Evasive Data Storage in Sensor Networks

by Peter Maximilian Cholewinski

Diploma Thesis
in
Computer Science

Laboratory for Dependable Distributed Systems
RWTH Aachen

First evaluator: Prof. Dr.-Ing. Felix C. Freiling
Second evaluator: Prof. Dr. rer. nat. Otto Spaniol

August, 2005

Peter Maximilian Cholewinski
Student Identification Number: ——
Born on: October the 27th, 1982
Email: Peter.Cholewinski@Rwth-Aachen.De

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

(Peter Maximilian Cholewinski)

Acknowledgments

I am very grateful to my advisor, Professor Felix Freiling, not only for providing unwavering support and helpful suggestions throughout several stages of the writing process, but also for his overall guidance and encouragement. Furthermore, I would like to thank Professor Otto Spaniol for finding the time and effort to evaluate this thesis.

出る杭は打たれる

deru kui wa utareru

Those who stand out will receive trouble.

Abstract

Sensor Networks are an emerging technology with a substantial potential to reshape the landscape in a myriad of areas. As with all technologies that involve information a significant effort has to be made in order to guarantee security. In case of sensor networks a special problem, the *Node Capture Problem*, is considered to be one of the most challenging ones, where particularly the aspect of this problem involving protection of data will be addressed in scope of this work. In order to pursue this goal, the thesis uses two fundamental notions: *Camouflage* and *Evasive Data Storage*. In case of Camouflage, current work is explored, a generalization of the most important current algorithm is given and several new Camouflage algorithms are developed. However, the most emphasis is placed on the investigation of the new Evasive Data Storage notion, which improves security from a different point of view. All algorithms are described in detail using formal as well as simulative means. Furthermore, in the course of the work the concept of a *Choice Function* is identified as a manifold element whose usefulness goes even beyond Camouflage or Evasive Data Storage.

Zusammenfassung

Sensor Netzwerke sind eine neue Technologie mit einem beträchtlichen Potential eine Vielzahl von Bereichen zu verändern. Vergleichbar mit anderen Technologien, welche die Verwaltung von Informationen umfassen, muss die Sicherheit von Daten garantiert werden. Im Falle von Sensor Netzwerken stellt das *Node Capture Problem* eines der herausfordernsten Probleme dar, wobei der Aspekt der Datensicherheit dieses Problems in dieser Arbeit besondere Aufmerksamkeit erhält. Um dieses Ziel zu verfolgen, macht sich die Arbeit zwei grundlegende Techniken zunutze: *Camouflage* und *Evasive Data Storage*. Im Falle von Camouflage werden derzeitige Verfahren betrachtet, eine Generalisierung des wichtigesten Verfahrens wird beschrieben und einige neue Camouflage Verfahren werden entwickelt. Jedoch wird der Schwerpunkt der Arbeit auf die Idee des Evasive Data Storage gelegt, welche Sicherheit aus einer anderen Perspektive verbessert. Alle Algorithmen werden detailliert beschrieben, wobei sowohl formale als auch simulative Mittel hierzu benutzt werden. Desweiteren, wird im Laufe der Arbeit das Konzept der *Choice Function* als ausserordentlich vielfältiges Element identifiziert, dessen Nützlichkeit auch über Camouflage und Evasive Data Storage hinausgeht.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The recent decade has irrefutable proven that mankind has entered what can truly be described by the term the *Information Age*. Areas ranging from business to military over to personal matters seem to be affected and subject to change due to the information that has become available and demanded everywhere. Of course, over the history information has already been recognized as a valuable asset. At the times where the Roman Empire was considered to be the center of the world, strategical information that contained positions of enemy troops or numbers of their magnitude were considered to be crucial ingredients in deciding over victory or loss. Hence, at that times the first attempts were made to protect such data or also protect the delivery of key commands. Over the centuries, information has slowly become more and more important. Similarly, several centuries after the fall of the Roman Empire, military success depended heavily on obtaining detailed information about enemy territory in forms of maps. Hence, once again victory or loss was not depending on the magnitude of troops available, but if information was available or not. Nowadays, information has reached a seemingly climax point, there is just not a single moment where information is not available or sought after and has become critical to more than just the military.

Seeing how important and influential information has become, more emphasis has to be laid on its protection. Governments, industry giants and even individuals long for protection of their data, especially when exposed to the threat of malicious users illegally gaining access to their valuable information. Therefore, it is not surprising that the last century has seen several milestones in the field of cryptology, which can be seen as the main means used to protect data. However, under slightly different circumstances cryptography cannot be applied to the extend where the system can be considered secure. This is particularly true in case of such emerging technologies as *Sensor Networks*. Such networks, which are primary used for gathering data about the environment they are installed in, imposed several restrictions that simply prohibit the usage of strong encryption: limited energy, limited resources and physical accessibility, to name a few. Thus, other means for the protection of information stored in those networks have to be investigated. This is a challenging problem and will surely be even more crucial, when the technology becomes widely adapted. Imagining such a network, which consists of a myriad, i.e. hundreds and thousands, of small computing devices that have sensors to gather information, a simple question arises: Can accumulated information be protected? What are the properties that need to be considered? Especially the assumptions about a malicious entity that poses the main threat to gathered information have to be worked out.

In the course of research on the subject of Sensor Networks, a special problem has been identified to be one of the most challenging: the *Node Capture Problem*. The basic picture is that a sensor network is installed in an environment, where a malicious user or adversary is present. Due to the nature of sensor networks, this adversary can access nodes physically. Doing so, allows the adversary to extract information contained in such nodes, which is surely not a desirable scenario. However, before the adversary can succeed by physical access, he has to identify a node that has stored valuable information within. That is the point where this thesis is making an attempt to increase security. To do so, the thesis starts with a Chapter describing sensor networks, investigating their properties, presenting current forms of data storage methods and defining the kinds of adversaries that can be present in the hostile environment. This provides the needed background knowledge, before the first technique is introduced along with different algorithms, whose goal is to hamper the identification of valuable nodes by camouflaging the traffic a possible adversary could observe. Thus *Camouflage Technique*s are considered to be of utmost importance to increase the security of information in sensor networks. The respective Chapter focuses on the introduction of current work on such techniques for sensor networks, which were developed mainly in [1] and also presents the most usable Camouflage algorithm, namely the Fractal Propagation algorithm. The same chapter also develops several new, special purpose Camouflage notions that will prove to be useful as well. Furthermore, one of important results is a natural generalization of the Fractal Propagation algorithm that will be proved to lead to superior algorithms compared to the original one. This generalization, however is investigated first in scope of the Simulation Chapter and given precisely in the appendix for various reasons. The second mean which is also considered to be the main topic of this thesis is the notion of *Evasive Data Storage*. This notion is developed in form of several algorithms in the Chapter named Data Oriented Techniques and is extended even further in the Advanced Data Oriented Techniques Chapter. The notion is different from the goal the Camouflage pursues, in the sense that Evasive Data Storage mainly increases the security of the network when an adversary found a node containing information, remembered the location of that node and wants to access the node again for updated information. Although, this thesis presents several different algorithms that provide means of storage using the Evasive Data Storage notion, the main ingredient of those procedures is the concept of a *Choice Function*. The work shows that this element is the main parameter that decides how well an Evasive Data Storage algorithm can guarantee certain security properties in the presence of an adversary. Furthermore, the work shows that the concept of the Choice Function is not limited to the Evasive Data Storage notion, but can also be used effectively in other circumstances, which is clearly shown by using Choice Functions to generate the natural generalization of the Fractal Propagation algorithm. Those results therefore indicate that Choice Functions are an important mean to both Camouflage and Evasive Data Storage, but surely can find application in other areas with similar properties.

The results regarding the Camouflage and Evasive Data Storage notions are mainly obtained by using Simulations and can be found in the respective Chapter. In case of, Evasive Data Storage several parts are also investigated using more formal means and are given in the chapter introducing the respective algorithm. Especially complexity issues and unreliability are investigated using those formal methods. The thesis finishes with an outlook into further paths to develop the notion of Evasive Data Storage and its integration with Camouflage and also provides a summary of the main results this thesis contains: the generalization of the Fractal Propagation Camouflage Algorithm, the effectiveness of Evasive Data Storage as well as their main ingredient the Choice Functions, and results showing the usefulness of the Data Splitting notion that is introduced as an advanced Evasive Data Storage technique.

Chapter 2
Data Security and Sensor Networks

Any system that stores, processes or gathers information, while not granting arbitrary persons access to that information, has to provide some means for data security. As sensor networks fall into this category in almost all of their applications, there is a need to clearly state how the domains of data accumulation and data security intersect and interact in such networks. The main focus of this chapter will be to provide background on the areas that are significant for investigating the relationship mentioned above: basics of sensor networks, data security in the context of sensor networks and data storage. First off, sensor networks are introduced and basic operations as well as problems in the realm of those networks are explained. Secondly, the important issue of data storage in sensor networks is described and current storage paradigms are introduced, providing an adequate overview of what to expect of current storage technology in such networks. Thirdly, data security is brought into the picture of sensor networks and data storage, especially the problem known by the term *Node Capture Problem* is emphasized. Sensor networks offer a lot of vulnerabilities that malicious users can take advantage of and the above mentioned problem is one of the major ones of that new and exciting technology. Hence towards the end of this chapter different possible adversaries with varying abilities are discussed and will prove useful in the evaluation of algorithms developed later on in this thesis. The chapter finishes with a section defining notation and terminology to ensure an unambiguous understanding of the material conveyed in the subsequent chapters.

2.1 A Sensor Network

Although not widely known and considered as a new technology, sensor networks have been around for more than a decade. The US military, in its efforts to create a sophisticated surveillance system, connected several (sensing) radar stations with each other to form a network in which information of each of its nodes (i.e. the radar stations) can be accessed and put into context with information obtained from other nodes. This simple notion provided a reliable monitoring system of enemies' actions and more importantly was one of the first appearances of so called Distributed Sensor Networks (DSN). The properties of such networks are quite straightforward: the nodes need to have sensing abilities in order to provide useful information of their surroundings and need to have means of communication to share that information and put it into context to the other participants' data. It is quite obvious that for most applications of such networks the gathered data has to be linked to the location of the node that generated the data, such that a proper way of putting the jigsaw puzzle of all the information obtained from all the nodes together is possible. Nonetheless the notion of DSNs has several limitation: the nodes need to be stationary, i.e. fixed at one location where a sufficiently powerful energy source is available, communication is conducted by wire and hence calls for additional deployment costs, and thus lacks the flexibility that could open up the door to a myriad of applications. Therefore, although an interesting notion, Distributed Sensor Networks did not attain much attention.

However, recently with the increasing advancements in wireless technology a new form of sensor networks emerged: Wireless Sensor Networks (WSN). The underlying idea is closely related to that of DSNs: having a set of sensing devices that somehow can reflect the environment they are deployed in. The inherent difference is that WSNs are freed from properties that would limit their flexibility, WSNs overcome the need for wires to communicate with each other by exchanging information with radio waves or similar wireless facilities. As a result deployment of such networks is easier, because there is no imminent need for corresponding wired

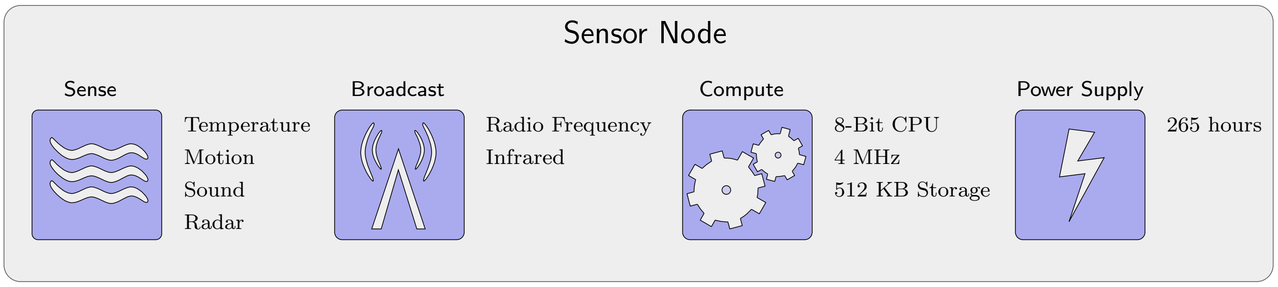


Figure 2.1: Basic Model of a Sensor Node

communication infrastructure. Furthermore, WSNs can be constructed from completely autonomous devices that have their own energy supply, which consequently implies even more applications. WSNs have become an active field of research lately and once this emerging technology is more mature, practical applications of such networks will be more and more tangible, even in everyday life. The main reasons for the prospective success of WSNs, as opposed to the DSNs, is their autonomy arising from wireless communication and independent source of energy of each participating nodes. Unfortunately, having such energy supplies also means that the energy that is available to nodes of such sensor networks is limited. But this is not the only deficiency that can be found in WSNs: the communication primitive is wireless and thus can be subject to eavesdropping. As a consequence, algorithms that are to be employed in WSNs not only have to be designed specifically with limited available energy in mind, but also bear in mind that malicious users can eavesdrop on the communication, take advantage of traffic-analysis and even worse: infiltrate the network.

