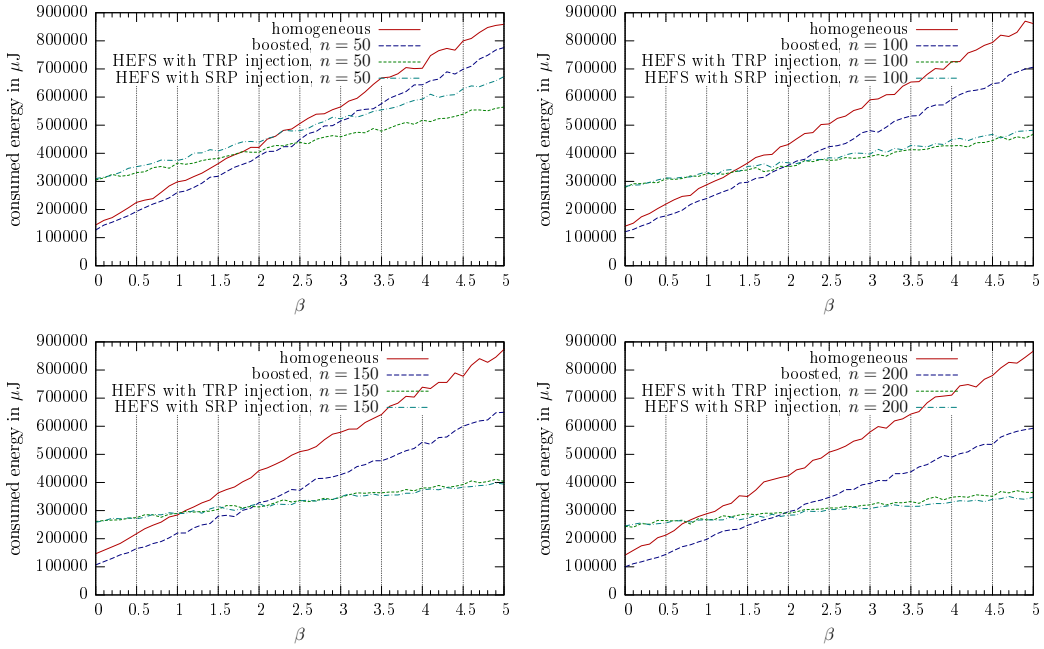
Appendix B

Simulation results

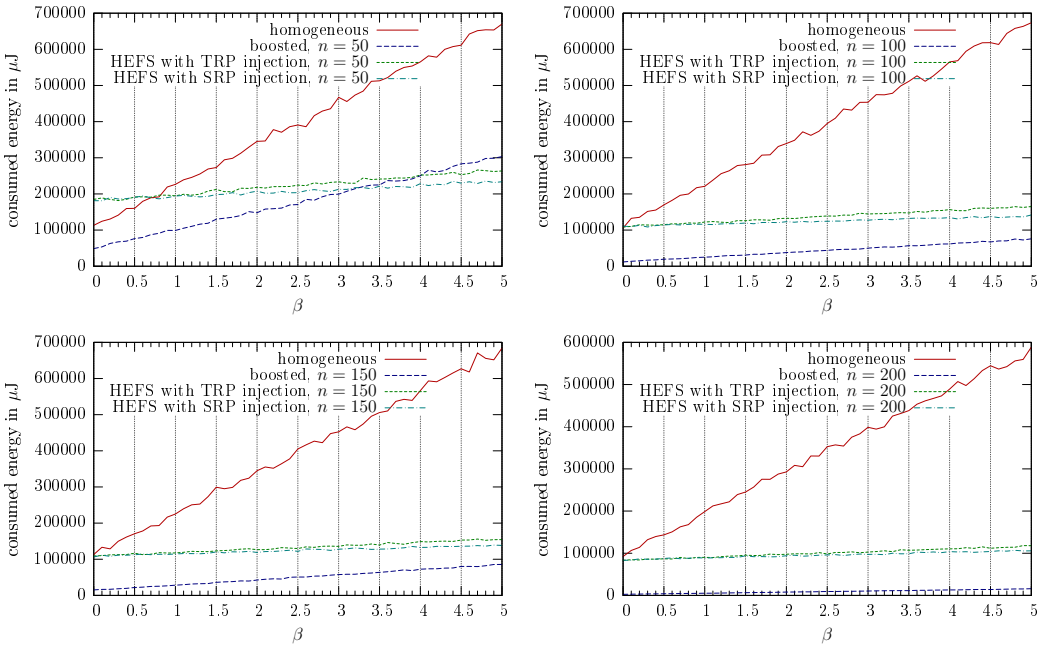
Travelled hops of an injected message with area size 4000×200 and 2000 nodes (left), 3000 nodes (right). Both graphs are almost identical.