This thesis will specifically address problems that can be found in Wireless Sensor Networks and hence, in the following, the term *sensor network* will refer to Wireless Sensor Networks, if not stated otherwise. Those networks, similarly to the DSNs mentioned above, consist primarily out of small devices that have their own power supply and a wireless communication primitive; but most importantly they have some sensing capabilities. In the following such a device will be referred to as a **sensor node**. Such a sensor node has three fundamental abilities: sensing, communicating, and computing. Figure 2.1 depicts those abilities along with several concrete examples or more detailed information, furthermore the model in the figure shows another relevant information: the life span of a sensor node that is limited due to the rather small energy source available to each node. Naturally, as time passes by, the development of more efficient power technologies will increase the available power for such small nodes, therefore elongating the overall life span of the whole sensor network. It might seem that this will at the same time alleviate the current severe need for efficient algorithms, specifically designed to be less energy consuming by reducing messages exchanged or the computations performed. Nonetheless, at the same time such energy sources become available there will be a call for even smaller devices, which will limit the life span again and thus create a new need for efficient algorithms. Therefore, putting an emphasis on the property that energy is scarce in such networks is not superfluous, even in case of significant advances in mobile energy sources. Moreover, the point above suggests that algorithms concerned with energy issues will remain crucial for a long time. Another basic part of the sensor node model are the sensing abilities, which can range from temperature, motion, seismic activity, to radar and depend on the the objectives of the sensor network. Combining several sensors on one node, i.e. creating a heterogenous sensor network with differentiated sensing abilities among the nodes, is also a possible scenario. A sensor node's communication primitive employs some kind of wireless technology, putting data to be broadcast to neighboring nodes in a dangerous situation. Such a primitive comes with obvious advantages, as mentioned above, but unfortunately makes communication in the sensor network susceptible to adversaries that can enter the network with wireless enabled devices and monitor data exchange or, even worse, send their own messages to nodes. Thus a wireless sensor network's communication is vulnerable to eavesdropping, traffic analysis and protocol attacks, particularly in form of

Denial Of Service (DOS). Additionally to the tasks of gathering and exchanging data, a sensor node possesses computing capabilities. Of course, those are needed for performing any sensing and communication, but are also intended to analyze data or to perform algorithms for securing the network. Due to the small dimensions of sensor nodes the computing capabilities are severely limited, resulting in a direct impact on the set of algorithms that can be usefully deployed on such a systems. The storage of a node, of course, faces the same severe restrictions and prohibits storing excessive amounts of data, especially when not directly related to the data the network is supposed to gathered from the environment (for instance, strong encryption keys or large routing tables). In order to keep a sensor network running as long as possible, it is crucial to ensure a long lifetime of the sensor nodes, which are the main building block of sensor networks. Hence, communication and computation performed by each node needs to be kept at a minimum. All algorithms that are employed on a sensor networks need to be designed with anticipated low power consumption in mind. This is also the cause that sensor networks offer such rich challenges, as ordinary algorithms used for wired networks cannot be simply used within the sensor network context, mostly because of such algorithms' highly computational demand and sometimes superfluous protocol messages. Communication overhead is nothing tolerable in sensor networks.

At this point, the presented picture of a sensor network contains the sensor node as the most important building block for this kind of networks. However the nature of those nodes is quite too limited for several applications. Therefore, most sensor networks have another integral part: the **base station**. Mostly assumed to be available only in minute numbers in realistic sensor networks, base stations weaken the property of ordinary sensor nodes that the available energy is severely limited. Thus, a base station is a node of a sensor network that has substantially more power to use and can therefore perform more computations, store larger amounts of data and communicate more intensely with its surrounding neighbors.[1] Hence, sensor networks can take advantage of this more powerful component and therefore extend the varieties of applications that can be addressed. Nonetheless, base stations can also become a target, in case the network's functionality heavily depends on them, or a bottle neck, when algorithms strongly rely on base stations to complete their tasks. Having seen positive as well as negative properties of base stations, it is important to keep in mind that in most realistic sensor networks base stations form a rare commodity, ranging from a single station to a few at most. In many applications, tasks performed in sensor networks rely on base stations, still it would be imprudent to design algorithms that make extensive use of base stations' capabilities. Such algorithms can be harmful as the failure of a single base station can render the functionality of such an algorithm futile; exceptions might be, for instance, fundamental primitives where extensive use of base stations is required, like simple cryptographic mechanisms to secure data exchange (see [10] for more detail). But a base station is not just a simple sensor nodes with a larger battery attached to it[2], a base station can have processing units, which are several magnitudes faster than those of simple sensor nodes or might offer communication primitives that have a noteworthy wider service area. [3] Furthermore, legitimate users at some point need to access the data gathered by a sensor network, calling for some kind of access point. Naturally, there is the possibility that they access the network by communicating with simple sensor nodes, but in most cases an access point is realized by the available base stations. The latter method, can either be a realized through a remote access via a satellite uplink or through a local physical access using some kind of wireless enabled device that the legitimate user carries around.

To sum up, sensor networks, as they will be seen in this thesis, consist of two basic devices: the sensing sensor nodes[4] and more powerful, but few in number, base stations. To get a feeling for the vast variety of possible applications that networks formed with those two components can create, refer to table 2.1. The table shows several small examples where sensor networks can offer useful solutions; the examples are divided by field of their application, once again emphasizing the flexibility and importance that comes with sensor networks. Now that the notion of sensor networks and some of their basic properties as well as their fundamental building

[1]Clearly, having more power allows for more powerful computing hardware, more storage, etc.
[2]And without sensing capabilities in most cases.
[3]However, in security critical applications of sensor networks, a measurable stronger communication primitive can lead adversaries to base stations. The result can be that base stations are put out of action, possibly rendering the whole network unusable, which is an avoidable scenario.
[4]In some cases the area covered by the sensing abilities of sensor nodes is referred to as **sensor field**.

Field	Description
military	In a military dispute, several thousand sensor nodes can be dropped over enemy's terrain to autonomously set up a sensor network and be used as an accurate mean for surveillance of the territory. This provides a tremendous advantage as military strategy can be adjusted accurately, possibly saving life.
military	Sensor networks can be deployed in cities in order to form a system for the detection of chemical, biological or nuclear materials, that might be used in a attack on the citizens. Early recognition of such an attack can initiate evacuation of other parts of the city not yet affected by any harming substance.
industry	Industrial machines can be equipped with sensors that accurately depict the current status of the machinery, precisely pinpointing any failures and contributing to fast and accurate maintenance.
industry	Traffic control can profit from the sensor network technology in allowing traffic lights to adapt accurately to the current traffic situation, even in a larger context than a single junction only.
industry	Monitoring movements of customers in a store through an installed sensor network can provide useful information for behavior analysis, and in the long run lead to a more comfortable shopping experience for customers.
academic	For geological research sensor networks can be deployed in remote, mostly unaccessible regions and provide useful data; for example, record climate changes or monitor seismic activity.
academic	Ample areas of wildlife sanctuaries can be covered with sensors and provide data for monitoring behavior of animals. Furthermore, such a network could also be used to protect species, if sensors could recognize poachers successfully.
personal	Sensor networks can be used to create so called smart housing concepts, where the house can become more conventional to habitants through noticing that the refrigerator lacks milk or provide more comfortable light switches. Furthermore, such in-house sensor networks can be used to more accurately recognize burglars, reacting more intelligently or recording in detail the actions of the intruder.

Table 2.1: Sensor Network Applications

blocks have been given, the following sections intend to provide a more elaborate and precise description of those networks, especially regarding data storage in such networks.

2.1.1 Big Picture, Properties and Problems

Having introduced sensor nodes and base stations, the basic building blocks of a sensor network, the next stage is to give a more precise picture of how these two are used in sensor networks, how they relate and what problems are to be expected. Furthermore, each sensor network is installed in some kind of environment, which the network has to monitor with the sensors available. Depending on the application, both the environment and the gathered information can vary, and other outside influences can be expected. Hence short investigation of those possibilities and their respective consequences are necessary to complete the introduction of sensor networks. At the end of this section, where the reader should have a pretty thorough intuition about sensor networks, the formal assumptions and the formal model that will be used in the algorithms to follow are presented.

To get started figure 2.2 shows a simplified model of a wireless sensor network. Instantly several assumptions that apply to sensor networks should be apparent: nodes are distributed densely in the environment and substantially exceed the amount of available base stations in the network. The number of sensor nodes in a

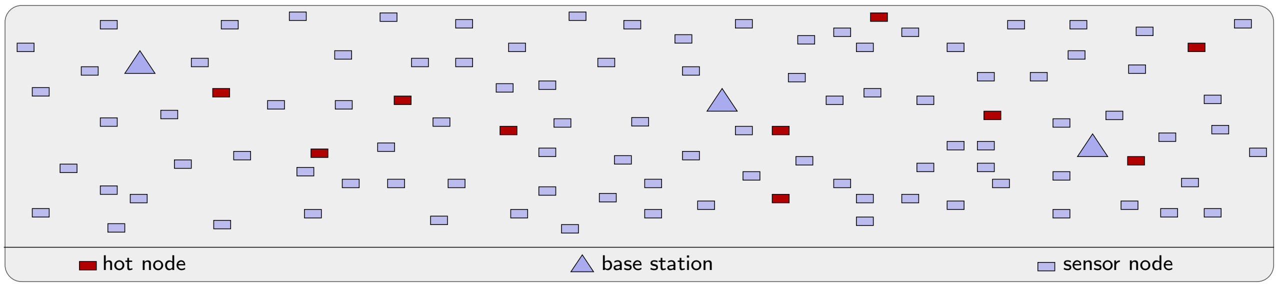


Figure 2.2: Basic Model of a Sensor Network

network could range from several hundred up to a million. The application determines what number is necessary, however base stations remain a rare commodity, a few base stations per network are assumable. There are cases where no base stations at all is available, nonetheless this work will not be concerned with such a case, although this assumption can be alleviated, if algorithms do not rely on the presence of base stations, in which case this will be pointed out by the algorithms respective description or will be simply clear. The relationship between the two kinds of nodes regarding their number can be stated mathematically as $n \ll b$, where n refers to the total number of sensor nodes in the network and b to the total number of base stations available. The vast number of sensor nodes is equipped with a certain device empowering it with sensing ability that reaches a limited area around the node. Adding all the covered areas of each node in the sensor network amounts to a quite large field that is observable by all the nodes, and is called the **sensor field**. The sensor network can be seen as a device reflecting this sensor field from the perspective of its sensors and making this observed state available as data to the users of the network. In the best case, the data stored in the sensor network will be accessed by the maintainer of the network (or other authorized parties) through the base stations, which either store the information themselves or need to query the network for the desired information.[5] In either case, the integral role of base stations is evident and thus they need to be protected from being recognized as such. The possibility of having the user query the sensor nodes directly, without any need for base stations, is also possible, but is less frequently found in current applications. Similar applies to this work, if not stated otherwise, the algorithms investigated will assume that queries are performed through access to the base stations.

As this work is focused on security problems related to the data stored within a sensor network, it is important to distinguish between nodes that actually hold data and those that do not. The former will be referred to by the term **hot node** and will play an integral role in the algorithms to follow. The latter, i.e. the nodes that do not store any data, will not be specially named in most cases; in extraordinary circumstances the term **cold node** can be used to refer to such nodes in order to stress the absence of data in those nodes. It is noteworthy that sensor networks[6] call for special data aggregation techniques, meaning the process of combining the data gathered by several nodes surrounding the location where an event occurred into a single data item that needs to be stored. It just does not make sense to store the sightings of the same animal, registered by 10 nodes located approximately at the same position, individually. Hence, there is a need for techniques dealing with such and similar situations, which are studied in more depth in the following papers [11], [12] and [13]. However, this work will not make this distinction and will stick to a simple data item that will be described in more detail in the respective data storage section. Now that the distinction has been made between hot and cold nodes, the question arises how those two kinds relate in a sensor networks. This heavily depends on the storage mechanism used, which itself depends on security, efficiency or similar metrics needed for the application the particular sensor network wants to address. Different basic paradigms concerned with storage, explained later in this chapter, will clarify the relation entailed by the respective paradigm. Nonetheless, assuming that

[5]This depends on the storage method and will be described in subsequent sections and chapters.
[6]This does not apply to all applications though.

the network contains h hot nodes, the assumption that will be prevalent through the algorithms introduced in this work is that the number h is substantially less than n, the number of nodes in the network.[7] This is also seen in figure 2.2 where the number of hot nodes (the red colored ones) is less than the amount of nodes not storing any gathered data.

In the context of hot nodes, it is important to mention their properties with respect to an **adversary**, who can be present in a sensor network's environment. Adversaries are entities that are assumed to have certain abilities and whose objectives are mostly defined to be harmful to the sensor network itself or to the objectives of the maintainer of the network. Hot nodes pose an especially interesting commodity for those adversaries, mostly because the data they gathered is of interest. Thinking of two competing companies, where one has obtained permission to deploy a sensor network and thus gained a competitive advantage over the other company, which then would have quite some interest in the data gathered in such a network. Hence the company that deployed the network seeks to protect its advantage and similarly the investment made in that sensor network. This exemplifies a scenario, where the network is exposed to a powerful adversary and hence needs protection. Another example, that shows a different perspective on data security is a sensor network used by military to monitor enemy terrain. The enemy has sufficient incentives to manipulate the data stored in the network, in order to render the network useless. Both examples, although slightly different in nature show that hot node protection in a **hostile environment**, i.e. an environment accommodating both sensor network and adversary at the same time, is a relevant topic and will play a major role in this thesis.

Now that the major concepts related to sensor networks and suitable data security issues have been introduced, the inherent properties that can be found in applications of sensor networks have to be investigated in order to motivate the assumptions made in the formal model presented in the next section. A sensor network consists (mostly) of a set of sensor nodes, which are densely distributed over an area (the sensing field) and can reach a magnitude of several hundred to several thousand. The area where the sensor nodes are located forms an environment that can influence properties of the network. Naturally, the environment where the nodes are deployed varies, depending mostly on the application of the network, but there are several properties that sensor networks exhibit in most practical environments and which will be taken to be true for the scope of this work:

- *topology of the sensor network*: the environment where nodes are located can have moving parts that can also dislocate sensor nodes. A simple example is monitoring wild life, where animals can move nodes, soil erosion can move large parts of nodes or humans can (intentionally or unintentionally) move sensor nodes around. Hence, sensor networks have to assume that their topology changes, and that such changes occur frequently.

- *connectivity loss*: sensor network take advantage of wireless communication primitives, that due to environmental influences can vary in their performance. A radio frequency signal can suffer severely from heavy rain or snow, disrupting temporarily communication in the affected part of the network.

- *sensor node failure*: similarly to the topology changes, sensor nodes can be destroyed by the environment. An animal can crush a node, rain could short circuit electronics or cars driving through a sensor network could destroy nodes. Additionally to the environmental forces that can damage sensor nodes, the network has inherently limited energy resources, hence nodes can fail to function due to energy exhaustion.

The properties stated above lightly touched on possible influences the presence of an adversary could cause, but nonetheless the above properties are also defendable when an adversary is not present in the environment. Assuming specifically a hostile environment, several attacks that can be carried out by the adversary become apparent:

- *disturb communication*: adversaries that dispose of adequate hardware can interfere with the wireless communication, for example totally jamming the range their hardware reaches and rendering the nodes with that range almost completely useless.

[7]Note: this does not need to be true for all storage paradigms.

- *impersonation*: adversaries could try to access the network as if they were legitimate users, for example by using stolen identification or similar, but also could imitate particular nodes found in the network in order to attack functionality of algorithms.
- *takeover of nodes*: an adversary can physically modify a node, for example uploading a new piece of software onto a node causing it to log all data passing through or executing any other form of malicious code.
- *capturing nodes*: adversaries with physical access to a sensor network can collect nodes and extract data that is stored within them, including gathered data, security keys or identifiers. Similarly, the software code that runs on the nodes can be extracted and analyzed in order to exploit security holes that can be used for Denial Of Service or remote takeover attacks.

Especially the last two mentioned attacks will be of interest in this work and will be described in more detail in one of the following sections. Nonetheless, it is helpful to keep the other mentioned possibilities of adversaries in mind as well as attacks from other fields besides sensor networks, as adversaries are smart and inventive, even in restrictive environments surprises are possible: such as an adversary using a technique from a completely other area in order to gain advantages. The formal model that should be introduced, analogously to the (to be presented) sensor network model, will mostly focus on traffic-analysis strengths of an adversary, but also distinguish between adversaries that can capture nodes and ones that cannot. Those models are introduced later in this chapter.

Now in order to round up the properties that were described above to provide a more elaborate feeling of what has to be remembered when dealing with algorithms for sensor networks, especially those that are concerned with security issues, several problems that are of interest in the context of sensor networks and are still subject active research should be listed. Those listed below are just an excerpt (a detailed description can be found in [9]), but should point out to the reader where sensor networks' demand for smart and efficient solutions lies. It is also noteworthy, that solutions known from networks, that do not suffer from the same severe restrictions as sensor network and hence can rely, for instance, on strong cryptographic primitives, cannot be adopted thoughtlessly for sensor networks.

- *key establishment and trust setup*: In most settings where key establishment is to be employed strong cryptographic primitives are available, mostly through use of public-key techniques, like the Diffie-Hellman key establishment. In sensor networks the need for large keys and intensive computations by public-key cryptographic renders their usage impossible, because of the restrictions of such networks. But there have been plenty of approaches using strictly symmetric cryptography methods having each certain advantages and drawbacks. Pre-distribution of symmetric keys or a single network wide secret key come into mind.
- *secrecy and authentication*: Here the key establishment is an integral part of authenticity and secrecy, but it is more limited, as the case of having only a single key for the entire network is not admissible. Furthermore, as there is no public-key cryptography, symmetric cryptography is the only way to go for authenticity which includes methods to distinguish all nodes from each other and to authenticate each node. Hashing is mostly excluded because of its computational demanding nature. Hence pre-distributed keys are the most promising approach, which also has to bear in mind that the sensor network may need to be extended after deployment.
- *robustness to denial of service*: An adversary might disrupt the operation of a sensor network by jamming the wireless communication of the network with a high-energy signal or by violating the Medium Access Control implemented by the network. Spread-spectrum techniques offer a natural starting point, but need to be cryptographically secured which is still not easy achievable.
- *secure routing*: Several secure routing protocols have been proposed, mostly making use of the available symmetric keys. In order to avoid malicious nodes that infiltrated the sensor network multipath routing

offers an acceptable solution. Hereby, data is send on several disjunct paths limiting the probability that a malicious nodes can disrupt the transfer. Such an increase in security, as in most cases, has to be paid with redundant data transmissions.

- *resilience to node capture*: In most cases it is assumed that an adversary can physically access nodes and that nodes cannot be equipped with a temper-proof case, therefore once a node is captured all contained data and all keys used by the node are revealed. In most cases, such a capture offers an adversary to explore the network without knowledge of the operator or other nodes. More importantly, for this work, is that data is revealed in case that the attacked node is a hot node. Techniques against such physical intervention are mostly Anti Traffic-Analysis/Camouflage concepts or the idea of Evasive Data Storage, introduced in this work. A more detailed description of this problem will be given in one of the following sections.
- *secure group management*: In many applications of sensor networks a certain cooperation among nodes is required, for instance in object tracking. Hence, secure management of such a group needs to be available to avoid malfunctioning of the network.
- *intrusion detection*: Very similar in nature to the node capture and secure routing scenarios, intrusion detection aims to monitor the network for anomalies, enabling to take appropriate countermeasures in case an intrusion is likely.
- *secure data aggregation*: This problem addresses the aggregation process itself, which results in data at a hot node. The aggregation or fusion of data has to be secured in order to avoid the introduction of false data or pinpointing the location of a hot node by an adversary.

2.1.2 FORMAL MODEL

Now that sensor networks have been described informally in detail, there is a need for a formal model that can be used to speak about sensor networks more mathematically and thus allow to evaluate the coming algorithms appropriately. Hence the approach will be to take some of the properties mentioned in the previous section and transform them into formal statements, resulting in the desired model. Naturally, a property of almost all models is to leave out several details or not to reflect the real world perfectly. This is necessary to get ride of the overwhelming details that the real world forces upon the analysis of approaches or algorithms. Therefore, this work studies its algorithms under idealistic assumptions that are intended to reflect the real world as good as possible.

This thesis regards a sensor network as a distributed system, where n many processes participate in. Each process, or node (as participants in the network will be referred to in later chapters), has several properties that are assumed to be true and intend to reflect properties of sensor networks as accurate as possible. In the same way, properties of the assumed communication primitive available to processes are formulated precisely. It is assumed that the reader is familiar with the language used, if not there are several introductions into the field of Distributed Systems and Algorithms (see [14], [15] and [16], for instance). Furthermore, there are several cases where more than one set of properties is introduced, which also entails several different evaluations of algorithms. The assumed configuration of the model will be explained and defended at the point the algorithm itself is evaluated, which is mostly applicable in scope of the adversary models that will be explained in a later section.

To begin, the most basic behavior that can be found in sensor networks, the properties of sensor nodes should be stated formally. Those properties form the failure part of the model, that describes what bad situations can arise in the network or, from another point of view, states what algorithms have to cope with when operating in sensor networks. Nodes are taken to be processes that can execute code, store data and communicate with other nodes in the network. Sensing abilities are not modeled, they are assumed to provide data properly and the thesis will only concentrate on data items where the way they were generated is neglected. One inherent property of sensor nodes is that nodes can fail frequently in the network. Hence due to the nature of sensor networks several failures will be introduced:

- **partial node failure**: This failure type assumes that a node can fail to receive or send a subset of the message it is supposed to.
- **node failure**: This failure assumes that a node completely stops working, i.e. does not execute any code, does not exchange any messages and thus will never participate in the sensor network again.
- **malicious node failure**: This failure type adds to the model the presence of an adversary that somehow can influence the nodes. This type is described in detail once the respective adversary models are introduced towards the end of this chapter.

The *partial node failure* is applicable in sensor networks because of the environmental forces that can have influence on successful delivery of messages, which is quite distinguishable from the ordinary *node failure* which can be ascribed to possible destruction of nodes by the environment or to the exhaustion of available energy. This thesis will mostly focus on the second type of failure (node failure) with some comments on the malicious failure type. The first type, partial node failure, will be incorporated into the system model or as a part of the malicious node failure type. To precisely specify the model, the term **correct node** will be used: this is simply a node that does not fail (according to one of the specified failure models).

Now that the failure part of the formal model of sensor networks is stated, the other part of the model that needs to be defined precisely deals with the communication primitive. Sensor nodes can use their wireless communication primitive in different ways. The first one is simple communication with the neighbors of a node that find themselves in the range of the node's communication device. This can be seen as a **local broadcast**. The next stage is to send **point-to-point** messages to other nodes in the network (including base stations), which is realized by forming a path of repeated local broadcast invocations that proceeds through the network until the destination of the point-to-point communication is reached. Furthermore, the other primitive is **global broadcast** that a node can use to send a message to all nodes in the network. Although those primitives deal with different communication partners, the properties all three are assumed to satisfy are similar. As a model has the intention to reflect the actual properties of the entity that is being modeled, the communication primitive adds a common observation made in sensor networks: the message a node locally broadcasts to its neighboring nodes, is not always received by all those nodes. Moreover, the set of receiving nodes can vary, in the sense that different subsets of the vicinity receive a sent message when it is repeated. Thus in order to include that intrinsic property into the model that will be used for this thesis a probability is included that specifies the success of a local broadcast. This probability is thought to account for the possibility that a node does not receive a message although being part of the neighborhood of the sending node. Of course, this deficiency propagates from local broadcast to both the point-to-point and the global broadcast primitives. The second important restriction is that a broadcast is not instantaneous, therefore the time delay that is exhibited between sending and receiving should be taken into account as well.