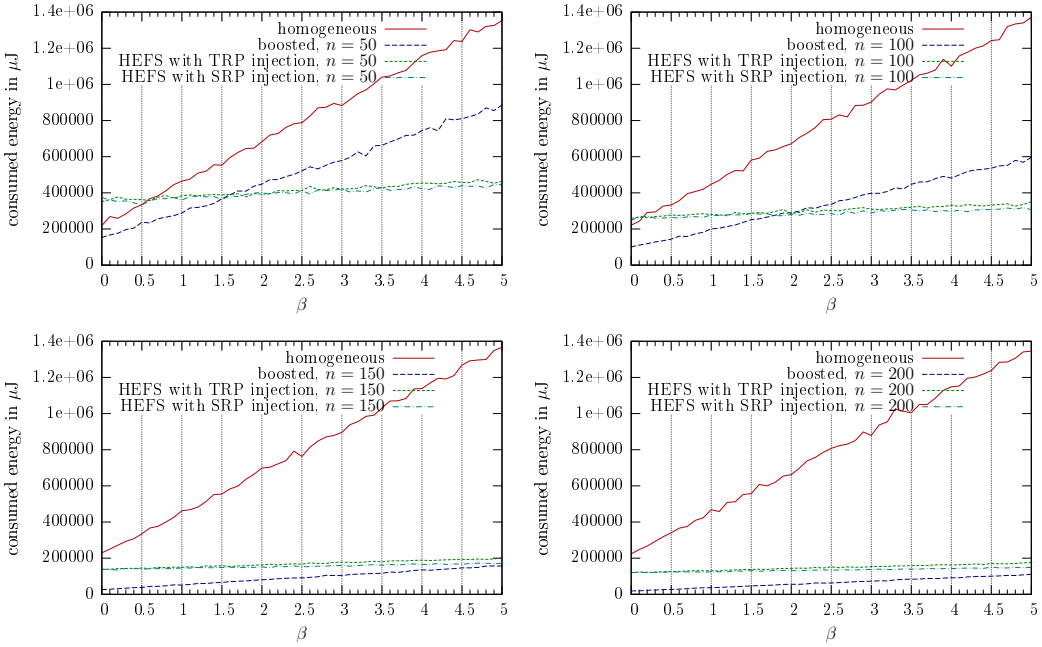
Consumed energy of one message and β injected messages for the quadratic scenario (2000×2000 with 6000 nodes and n large nodes)



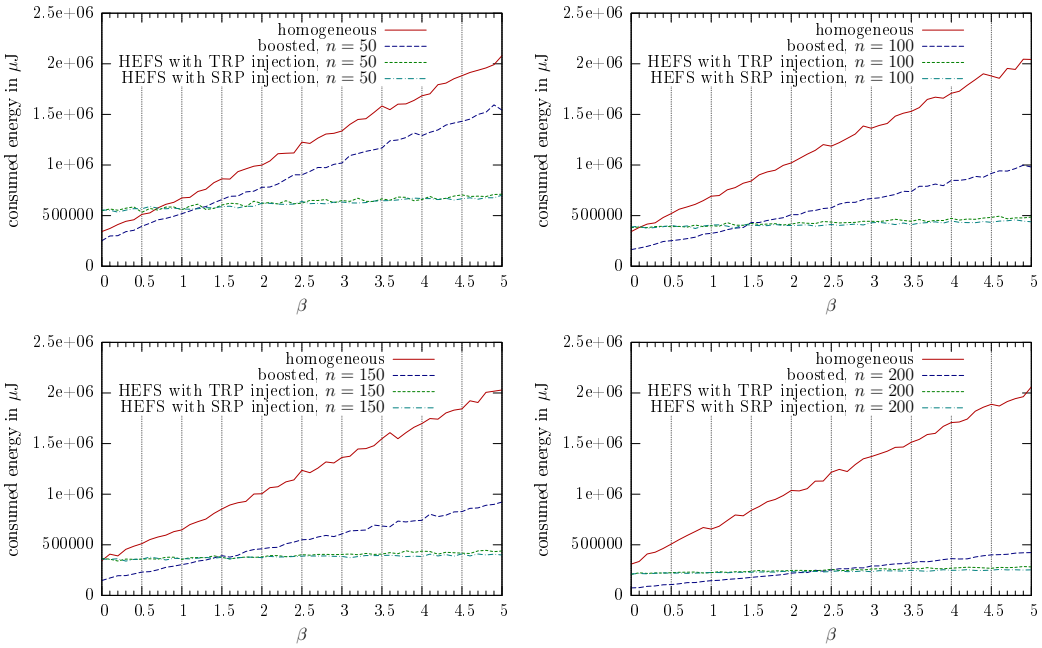
Consumed energy of one message and β injected messages for the prolate scenario (2000×200 with 1000 nodes and n large nodes)