In order to state the communication properties precisely some notation is needed: the node that is executing the communication primitive is referred to as s. Nodes that are in the range of the communication device of s are members of the set $V(s)$, whose elements are mostly denoted as $v, v', v'', \ldots$ or $v_1, v_2, v_3, \ldots$. In situations where a node has the function of a distinguished destination node, this node will be denoted as d or dst. Other arbitrary nodes in the network are denoted as $a, a', a'', \ldots$ or $a_1, a_2, a_3, \ldots$. The data that is to be send my the communication primitives is denoted as msg or, in case several messages are considered, $msg, msg', msg'', \ldots$ or $msg, msg_1, msg_2 \ldots$. If not stated otherwise nodes or messages referred to by different variables can be assumed to be different. In order to include the time delay mentioned in the explanations above, a linear, real timeline is used and moments on this timeline can be referred to by variables $t, t', t'' \ldots$ or $t_1, t_2, t_3, \ldots$. There is no need to explicitly define a starting point t_0 of the timeline as the important issues will be addressing relative time spans not absolute time passed. In several circumstances, there is a need to refer to a time interval, which will be denoted at (t, t') and includes all real points of time t'', such that $t \leq t'' \leq t'$. With those clarification, the formal properties of the three communication primitives can be stated:

local broadcast:

- *Probabilistic Validity*: If sending node s and arbitrary node $v \in V(s)$ are correct, then any msg send by s at time t will be received by v at time t' such that $t \leq t' \leq t + \Delta t_{lb}$, with a certain probability p_{lb}.
- *No Duplication*: No message is received more than once.
- *No Creation*: No message is received unless it was sent.
- *Order*: If a correct node s sends message msg before msg', then there is no correct node $v \in V(s)$ that receives msg' before msg.
- *Agreement*: If a correct node s sends a message msg at time t and a correct node $v \in V(s)$ receives msg at time $t' \in (t, t+\epsilon)$, then for every correct $v' \in V(s)$ holds: if v' receives msg' from s in time interval $(t, t+\epsilon)$ then $msg' = msg$.

The local broadcast is a communication primitive that allows s to communicate with *all* nodes in its vicinity at once or, by using encryption s can address a single node only. However, all the other nodes will overhear that a transmission took place in case s sends encrypted data to a single node only. Properties above that are given in the perspective of node s and node v, of course, apply to all nodes in the vicinity. A special note needs to be given in context of the *Agreement* property: in order to have a clean definition it needs to be assumed that node s does not send several distinct messages in the interval $(t, t+\epsilon)$, which otherwise, of course, would not confirm with the intended meaning of the *Agreement* property and force those distinct messages to be the same. As the specification of local broadcast indicates, a node s that wants to send a message msg to a node in its vicinity, for instance when trying to forward msg to the node closest to msg's destination, will reach the node after a finite time delay Δt_{lb}, if the transmission was successful, which is influenced by the probability p_{lb}. In order to overcome the deficiency of not reaching a certain node, s can repeat the local broadcast. A short look into the properties of this repetition shall be taken. To do so, let X be a random variable that defines the number of attempts necessary to have a successful transmission of msg to a specific node in the vicinity. X is geometrically distributed, thus the probability $P(X = l)$ for needing $l \in \mathbb{N}_0$ many repetitions till successful transmission is given by: $P(X = l) = (1 - p_{lb})^l p_{lb}$. Furthermore, the expectation of X is $E(X) = \frac{1}{p_{lb}}$, meaning that on average $\frac{1}{p_{lb}}$ many repetitions are needed to have msg be transmitted to the chosen node. The more interesting probability is the probability that a msg is transmitted successfully by s after l many repetitions, which is given by $p_{lb}(l) = 1 - (1 - p_{lb})^{l+1}$. To get a feeling of how fast this probability increases, concrete values can be plugged in: let p_{lb}, the probability that s can transmit msg successfully without any repetitions, be equal to $p_{lb} = 0.8$. After one repetition $(l = 1)$ the probability increases to $p_{lb}(1) = 1 - (1 - 0.8)^2 = 0.96$, after the second repetition the probability reaches $p_{lb}(2) = 1 - (1 - 0.8)^3 = 0.992$. On average the number of needed retransmissions for $p_{lb} = 0.8$ is $E(X) = \frac{1}{0.8} = 1.25$. Hence, by incorporating retransmissions the probability can be increased. This will be important for the higher level communication primitives. Now that the resulting properties of the **local broadcast** specification given above have been illustrated, a comment on the upcoming **point-to-point** primitive is necessary. A message msg sent between a source node s and a destination node d needs to pass through several intermediate nodes each employing local broadcast to forward msg. Thus the probability p_{pp} of successful point-to-point forwarding of msg from s to d depends on p_{lb}. Denoting the number of intermediate nodes or *hops* as l, the relation of those primitive probabilities is given by: $p_{pp} = p_{lb}^l$. This is obvious, because each intermediate node or hop has only a success probability of p_{lb} to forward msg, which over the course of l many hops results in the given probability p_{pp}. This probability can drop fast even with high values for p_{lb}, thus in order to guarantee a high p_{pp} retransmissions at the local broadcast level become a valuable tool to shift the odds in favor of successful transmission. With this in mind, and knowing that acknowledgments can be incorporated into the transmission process as well, p_{pp} can be made arbitrary close to 1 at the cost of repetition or equivalently higher energy consumption. With the relationship between p_{pp} and p_{lb} in mind the formal properties of the point-to-point primitive can be given as follows:

point-to-point:

- *Probabilistic Validity*: If sending node s and destination node d are correct, then any msg send by s to d at time t will be received by d at time t' such that $t \le t' \le t + \Delta t_{pp}$, with a certain probability p_{pp}.
- *No Duplication*: No message is received more than once.
- *No Creation*: No message is received unless it was sent.
- *Order*: If a correct node s sends message msg before msg' to d, then d does not receive msg' before msg.

Looking at the point-to-point properties above, a similarity to the local broadcast properties is evident, however the communication partner now is a fixed node d, not an arbitrary vicinity node as in the local broadcast case. The probability p_{pp} has a dependability on the p_{lb} probability and can be made arbitrary close to 1 through retransmissions, this has been discussed previously in detail. Analogously, the time statements of the *Probabilistic Validity* have to exhibit a similar connection. Assuming that between the source node s and the destination node d the number of hops is described by l, then each of those intermediate local broadcasts will involve a time delay of Δt_{lb} for the transmission. Thus the overall minimal time delay for the point-to-point message transmission from s to d will amount to $l \cdot \Delta t_{lb}$, which of course does not account for the repetitions that might be needed due to the unreliable transmission at the local broadcast. Therefore, the Δt_{pp} time delay stated in the properties above has a value greater or equal to $l \cdot \Delta t_{lb}$.

Similarly to the relationship between local broadcast and point-to-point described above, a relation is also given between global broadcast and the two preceding primitives. Nonetheless, a more detailed investigation of the relationship of the used probability p_{gb} is omitted and the specification should be sufficient for the needs of this thesis:

global broadcast:

- *Probabilistic Validity*: If sending node s and arbitrary node $a \in \Pi$ are correct, then any msg send by s at time t will be received by a at time t' such that $t \le t' \le t + \Delta t_{gb}$, with a certain probability p_{gb}.
- *No Duplication*: No message is received more than once.
- *No Creation*: No message is received unless it was sent.
- *Order*: If a correct node s sends message msg before msg', then there is no correct node $a \in \Pi$ that receives msg' before msg.
- *Agreement*: If a correct node s sends a message msg at time t and a correct node $a \in \Pi$ receives msg at time $t' \in (t, t + \epsilon)$, then for every correct node $a' \in \Pi$ receives msg' from s in time interval $(t, t + \epsilon)$.

Note that the properties stated are assumed to be true for correct processes or correct communication partners. Naturally, the *Agreement* property is not applicable when assuming point-to-point communication, as there is only one recipient in such a case. Having seen what is assumed to be granted in the model, it is clear that the thesis will not be particularly concerned with low level issues as access control to the wireless medium, collision detection or other matters that might come to mind.

The other important part of the model is that of time. In addition to the global real time, which is more important for the primitives, each node in the system provides a local clock. It is assumed that the time drift between any two local clocks in the system is constant, which is assured through the presence of some distributed synchronization algorithm (see [18] for a synchronization algorithm for sensor networks). Furthermore, each node is assumed to have certain storage capabilities, however a specific interface to adequate primitives is not given, but just assumed to exist and work properly. There will be no emphasis on how to deal with data errors, loss of data or similar issues that storage primitives of a single sensor node might exhibit. The basic assumption therefore is a perfect storage primitive that stores and retrieves data without failure, except when the node itself fails. In a similar sense, the computing abilities that each sensor node disposes of are also assumed to work properly. Moreover, for most analysis the computation time of arithmetic or comparable tasks is assumed to be finite or even neglected and assumed to be instant. This assumption is chosen to shift the attention from issues that are of minor concern to the actual security related objectives this thesis and the presented algorithms aim to provide.

It is a natural property of a model that the weaker the assumptions are the more difficult it gets to write algorithms that work, especially in presence of failures. On the other hand, stronger assumptions lead the way to easier algorithms as there is less to be worried about. Thus, for instance, taking strong communication primitives for granted leads to simple algorithms, but at the same time the possibility of practical realization can be questioned. The algorithms will state precisely which model they are assumed to be running in (what failure, what communication primitive, etc.). Furthermore, the section concerned with the notation of the algorithms each primitives will be incorporated into the syntax, such that a clear reference can be taken when analysis is conducted.

2.1.3 Other Sensor Network Assumptions

The assumptions presented in the preceding section are concerned with the formal model that will be used to analyze algorithms formally. Nonetheless, there are other assumptions that need to be made, especially concerning other primitives that are available for the algorithms to use. Also practical assumptions about communication that are not covered in the formal model need to be stated as well. This section thus details about assumptions that are rather out of the scope of the formal model needed to analyze algorithms, but are employed in the practical execution.

The previously described formal communication primitive is supposed to be realized by a routing scheme for sensor networks that also supports security primitives. A detailed description of such a secure routing scheme for sensor network can be found in [4], which is basically using simple symmetric encryption based on pre-distributed encryption keys. The paper also describes extensions that allow for secure point-to-point communication between arbitrary nodes in the network. Furthermore, it is assumed that nodes dispose over keys that can be used for additional purposes, like secure group communication, those encryption keys can either be pre-distributed, generated from sensor readings or similar. Hence, as a result such a scheme is used to ensure so called *inside security* of the network, which intends to protect the data transmitted through the network (see [3] for more details). The dual notion is that of *outside security*, which is concerned with secure communication of the legitimate user and the sensor network. The outside security is important when users query the network for data, such that the transmission cannot be eavesdropped on. For this work is not concerned with that kind of security issues it is assumed to be secure and will not be of major concern in the following.

Additionally to adding security to routing, it is assumed that each node has knowledge about its location and therefore can estimate how close it is to a given location. Furthermore, nodes can exchange their position and thus are aware of the position of the nodes in their vicinity. The assumed routing scheme takes advantage of this setup and does not deliver messages according to addresses but according to coordinates that specify the location of the destination node. For a detailed description of such a routing scheme see [19] or [5]. This wraps up the assumptions made about routing in the sensor network.

A tool that is extensively used in the algorithms that will be presented in this work is that of a (pseudo) random number generator(RNG). It is assumed that such a generator is available to each node in the network. Several ways to obtain random bits have been researched. The work on secure routing, see [4], describes that random bits can be generated from a cryptographic key or the usage of sensor readings as source for random bits or as random seed for a RNG. A pseudo random number generator that is very sensitive to the resource constraints found in sensor networks is presented in [17]. The approaches based on keys of some sort have to be regarded as dangerous in several cases, as such keys could be compromised and lead to a computability of random bits by other parties than the node itself.

This finishes the assumptions that are used throughout the thesis, but should be subject to change in future work, where weaker assumptions could prove to be sufficient for the purposes of the algorithms investigated hereafter.