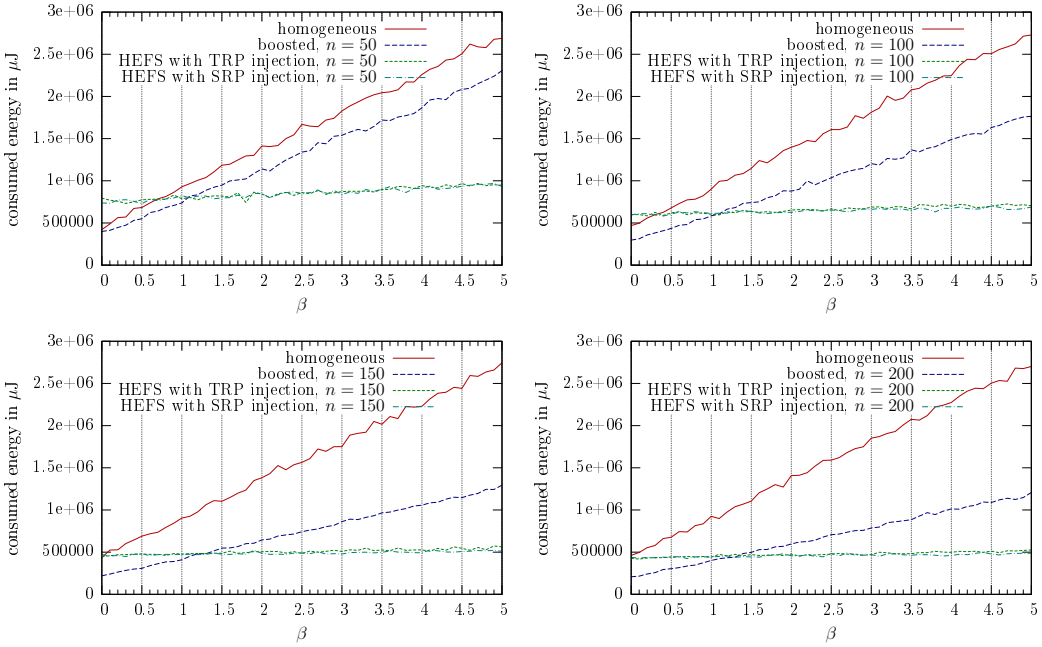
Consumed energy of one message and β injected messages for the prolate scenario (4000×200 with 2000 nodes and n large nodes)



Consumed energy of one message and β injected messages for the prolate scenario (6000×200 with 3000 nodes and n large nodes)



Consumed energy of one message and β injected messages for the prolate scenario (8000×200 with 4000 nodes and n large nodes)



Appendix C

Scenario definition files

A scenario definition consists of three files that have to be placed in the same directory: The file *layout.txt* contains information about the nodes, *topo.tex* contains information about the network connections and can be derived from *layout.txt*. The file *heat.txt* contains information about the heat model and is optional.

A data directory can be loaded using the `DataDirectory`-class in the package `org.knappi.wsn.definitionfiles`, which encapsulates an accessor class for each file. Figure C.1 shows a simple example for a layout file, a topology file and the resulting network.

layout.txt

This file is loaded by the class `LayoutFile`. The file consists of one row per sensor node. Each line has the following fields separated by spaces:

1. **nodeId**: The ID of the node
2. **x**: The x-location of the node
3. **y**: The y-location of the node
4. **type**: A string that defines the type of this node. This field can be *small*,*large* or *sink*.
5. **normal range**: The radio transmission range of the node, when it is sending without boost-mode.
6. **boost range**: The radio transmission range of the node, when it is sending in boost-mode. For the sink-routing protocol to work, this must be twice the *normal range* for large nodes. This value is irrelevant for small nodes.

topo.txt

This file is loaded by the class `RadioTopologyFile`. The file consists of one row per connection in the radio topology. The fields in each lines are