2.2 Data Storage

The preceding sections introduced the basic formal model and assumptions that are made on available technologies in sensor networks. This section is supposed to introduce the reader to the field of data storage in sensor networks that is especially important when the notion of Evasive Data Storage is introduced later in the thesis. Before diving into several approaches how data can be stored in a sensor network, it is important to stress how this work will look at data storage. It is most significant to the network to gather data and to provide this data to a legitimate user asking for it. Depending on the application of the sensor network the kind of data varies: a network concerned with monitoring wildlife might want to store each sighting of an animal separately or try to store only the current location where a particular animal is located, thus updating a particular data item each time an animal moves. On the other hand, a network that monitors temperature needs to continously update data items in case changes of temperature are measured by some of the sensor network's node. Furthermore, in order to save communication messages or space for storing gathered data there are several notions that try to aggregate data before storing it in the network. In the case of wildlife monitoring this can be imagined as the process of sending only one message containing the (aggregated) information of a sighting, instead of having each node that sensed the sighting sending its own message. Thus aggregation helps to minimize messages sent by trying to avoid repeating the same information. As it can be seen, there are different properties of how to look at data of a sensor network and its respective needs for updates. As not every single aspect can be addressed, an assumption is made of how data is regarded from now on.

Data is either created directly from a node that sensed some event or is generated by a node that aggregated data from several other nodes. The data is packed into a *data item* D, which along with the application depended raw data contains following information:

- *generating node identifier*: a node identifier that can be used to trace data to the node that sensed or aggregated the data contained in the data item.
- *timestamp*: this simply describes when the data item was generated and can be imagined to be a field containing the local time of the generating node at the time it created the data item.
- *type identifier*: this is an identifier for the data itself, such that a specific query can be matched with the data item. For instance, in case a legitimate user wants to query animal sightings instead of the temperature, he can use the type identifier to distinguish between data items of the specific category.

Thus besides the fact that data items need to be stored, updated, retrieved and need to be protected from malicious users, no other aspect, like data aggregation techniques or similar, will be of concern.

As mentioned already, a data storage approach needs to provide means to allow a legitimate user to access the data that was gathered by the network. This entails that the data storage algorithm needs to exhibit qualities that guarantee basic properties concerning the provided data. A user surely wants to obtain data that reflects the latest state of the environment and not obtain data that due to some insufficiency in the algorithm is old and thus rather useless. If no data items matching the queried data are available, a default value $\square$ should be return to signal that no data of the requested type has been stored yet. Summarizing, the data provided needs to satisfy freshness. Similarly, the user wants to obtain data that is *correct*, meaning that it is the data the user queried for, i.e. matching the wanted type $Type$, and that the data has not been altered or damaged in the query or storage process. There are several other properties that are stated below in a formal fashion. Furthermore, an important issue for data storage is that data should be protected from malicious users. This property is the major interest of this thesis and hence deserves mentioning in the section introducing data storage. More on the issue of security of data is provided in the chapter that is concerned with Evasive Data Storage. Hence for a data storage algorithm to work properly, it needs to satisfy the following properties, which are short descriptions of the desirable functionality stated above:

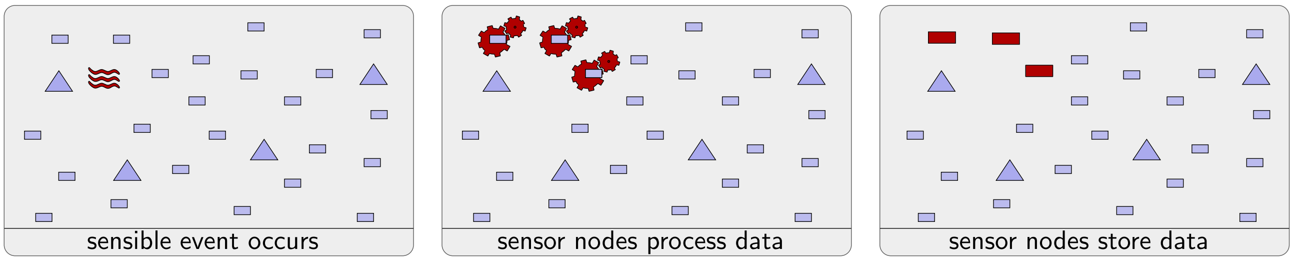


Figure 2.3: Local Storage in Sensor Networks

- **Availability:** If a data item of type $Type$ is requested by a legitimate user at time t, then the data storage algorithm will return some data item D at time $t + \Delta t_{da}$ with probability p_{da} or return $\square$ in case no matching data is available.
- **Correctness:** If data is provided, then it is *correct*.
- **Freshness:** Let Ts_f be the timestamp of the latest data item successfully stored[8] and matching $Type$ at time t of the query, then the data item D provided satisfies $D.Ts \geq Ts_f$ with probability p_{fr}.

In order to satisfy those properties the algorithm needs to address failures that might occur, especially the node failure introduced in the formal model is important, but also the malicious node failure that introduces an adversary into the picture is essential when thinking of securing data in a network that is exposed to a hostile environment. However, an additional design criterion that must not be neglected is the sensibility for energy, resulting in an aim for limiting computations and communications. Having an idea what data storage will refer to in the scope of this thesis, especially when the notion of Evasive Data Storage is investigated, it is worthwhile to take a look at how data storage currently is approached in sensor networks. The following sections introduce the most common paradigms and end in a short comparison based on message complexity. The approaches will not be evaluated thoroughly, but should be thought of as a guideline of what methods of storage are currently available.

2.2.1 Local Storage

One of the simplest storage methods is Local Storage and is depicted in figure 2.3. This method keeps the data from sensor readings local to the sensor node, where the data was originally sensed. The figure describes that process in three scenes, the left one shows that a sensible event occurs somewhere in the network. This is registered by the sensor nodes and the corresponding data is processed (middle picture). The right picture shows where the data is stored and, as depicted, the same nodes that sensed the data also store the data. This happens without notifying any other nodes that data has been processed, thus incurring no communication costs[9] at all. Thus this method has the property that sensor readings are not inclined to cause communication overhead at the time an event occurs. Whereas both other approaches, which will be presented next, cause messages to be transmitted immediately (or shortly) after data is sensed. Such behavior that blessed the algorithm with no communication overhead for sensing events, entails that once a legitimate user needs to obtain the data, substantial overhead occurs as there is no information available as to where data is stored. All sensor nodes that gathered data silently stored it locally, hence the user needs to ask each node or flood the complete network to get the data needed. Such a flooding involves significantly more messages than the direct sending of data as seen for example in External Storage (see below). To put this into more concrete terms the *event costs* (i.e.

[8]In terms of the used Storage Mechanism, which basically means that the data item is stored and ready to be retrieved.
[9]The communication costs refer to the number of messages that have to be send.

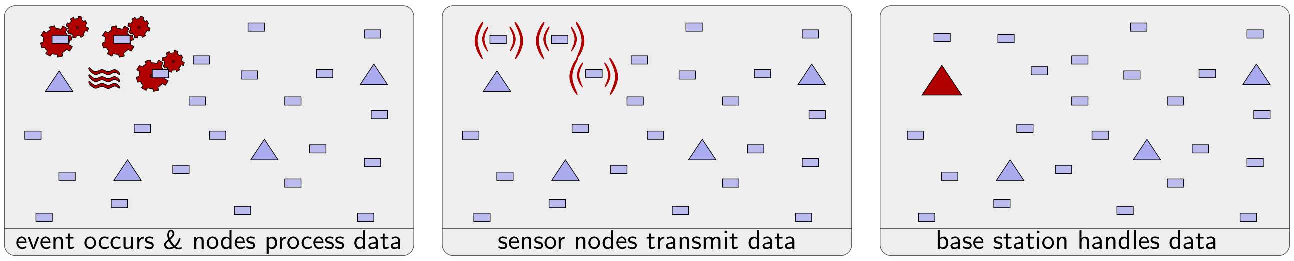

event occurs & nodes process data
sensor nodes transmit data
base station handles data

Figure 2.4: External Storage in Sensor Networks

the number of messages that have to be sent at the time the event occurs) are zero, local storage makes nodes silently store sensed data locally. The other important costs are the *query costs* (i.e. the number of messages needed to successfully provide data that a legitimate user asks for). Those costs are defined by the need to ask every node in the network, if it does store any data items matching the query, thus the resulting query costs are of magnitude $\mathcal{O}(n)$, where n is the number of nodes in the network.

2.2.2 External Storage

Another storage method that is applicable in sensor networks, is External Storage. The figure 2.4 illustrates the workings of this method. The left image shows nodes that sense and process an event that occurs within their reach, and directly forward the data items to a base station (middle picture). The base station has two options either to store the data at a location completely outside of the sensor network or to keep the data locally in its own storage. For the former option, the base station needs to function as an access point with uplink capabilities to transmit the data to some storage facility, this could be, for instance, an Internet/Satellite uplink connecting with a database of the maintainer of the network. Thus in case of external storage the event costs are of magnitude $\mathcal{O}(\sqrt{n})$, as each node that sensed and gathered data needs to send it to the base station incurring the stated number of messages for routing the data items through the network. [10] Now that the principle of storing is clear, the question arises what happens when data is requested/queried: As the data items have already been moved by the generating nodes to a base station, the data is instantly available to a legitimate user that queries the base station for it or an external data base where the data was sent to by an uplink. Thus there are no costs for querying data in external storage. However, such zero query costs can only be assumed if the legitimate user asks the base station that stored the data directly, in the case that the user queries from other parts of the network a query cost of $\mathcal{O}(\sqrt{n})$, needs to be assumed.

2.2.3 Data Centric Storage

Data Centric Storage is an approach that stores data at some location in the sensor network, which is determined by the data itself. This is realized with a global mapping (or more precisely a Distributed Hash Table (DHT)) that maps data to coordinates that specify a storage node or a group of storage nodes. As long as this map is the same at all nodes an interesting storing mechanism is provided. Once a node gathers data, it can use its copy of the hash table to get the coordinates of the node that should store the data (see figure 2.5, left picture), and begin forwarding packets to that node towards that storage destination (see figure 2.5, middle picture). Hence, data is neither stored at the node that sensed an event nor at a base station, but at some node that is determined by the DHT and the data's type identifier. Similarly, a legitimate user that longs for a specific data item can determine the node that is responsible for storing the data with the help of the hash table and needs

[10]The $\mathcal{O}(\sqrt{n})$ bound for point-to-point is based on the assumption that the network is uniformly disperse and that the diameter of the sensor network is $\mathcal{O}(\sqrt{n})$ (see [5] for more details).

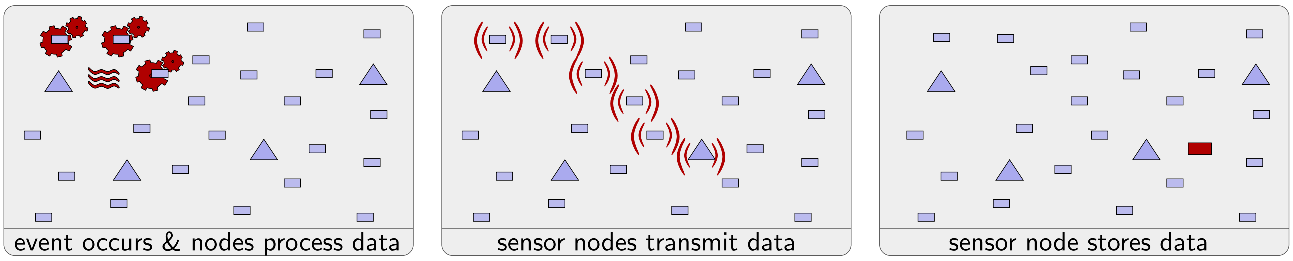


Figure 2.5: Data Centric Storage in Sensor Networks

only to query the node pointed to by the mapping. Both operations involve rather low amounts of messages transmission and are surely by far more efficient than flooding the network as required by the Local Storage method. The event costs involve sending the data item to the storage node, thus involving $\mathcal{O}(\sqrt{n})$ messages. Similarly, a querying user knows which node is responsible for storing the wanted data and hence incurs costs of $\mathcal{O}(\sqrt{n})$ to obtain the data from that node.

To understand how Data Centric Storage works in detail, first it is crucial to grasp the idea of a Global Hash Table (the Data Centric Storage approach presented here was introduced in [5] and [6] respectively). Every node needs to be equipped with sufficient information to compute the hash value (i.e. the location of the designated storage node for each type of data), this can be realized before deployment of the sensor network or during setup[11] along with several other administrative tasks performed at that time. The hash table transforms a type identifier into the coordinate of a node that is supposed to store data of the given type. Once the coordinate is computed, the node starts forwarding the data towards this coordinate. There is no possibility to acquire the identification of the node that will store the data, as the coordinate is not a representation of such an identification, it merely gives a hint as to where approximately the data has to be forwarded to, but not what node exactly will be responsible for the data. This implies that nodes need to have a sufficiently accurate understanding of a coordinate system to forward data according to the coordinate information returned by the hash function, as well as some mechanism to finally decide what node will store the data. Hence, the process starts with a data item and a corresponding coordinate of the destination. The node s currently holding [12] the piece of information also knows its own position and the positions of its neighbors $v \in V(s)$. Knowing the coordinate of the destination and the coordinates of the neighbors s can decide which node to forward the data to. Data Centric Storage makes use of the *Greedy Forwarding* method, which, as the name implies, makes s choose the node $v \in V(s)$ in its vicinity whose position is the closest to the destination. In the following step, this closest node v repeats the same greedy choice and forwards the data to the next node which is closest to the destination in v's vicinity. This repetitive cycle therefore forwards the data gradually to the final destination, always trying to choose the node that is closest to the destinations coordinates. Figure[13] 2.6 (left picture) shows an illustration of a node s (labeled as source) holding data to be forwarded, its vicinity, the closest node to the destination (labeled as closest) and an excerpt of the network including the destination. As in most greedy approaches, this method has the desirable property of being simple and in most regards sufficiently effective. But it suffers from a problem that is illustrated in figure 2.6 (middle picture), where none of the nodes in the vicinity is closer to the destination than the current node s. This is referred to as the *Void Problem*. As the greedy method needs to forward the data to the closest node, it would command the current node to forward the data to itself as it is the closest node, resulting in a dead end for the forwarding and storing process. To avoid such a deadlock behavior, a node, recognizing that all its neighboring nodes are father away from the

[11] Referring to a network's first tasks performed after deployment to start working, including set up of additional cryptographic keys, or routing information for parent base station assignment, etc.
[12] The terms current node, source node or s are used interchangeably in the following description.
[13] Adapted from the original figures found in [5].

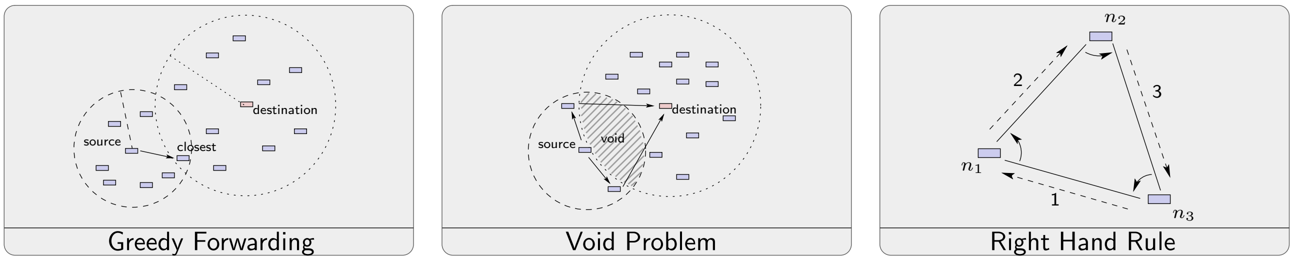


Figure 2.6: Data Centric Storage Ingredients

destination than it is itself, switches from Greedy Forwarding to so called *Perimeter Mode*, which causes packets to be forwarded by the *Right Hand Rule* (see figure 2.6 right picture). The Right Hand rule causes the packets to move around the region that is responsible for the Void Problem and thus eventually runs into a node that can effectively forward to the destination. For more details on Data Centric Storage see [5], where aspects like replication are addressed, which are not needed for the purpose of this brief introduction given here.

For a quantitative comparison of the three storage methods seen so far consult Table 2.2, where E is assumed to be the total number of events, i.e. number of sensor readings that resulted in data being stored, and Q is taken to be the total number of queries by the legitimate user.

Storage Method	**Event Costs**	**Query Costs**	**Total**
External Storage	$\mathcal{O}(\sqrt{n})$	–	$\mathcal{O}(E \cdot \sqrt{n})$
Local Storage	–	$\mathcal{O}(n)$	$\mathcal{O}(Q \cdot n)$
Data Centric Storage	$\mathcal{O}(E \cdot \sqrt{n})$	$\mathcal{O}(Q \cdot \sqrt{n})$	$\mathcal{O}(Q \cdot \sqrt{n} + E \cdot \sqrt{n})$

Table 2.2: Storage Communication Costs

Out of the three storage methods, the most interesting one is the Data Centric approach. Local Storage is clearly quite inefficient considering the flooding entailed by a user querying data and External Storage is restrictive in the applications that can be addressed with a sensor network, as base stations could not be present in the network. A more convincing argument is that External Storage causes all data to be sent to base stations, not only making them a bottle neck, but also jeopardizing all the gathered data in case a base station is damaged or found by an adversary. Data Centric Storage has the property of integrating nicely into the concept of hot nodes, caused by the fact that the mapping involved in determining where data should be forwarded and stored at, can be implemented to point exclusively to fixed h hot nodes. Now that an overview of data storage approaches in sensor networks has been given and sensor networks have been introduced, the next section can present an important problem that found in sensor networks and is inherently related to the data stored in the network.

2.3 The Node Capture Problem

The preceding sections gave an introduction to the field of sensor networks and revealed gathering data as the most important task of such networks. Furthermore, in order to cope with the need to make that data available, different notions of data storage in such networks were explored. Now the picture showed so far contains a network and storage algorithms that would suffice, if it were not for the possible deployment of sensor networks in hostile environments. Such hostile environments introduce, as it has been mentioned before, a malicious entity, the adversary, into the area where the network's components are located.

Before investigating possible actions of an adversary that has physical access to the nodes of a network, one of the fundamental assumptions that encourages those actions is explained. Due to the vast number of nodes that are assumed to constitute the network, the sensor nodes cannot be assumed to be protected in any sophisticated way, i.e. sensor nodes are assumed not to be tamper proof. The persuasive reason is that tamper proofness would incur a significant rise in cost of each single node, which quickly adds up in case of hundreds or thousands of nodes. Furthermore, smart adversaries always find a way to overcome even the most sophisticated protection. Hence, altogether assuming that sensor nodes are not physically protected from adversaries is inevitable.

Depending on the objectives and the abilities of the adversary several different attacks can be thought of. An adversary that accesses nodes can read out data contained in the node, which can include gathered data that was stored in the network or cryptographic keys that the node makes use of. The former is naturally a most undesirable thing to happen, as the gathered and stored data might be confidential. Letting a malicious party access the data or even manipulate data can be disastrous. An even more uncomfortable scenario is that of an adversary overtaking a node, meaning that the node performs arbitrary actions, like sending messages or actively trying to keep algorithms from working properly. Such an adversary that controls one or several nodes in a sensor network can cause significant damage, he can monitor data passing through his malicious nodes or start Denial of Service attacks against the network. It becomes evident that this problem, referred to as *Node Capture Problem*, calls for solutions.

This thesis especially addresses the protection of gathered and stored data. Thus it needs to be clarified what possibilities an adversary has in order to achieve his goal of finding interesting nodes or hot nodes. Two fundamental viewpoints are taken, the first is that an adversary can *find* interesting nodes and the second is that an adversary can *remember* interesting nodes:

- *Finding* interesting nodes is grounded in the notion that an adversary can take advantage of traffic analysis in order to identify nodes that might store data. In the case of external storage, for instance, message will be frequently send through the network towards base stations, which an adversary monitoring the network could observe and thus be lead to a base station. Similarly, in the case of data centric storage, nodes that sense data of a certain type will forward messages containing their readings to the node responsible for storing that particular type. An adversary can follow the path that such messages take through the network and hence eventually find the corresponding hot node. Even the observation of the network at the time a legitimate user queries the network for data can provide insights as to where interesting data can be found. Usage of traffic analysis is not only limited to tracing messages, other communication patterns can emerge that an adversary can observe and consequently deduce if data is involved or where data might be found.

- *Remembering* interesting nodes is grounded in the idea that an adversary, once he finds hot nodes, can access them over and over again. As in the case of data centric storage the node that is responsible for storing the data will remain the same over the lifetime of the network. An imaginable scenario is that sensor networks might offer data based on a subscription service, where users can temporarily subscribe to the network, access its data and then unsubscribe. Naturally, in the storage approaches seen in the preceding section such a temporal legitimate user can remember where his queries are answered from, unsubscribe and then access updated data as frequently as wanted, without paying for a subscription service.

Having stated those two possibilities and the objective of this work to address the problem of securing data in sensor networks, the goal is obviously to hinder an adversary from identifying hot nodes in the network by providing a storage mechanism along with camouflage techniques that substantially reduce his chances of being successful. In order to investigate solutions, the adversary needs to be specified precisely, as he can enter the network with varying abilities and thus calling for different approaches. The next section provides a detailed description of the different possible adversaries that can be encountered in a hostile environment and which will be used for the rest of the thesis.

2.4 Adversary Models for Sensor Networks

The previous section showed that an adversary can pose a threat to the data stored in the sensor network and to the nodes themselves. Thus it is important for the algorithms that are presented next to specify what kind of adversaries the hostile environment can exhibit and hence allow to an appropriate evaluation of algorithms' performance. This is done by specifying the possible adversaries and explaining the Malicious Node Failure that was deferred to this section.

2.4.1 Adversary Models

The Node Capture Problem, explained previously, gives a good intuition of how to model the abilities of adversaries. Adversaries are ascribed two basic abilities that will influence the model that will be used for adversaries trying to exploit sensor networks' weaknesses. Those two abilities are characterized as follows:

- **Traffic Analysis**: the ability to analyze the network's traffic patterns and to deduce information about the behavior of algorithms, especially where data is stored. This ability is important when thinking of the objective of an adversary to find interesting data, where he naturally can only rely on communication behavior of nodes that he is able to observe.

- **Intervention**: defines to what extend an adversary is allowed to act on the network or more specifically on the network's nodes. This ability describes the physical impact an adversary can have on nodes, including their data and their behavior.

The Traffic Analysis and Intervention abilities/skills stated above are just rough dimensions that will be considered, but in the concrete models each of these abilities is specified more precisely. In order to do so, each rough ability needs to be specified in detail, yielding an order of different levels of power an adversary might dispose of in one of the two rough abilities given above.

First, Traffic Analysis will be investigated more thoroughly leading to the following levels, which constitute an order of different traffic analysis powers that an adversary might own and are given in order with the weakest level first:

- *blind*: this is the most restrictive level of traffic analysis, as an adversary matching this category cannot perform any analysis of the traffic whatsoever, and therefore cannot exploit traffic patterns of the network to reach his goal of find interesting nodes. Such an adversary can be imagined as not disposing of the necessary knowledge or devices to perform any analysis, yielding the weakest adversary in terms of finding hot nodes in the network.

- *local*: this level ascribes an adversary $\mathcal{A}$ the ability to analyze traffic in a locally restricted part of the sensor network. Furthermore, the adversary can change the excerpt of network he is able to analyze (by moving around). Thus a local adversary can analyze only a fragment of the network's traffic at a time, but vary the fragment over time. To visualize a local adversary, an entity disposing over some traffic analysis device, which has a restricted range compared to the area the sensor network occupies, should be thought of. Such an adversary can thus move around in the network and monitor traffic patterns of the nodes which are in the range of his device. Based on his observations, he can follow patterns, which might lead him to interesting nodes. Putting this into a more formal ability, adversary $\mathcal{A}$ can monitor nodes that are within his vicinity $V(\mathcal{A})$, where it can be assumed that $|V(\mathcal{A})| \ll n$ (the fragment the local adversary can analyze is only a fraction of the whole network).

- *global*: the most powerful level of traffic analysis an adversary can exhibit is that of a global adversary. Such an adversary can analyze the complete network, hence in formal terms this means that for an adversary $\mathcal{A}$, it holds true that $V(\mathcal{A}) = n$.