1. **startId**: The node ID of sending sensor node

layout.txt	*topo.txt*	Resulting network

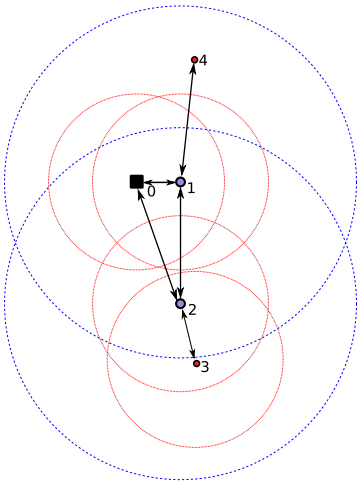
`3 70 246 small 60 60`
`2 60 208 large 60 120`
`4 70 41 small 60 60`
`1 60 125 large 60 120`
`0 30 125 sink 60 60`
	`3 3 false`
	`3 2 false`
	`2 3 false`
	`2 2 false`
	`2 1 true`
	`2 0 true`
	`4 4 false`
	`1 2 true`
	`1 4 true`
	`1 1 false`
	`1 0 false`
	`0 1 false`
	`0 0 false`
		

Figure C.1: Sample network scenario without a heat model

2. **endId**: The node ID of the receiving sensor node
3. **boost**: *True* if the receiving node is within the boost range of the transmitting node. *False* if the receiving node is within the normal radio-range of the transmitting node.

Connections from a node to itself are allowed in the file. The radio model of the simulator prevents messages to be received by the transmitting node.

heat.txt

This file is loaded by the class `HeatFile`. The first line of this file has two fields separated by spaces:

1. **ambient heat**: The ambient heat value
2. **decrease**: The decrement value of the heat model

Each of the following lines defines one heat source:

1. **x**: The x-location of the heat source
2. **y**: The y-location of the heat-source
3. **value**: The temperature of this heat source

A an example for the contents of a heat file is

```
20.0 0.5
611 518 29.355995310402015
623 560 27.597014089728027
681 563 43.941141713641514
628 517 39.44759026891465
```

Bibliography

[1] http://www.btnode.ethz.ch/Projects/Mica2 (11th January 2007).

[2] http://www.flickr.com/photos/projectsunspot/240917887/ (11th January 2007).

[3] Digital realtime computation laboratory, ohio state university. http://www.ece.osu.edu/drcl/ (11th January 2007).

[4] Java sun spot. http://www.sunspotworld.com/ (11th January 2007).

[5] MICA2: Wireless measurement system. http://www.xbow.com/ (11th January 2007).

[6] NIST, digital signature standard (dss), fips pub 186-2. http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf (11th January 2007).

[7] Shot spotter. http://www.shotspotter.com/ (11th January 2007).

[8] Mehdi-Laurent Akkar, Nicolas Courtois, Romain Duteuil, and Louis Goubin.
A fast and secure implementation of sflash.
In *PKC '03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 267–278, London, UK, 2003. Springer-Verlag.

[9] Christopher Benjamin.
Shot spotter and faceit: The tools of mass monitoring.
UCLA Journal of Law and Technology, 2002.
http://www.lawtechjournal.com/articles/2002/02_020421_benjamin.pdf (11th January 2007).

[10] Burton H. Bloom.
Space/time trade-offs in hash coding with allowable errors.
Commun. ACM, 13(7):422–426, 1970.

[11] Alan Colvin.
CSMA with collision avoidance.
Computer Communications, 1983.

[12] Ana Paula R. da Silva, Marcelo H. T. Martins, Bruno P. S. Rocha, Antonio A. F. Loureiro, Linnyer B. Ruiz, and Hao Chi Wong.

Decentralized intrusion detection in wireless sensor networks.
In *Q2SWinet '05: Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*, pages 16–23, New York, NY, USA, 2005. ACM Press.

[13] Jing Deng, Richard Han, and Shivakant Mishra.
Defending against path-based dos attacks in wireless sensor networks.
In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 89–96, New York, NY, USA, 2005. ACM Press.

[14] John R. Douceur.
The sybil attack.
In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[15] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler.
The nesc language: A holistic approach to networked embedded systems.
Proceedings of Programming Language Design and Implementation (PLDI), June 2003.

[16] Christian Gehrmann and Mats Näslund.
ECRYPT yearly report on algorithms and keysizes (2005).
Technical report, Europan Network of Excellenz in Cryptology, 2006.