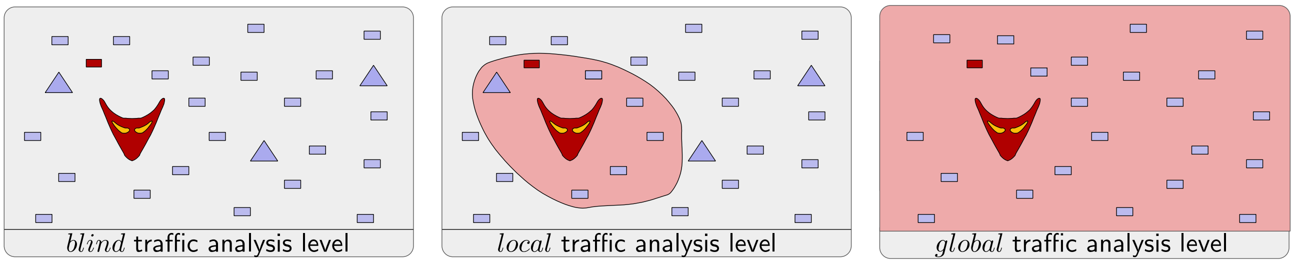

blind traffic analysis level
local traffic analysis level
global traffic analysis level

Figure 2.7: Adversary Traffic Analysis Order

Those ordered levels of ability are depicted in figure 2.7, where a reddish color in the background of nodes implies that the adversary can perform traffic analysis on the nodes contained within that marked region. The left image depicts a blind adversary, where no nodes at all can be monitored by him. The middle image shows a local adversary that can analyze an excerpt of the network. Moreover the picture shows a hot node that is located in the scope of the adversary, thus if the adversary analyses the traffic he might come to the conclusion the mentioned node is hot and therefore gain access to data. The right image shows the most powerful adversary that can monitor the complete network, such an adversary can follow patterns without physically moving in the network and hence quite easily identify hot nodes or identify other properties about nodes in certain locations.

Now that Traffic Analysis has been investigated and an ordered set of traffic analysis abilities has been given, the same needs to be done for the Intervention capabilities an adversary can be ascribed. There will be two categories: the *passive* category ascribes an adversary abilities related to data only, whereas the *active* category adds more physical or interactive abilities. The Intervention order is constituted of the following levels, starting from the weakest and proceeding towards the strongest:

- *limited passive*: an adversary of this level, can retrieve all information on a node, but in order to do so he needs to completely remove the node physically from the network.
- *passive*: a passive adversary can solely go and open up arbitrary sensor nodes to get the information currently stored in such a node, there is not need to leave the network to do so. Furthermore, such an adversary is assumed to be able to modify the data the node contains.
- *limited active*: this level of interaction allows, in addition to the actions of a passive adversary, to perform physical disturbances like *dislocation*, *jamming* or similar; however in the scope of this work only the disturbances listed below will be ascribed to this level of intervention.
- *active*: an active adversary, on the other hand, can not only access and modify data on sensor nodes of his choice, but can gain unrestraint control over them. Such overtaken nodes or malicious nodes act according to the will of the active adversary: sending arbitrary messages, monitoring data passing through or trying to disturb protocols.

The order above defines the different abilities of intervention that can be ascribed to an adversary, but still left the notion of a limited active adversary open to extensions. Nonetheless, this thesis will consider only the following actions possible for a limited active adversary:

- *jamming*: the adversary is able to hinder sensor nodes from accessing the communication medium within a certain region around his location. Hence, an adversary $\mathcal{A}$ making use of jamming causes temporal disruption of the message sending abilities of all nodes in $V(\mathcal{A})$. This action is assumed to be local in nature, as jamming the whole network (*global jamming*) leaves no space for analysis: the whole network would stop to function.

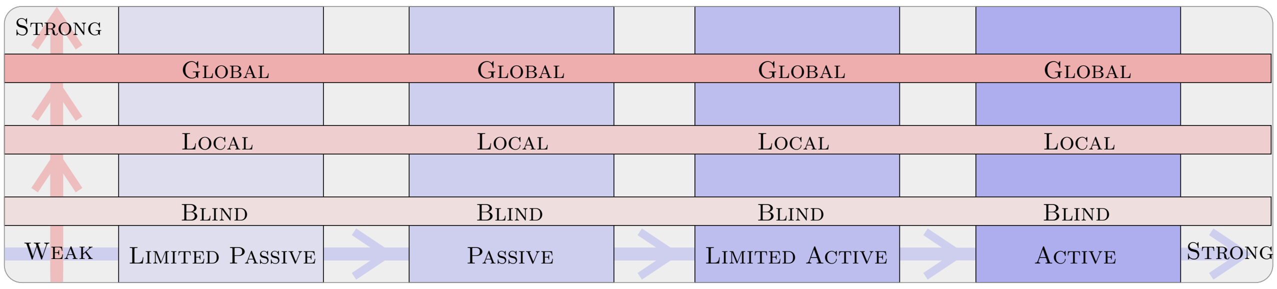


Figure 2.8: Adversary Model Lattice

- *dislocation*: this property states that the adversary has the capability of dislocating an arbitrary sensor node a. Doing so, the adversary causes that $V(a)$ to change completely. Practically, this action can be thought of as an adversary that picks up a node and throws it to another part of the sensor network. Similarly, natural topology changes can be seen to be included in the dislocation notion, but mostly in a more restrictive form, where the changes $V(a)$ has to cope with are minor: instead of assuming a change of the complete vicinity only a subset $D \subseteq V(a)$ is assumed to change completely.

Having defined ordered levels for both traffic analysis and intervention, those two abilities are now combined to form the complete adversary models that will be used in the following. An adversary model $\mathcal{A}$ is seen as a pair (Traffic Analysis, Intervention), where the first component specifies the level of traffic analysis the adversary model assumes and analogously the second component states the level of interaction. A fully specified adversary model thus looks like $\mathcal{A}(blind, active)$, in case of an adversary with blind traffic analysis abilities, but active interaction skills. The result is a partial order of adversary models. This partial order is depicted in the figure 2.8 as a lattice. It can be seen that the most powerful adversary is $\mathcal{A}(global, active)$ (in the upper right corner) and the weakest is $\mathcal{A}(blind, limited\ passive)$ (in the down left corner). As it is a partial order not all adversary models are comparable in the sense that one model is truly stronger than another. $\mathcal{A}(blind, active)$ and $\mathcal{A}(local, passive)$ are such a pair, as neither of them is stronger than the other. However some models can be put into relation, thus the symbols $\prec$ and $\preceq$ are used to refer to the partial order implied by the lattice in figure 2.8.

There are issues that only concern one ability, either traffic analysis or interaction, thus being ignorant about the other ability. For such cases, a $\star$ symbol can be inserted for notational convenience into the appropriate component of the model to stand of all different levels possible. For instance, wanting to talk about an adversary that has local traffic analysis skills and neglecting completely interaction, the adversary model $\mathcal{A}(local, \star)$ would be appropriate to refer to such an adversary. Another convention applied is how to state that an adversary falls into a particular model: a statement like adversary $\mathcal{A}(blind, active)$ implies that $\mathcal{A}$ is an adversary having the skills specified and alleviates the need for cumbersome descriptions like assume an adversary $\mathcal{A}$ that matches the adversary model $\mathcal{A}(blind, active)$. Of course, statements as $\mathcal{A}_b(blind, \star)$ or $\mathcal{A}_{all}(\star, \star)$ are also valid adversaries that can be referred to by $\mathcal{A}_b$ and $\mathcal{A}_{all}$ respectively.

The ordering of the levels of abilities yields the nice property that a statement ascribing an adversary of a low level (for example a $\mathcal{A}(blind, \star)$) to perform a malicious action, will automatically ascribe the same for adversaries of higher levels ($\mathcal{A}(local, \star)$ and $\mathcal{A}(global, \star)$). Similarly, it should be clear that an algorithm that can cope with a $\mathcal{A}(global, active)$ adversary will be able to sustain all other presented adversaries. However, as the lattice presents only a partial order, an algorithm coping with a $\mathcal{A}(local, limited\ active)$ adversary will not necessary work as well with a $\mathcal{A}(global, passive)$.

This finishes the basic adversary models and the discussion of the (notational) usage in the following chapters. However, before proceeding, an extension to the adversary models needs to be discussed, which is specifically tailored to the investigation of Data Security, hot nodes and the timing involved. Assuming an adversary $\mathcal{A}(ta, in)$ matching some of the models defined above, the **extended adversary model** $(\kappa, \Delta) - \mathcal{A}(ta, in)$ represents an adversary that

- has traffic analysis abilities at the $ta \in \{blind, local, global\}$ level and intervention abilities at the level specified by $in \in \{limited\ passive, passive, limited\ active, active\}$,
- can open up and access at most κ many nodes,
- can do so only within time interval specified by Δ.

Of course, assuming a $(\kappa, \Delta) - \mathcal{A}(\star, \star)$ entails certain basic properties. Let p_{as} be the probability that an adversary successfully accesses data, then as κ approaches $(n - h + 1)$ the probability p_{as} converges to 1. This is clear, because once $\mathcal{A}$ has opened up $(n - h)$ many nodes there are only hot nodes left, therefore any further access of a node not opened yet will lead $\mathcal{A}$ to obtaining data items.[14] The actual application of the adversary model within this thesis is in the scope of time-constraint algorithms, however this notion can also prove useful in other contexts beyond the techniques discussed herein.

2.4.2 Malicious Node Failure

Now that the possible adversaries that can be assumed in hostile environments are specified, those need to be put in context to the node failures that were specified in the the formal model of sensor networks. The formal model stated two possible failures that can occur in networks the **node failure** and the **malicious node failure**. The former is modeling the inherent properties of sensor networks, like energy exhaustion of nodes or environmental forces that can destroy nodes. This kind of failure is also assumed possible in case of malicious node failure, but adds other possible behavior based on the given adversary model.

Thus assuming a different model in malicious node failure results in a different range of additional failures that need to be accounted for, besides the simple notion of node failure. The additional failures are depending only on the interaction skills of the adversary model. The precise additional failures for each level of interaction are stated below, along with restrictions on the number of failures, if applicable. Each level, naturally, inherits the additional failures of the weaker levels.

- *limited passive*: such an adversary does not add additional failures. The malicious node failure is the same as the node failure.
- *passive*: given such an adversary and malicious node failure, the adversary is assumed to be able to modify up to t nodes to store arbitrary data.
- *limited active*: the additional failures include dislocation of a node a, specified as a change of $V(a)$ and jamming which can cause temporal send and receive omissions.
- *active*: an active adversary leads to arbitrary behavior of up to t nodes, where this number includes nodes that failed completely (as specified in by the notion of the node failure).

It is crucial to remember, that an adversary does not only add failures to the malicious node failure model, but also poses different threats, like data extraction, et cetera, that cannot be put as a failure of a node, but still need to be addressed and moreover are of most concern in this thesis.

[14]This assumes that the h hot nodes are all distinct and fixed.

2.5 Notation and Terminology

The last sections have given an introduction to the field of sensor networks from a data security perspective, where also assumptions on those networks and the adversaries that can be encountered were included. Those will be essential for the evaluation of the algorithms that will be presented in the following. Nonetheless, in order to understand the algorithms their notation needs to be specified, which includes the pseudo code language as well as terms that will refer to participants of those algorithms.

2.5.1 Definitions

It is crucial to have a coherent notation to refer to nodes of different functions, their vicinity or any other parts that can be referred to in the context of sensor networks and data security. The previous sections have already defined several notations, hence the purpose of this section is to repeat those definitions already given and complete the overall notation with definitions that are still lacking.