[17] Germano Guimaraes, Eduardo Souto, Djamel Sadok, and Judith Kelner.
Evaluation of security mechanisms in wireless sensor networks.
In *ICW '05: Proceedings of the 2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05)*, pages 428–433, Washington, DC, USA, 2005. IEEE Computer Society.

[18] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister.
System architecture directions for network sensors.
ASPLOS 2000, November 2000.

[19] David B Johnson and David A Maltz.
Dynamic source routing in ad hoc wireless networks.
In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[20] Chris Karlof and David Wagner.
Secure routing in wireless sensor networks: attacks and countermeasures.
In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 113–127, 2003.

[21] Leslie Lamport.
Password authentication with insecure communication.
Commun. ACM, 24(11):770–772, 1981.

[22] Philip Levis, Nelson Lee, Matt Welsh, and David Culler.

TOSSIM: Accurate and scalable simulation of entire tinyos applications.
Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003), November 2003.

[23] Christian U. Grosse Markus Krüger.
Structural health monitoring with wireless sensor networks.
Otto-Graf-Journal, Materialprüfungsanstalt Universität Stuttgart, 15, 2004.
http://www.mpa.uni-stuttgart.de/publikationen/otto_graf_journal/ogj_2004/beitrag_krueger.pdf (11th January 2007).

[24] Jacques Patarin, Nicolas Courtois, and Louis Goubin.
Quartz, 128-bit long digital signatures.
In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 282–297, London, UK, 2001. Springer-Verlag.

[25] C. Perkins, E. Belding-Royer, and S. Das.
Ad hoc On-Demand Distance Vector (AODV) Routing.
RFC 3561 (Experimental), July 2003.

[26] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler.
SPINS: security protocols for sensor networks.
Wirel. Netw., 8(5):521–534, 2002.

[27] R. L. Rivest, A. Shamir, and L. Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
Commun. ACM, 21(2):120–126, 1978.

[28] Adi Shamir.
Identity-based cryptosystems and signature schemes.
In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[29] Alan F. Smeaton and Mike McHugh.
Towards event detection in an audio-based sensor network.
In *VSSN '05: Proceedings of the third ACM international workshop on Video surveillance & sensor networks*, pages 87–94, New York, NY, USA, 2005. ACM Press.

[30] Ahmed Sobeih, Wei-Peng Chen, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Hung-Ying Tyan, and Honghai Zhang.
J-Sim: A simulation environment for wireless sensor networks.
In *ANSS '05: Proceedings of the 38th annual Symposium on Simulation*, pages 175–187, Washington, DC, USA, 2005. IEEE Computer Society.

[31] Raylin Tso, Chunxiang Gu, Takeshi Okamoto, and Eiji Okamoto.
An efficient id-based digital signature with message recovery based on pairing.
Cryptology ePrint Archive, Report 2006/195, 2006.
http://eprint.iacr.org/.

[32] Arvinderpal S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz.
Energy analysis of public-key cryptography for wireless sensor networks.
In *PerCom '05: Third IEEE International Conference on Pervasive Computing and Communications*, pages 324–328, 2005.

[33] Ronald Watro, Derrick Kong, Sue fen Cuti, Charles Gardiner, Charles Lynn, and Peter Kruus.
Tinypk: securing sensor networks with public key technology.
In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 59–64, New York, NY, USA, 2004. ACM Press.

[34] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin.
A wireless sensor network for structural monitoring.
In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, New York, NY, USA, 2004. ACM Press.

[35] Hao Yang, Fan Ye, Yuan Yuan, Songwu Lu, and William Arbaugh.
Toward resilient security in wireless sensor networks.
In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 34–45, New York, NY, USA, 2005. ACM Press.

[36] Fan Ye, Haiyun Luo, Songwu Lu, and Lixia Zhang.
Statistical en-route filtering of injected false data in sensor networks.
In *Proceedings IEEE INFOCOM*, 2004.

[37] Yanchao Zhang, Wei Liu, Wenjing Lou, and Yuguang Fang.
Location-based compromise-tolerant security mechanisms for wireless sensor networks.
IEEE Journal on Selected Areas in Communications, 2006.

[38] Li Zhou and Chinya V. Ravishankar.
A fault localized scheme for false report filtering in sensor networks.
In *ICPS '05: Proceedings of the International Conference on Pervasive Services*, pages 59–68. IEEE, 2005.

[39] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia.
LEAP: efficient security mechanisms for large-scale distributed sensor networks.
In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 62–72, New York, NY, USA, 2003. ACM Press.

Index