A sensor network is assumed to consist of n nodes, which are assumed to be distributed at uniform density in the sensor network, to refer to all n nodes in a network the set $\Pi = \{a_1, \dots, a_n\}$ is used. Within those nodes b base stations are located somewhere, and a single base station will be referred to as BS or in case of several base stations need to be distinguished, the notation $BS_1, \dots, BS_b$ is applicable. Similarly, a convention is defined on the naming of nodes of the network that is influenced by the functions such nodes are ascribed. Most algorithms need a *leading node* that is executing the main part of the algorithm, this node will be denoted as s. Important for such a node s are the nodes that are in its neighborhood, or more precisely the nodes that s can reach by using its communication device without any intermediate routing or forwarding done by other nodes. This set of nodes depends on the node s itself, and is denoted as $V(s)$. The leading node s can interact with other nodes, which can be nodes in the vicinity of s, which will be individually referred to as $v, v', v'', \dots \in V(s)$ and $v_1, v_2, v_3, \dots \in V(s)$ respectively, or *arbitrary nodes* in the network (nodes with no particular distinction in terms of function or location), denoted as $a, a', a'', \dots$ and $a_1, a_2, a_3, \dots$ respectively. Thus for nodes the variables s, v and a are used with the mentioned meanings, naturally the notion of vicinity V applies to all defined kinds of nodes. Furthermore, if it is of particular importance then nodes that store data, i.e. hot nodes, will be distinguished by a tilde. Thus an arbitrary node that stores data and that fact needs to be stressed, the node will be denoted as $\tilde{a}$. If the overall number of hot nodes in the network needs to be specified, the variable h is used to refer to the count of data storing nodes in the network. The other important part the algorithms will work with are messages, if those messages have no particular structure or their contents is rather irrelevant, they will be denoted as $msg, msg', msg'', \dots$ or $msg_1, msg_2, msg_3, \dots$ respectively. The notation for adversary models has been introduced in the preceding section, hence restating it here is omitted. Naturally, the notational conventions introduced in this section apply only if not stated otherwise.

In addition to the variables that are used to talk about nodes, each node needs to have an identification, which is used to distinguish a node from others in the network itself. Such identifications are provided by the routing scheme, which uses such a identification/location to perform the routing of messages. In the following this identification of a node a will be referred to as $ID(a)$, if it needs to be particularly stressed that a node's identification is used. However algorithms can use both notations a and $ID(a)$ in order to refer to a particular node in the context of the procedure described. In several instances it becomes necessary to have a possibility to refer to the location that a node a occupies in the network. As the assumed routing mechanism is coordinate base, it needs to assign each node some kind of information that has takes the form of *euclidian coordinates* or can be converted to such easily (for instance when *polar coordinates* are employed). Thus in order to refer to the location of a node a the notation $LOC(a)$ is used; in order to refer to the appropriate euclidian coordinates of $LOC(a)$ of the two dimensional plane the sensor network can be projected on, the following notation is used: $LOC(a).x$ gives the x-coordinate of a's location and correspondingly $LOC(a).y$ gives the y-coordinate of a's location.

2.5.2 Algorithm Notation

Now the basic notational conventions are clear, the next issue is to explain how algorithms are presented in order to guarantee an unambiguous understanding of the presented algorithms. In the code as well as in the explanations the widely accepted notation of mathematics will be used, especially set theoretic operations are frequently employed and the reader is assumed to have sufficient familiarity with that notation.

Algorithms presented in this thesis can be seen as distributed algorithms, as they run on several nodes at the same time with interactions among those nodes. In many cases, the algorithm needs to distinguish between two or more parties that need to perform different tasks, but still are part of the same algorithm. Thus when describing how an algorithm works a distinction can be made between those parties by preceding the actual code with a reference to the respective node that is supposed to run the particular part of code. To clarify the parties involved in a distributed algorithm, the participants are listed before the actual code is shown. Similarly, the parameters that influence an algorithm are also stated and shortly explained before the actual code is provided.

The pseudo code that is used to describing the algorithms is defined to be as descriptive as possible to allow for easier understanding of what the algorithm actually does. Below explanations of keywords and their respective function for the used pseudo code are given. The node that is running the program described with the code and is thus calling the respective instruction of referred to as executing node.

- **upon event** *event*
  Distributed Algorithms perform tasks when messages are received or certain conditions come true. Thus the parameter of this keyword, *event*, will need to become true in order to cause the body of this statement to be executed. It is assumed that at running time these *event* conditions of those statement are checked regularly. A special event is the **initialization** event, that is true at the start up of the algorithm and, if needed, allows the algorithm to perform setup tasks, like initialization of variables before any other events are handled. Similarly, the special event denoted **invocation** is raised when another code part running on the executing node invokes a specified algorithm. This can only be used for algorithms that offer a certain functionality that can be used by other algorithms.

- **raise** *event*
  The **raise** is used to allow algorithms to exchange information among each other by signaling that some event has occurred. Those can be intercepted by the **upon event** keyword and appropriate action can be taken. Of course, this does not only involve the algorithm at hand, but any algorithm running on a node can react to such raised events.

- **invoke** *method*
  The invoke keyword takes as parameter *method*, which is supposed to contain the name of an algorithm that should be started with possible parameters. This is mostly used to indicate use usage of other algorithms as part of an algorithm in order to reduce redundancy in code.

- **process message** *message*
  This keyword simply causes a *message* to be passed to the process that might be waiting for it to arrive. This keyword is important when describing Camouflage Techniques that can be imagined to act between the application side and the routing mechanism.

- **wait** *amount of time*
  This instruction causes the executing node to block for the given amount of time.

- **start timer** *time variable*
  This keyword causes a timer to be started for the passed *time variable*, which can either be functioning as timeout or as a time interval. The associated timer for *time variable* stops executing either when the timeout is reached or in case the respective **stop timer** command is called.

- **stop timer** *time variable*
  This is the complementary keyword to **start timer** and causes the timer for the *time variable* to be halted (if the timer was not started before, then this keyword is assumed to have no effect at all).

- **foreach** *time interval* **do**
  To repeatedly execute a statement or a block of statements, the **foreach** keyword will cause its body to be executed each time the given *time interval* elapses. This can be imagined as a timer that calls a code segment periodically.

- **forward message** *message*
  This keyword is used to cause the executing node to forward the message *message* according to the underlying routing mechanism.

- **local broadcast** *message* [**to** *destination*]
  This invokes a local broadcast that sends the *message* to all nodes in the vicinity of the calling node. If a node *destination* is given as parameter the broadcast will be directed only to that node, and should be ignored by the others in the vicinity of the caller.

- **point-to-point** *message* **to** *destination*
  This keyword is used to send a message *message* to a single node *destination*, by using the assumed routing mechanism. If the *destination* happens to be a node in the vicinity of the calling node then this call emerges to be the same as a **local broadcast** invocation.

- **global broadcast** *message*
  This keyword invokes the *message* to be send to all nodes in the network.

- **randomly generate**
  Using the assumed pseudo-random number generator, sensor nodes can generate random bits and therefore algorithms can take advantage of those during their operations. Such generators have been introduced for sensor networks, see [17] for instance. Moreover, some sensor nodes can take advantage of the data their sensors constantly provide to use as part of the random seed necessary for random number generators. Either a real number can be generated or an array containing a specified amount of random bits.

- **randomly choose** *element* **from** *set*
  The instruction causes an *element* to be chosen from a given set. If *set* is clear from the context, then an explicit mentioning of *set* in the call is superfluous and can be omitted. The choice of *element* is assumed to be made with a uniform distribution. Of course, if several several elements are needed this call can be shortened to using a set instead of a single element for *element*.

- **with probability** *probability* **do**
  In many cases it is necessary to use the available random number generator to control the behavior of an algorithm by executing a piece of code only with a specific *probability*. The **with probability** keyword causes the following code, in the respective scope, to be only executed with the probability passed as parameter. Naturally, this command can be combined with an **else**, whose respective body is then executed with probability 1 - *probability*. Note that in case **with probability** is used with an **else**, one of the two respective bodies will be executed for sure.

- **calculate closest nodes** *set of nodes* **to** *destination*
  This keyword triggers a calculation of the distance each node in the vicinity of the calling node has to the *destination*, sorts the nodes according to the resulting distances and fills up the set *set of nodes* with the the nodes exhibiting the lowest distance to the destination. For instance, a call like **calculate closest nodes** $\{v_1, v_2\}$ **to** *dst* will have the effect that v_1 and v_2 will refer to the two nodes closest to the destination node *dst* in the vicinity of the calling node.

The message sending keywords **point-to-point**, **local broadcast** and **global broadcast** can be preceded by **encrypted** in order to stress that the message send is encrypted by the assumed encryption scheme. Furthermore, the **point-to-point**, **local broadcast** and **global broadcast** sending methods are supposed to satisfy the respective properties of the communication primitives by the same names, that were defined in the formal model of sensor networks.

For easier description of algorithms a list structure is assumed to be available. This structure allows for simple bookkeeping and is mostly used in the chapter of Evasive Data Storage. The following keywords are used in the context of such lists:

- **initialize** *list_name*
  Initializes the list, if the list is limited in length then this limitation should be obvious from the context. The list can be also given explicitly, of the form $(\diamond, \ldots, \diamond)$, where the $\diamond$ symbol stands for an arbitrary entry including the possibility of an empty or default placeholder in the list, which is explicitly, referred to by the symbol $\times$.

- **H** *list*
  This command returns the head of the *list*, which is defined as the leftmost element located in the list or the last one that was inserted into the list respectively.

- **T** *list*
  This command returns a sublist of *list* namely, the list obtained when cropping the head of *list*.

- *element* $\rightarrow$ *list*
  The list allows for easy insertion of elements into the list. This is done by specifying the *element* to be inserted, followed by the symbol $\rightarrow$ and the list *list* to store the element. This can cause two behaviors, the first is that the inserted *element* becomes the head of the *list* and, if the list is assumed to store only a finite amount of elements, then the last element in the list is dropped and lost unrecoverable.

To denote a sublist relation between two lists $list_1$ and $list_2$, the symbol $\sqsubset$ is used in infix notation, i.e. $list_1 \sqsubset list_2$ states that the sequence of elements contained in $list_1$ can be found in the same order somewhere in $list_2$. It is important to note, that elements in $list_1$ that were depicted as $\diamond$, must be the same elements in $list_2$ in order to for $list_1$ to be a sublist; whereas any sequences of $\times$ at the end of a list are ignored, when testing for sublist relation.

In addition to the above keywords, common keywords are also used, however those do not need further explanation as they are supposed to be clear: **and**, **or**, **if**, **then**, **else**, **while**, **do**. Furthermore, the pseudo code uses indentation to convey the scope or body of statements, like needed for instructions similar to **if**, **while**, **foreach** or **upon event**.

Finally, many algorithms will deal with data items, hence a notation for referring to the assumed fields of such items is convenient. Assuming that D is a data item, algorithms can access the following fields:

- $D.Gnid$ is the generating node identifier.
- $D.Ts$ gives the timestamp of the data item.
- $D.Type$ refers to the type identifier of D.
- $D.Raw$ is the raw or application depended data that the data item stores.

The default data item $\square$ is assumed to have an predefined fields, where the $\square.Ts$ is to be the lowest timestamp possible in the network.

2.6 Summary

This chapter introduced the notion of sensor networks along with several basic properties that need to be kept in mind, when investigating corresponding algorithms for such networks. Special emphasis was put on conveying current data storage approaches in such networks, as subsequent chapters will focus on a topic closely related to that area. The main point, the thesis wants to address, is the node capture problem that was presented as a challenging problem for which the thesis intends to provide tools alleviating the severeness of that problem. Towards the end of this chapter, the needed adversarial models were presented that will be used to precisely describe the usability of the solutions given the presence of malicious users. However, the usefulness of those models is con restricted to the scope of this thesis only and can be used in various other contexts. Hence, this chapter must not be regarded as providing background knowledge and definitions only, but also as defining adversary models that can be used in other works concerning sensor networks. Finally, the chapter introduced the the needed terminology and notation for the remaining chapters. Hence sufficiently much background is available to study different Camouflage Techniques in the following chapter, which form the first step to an improved protection against node capture and for improved security of data.

Chapter 3

Camouflage Techniques

The last chapter argued that the Node Capture Problem is a major issue in sensor networks, mostly because of the threat this problem poses to the gathered data contained in the network. The entity that is responsible for the problem is the adversary that resides in the hostile environment, where a sensor network can be located. Such an adversary was argued to have two possibilities to achieve his objective of accessing hot nodes in the network: the first one is to find such interesting nodes and the second is to remember such nodes. This chapter focuses on the former part, hence presents approaches aiming to hinder the adversary from finding hot nodes in the network. The main techniques that will be used to create obstacles for adversaries are camouflage or anti-traffic analysis methods. This section will address two categories of camouflage, namely *global* and *local*. As the descriptions of those two categories indicate, the global category includes algorithms that can involve large parts of the network, whereas the local category tries to limit the camouflage to a few involved nodes.

The hindering of an adversary that exhibits traffic analysis skills equal to the *local* or *global* levels can be encapsulated into a security property, which the algorithms of this chapter try to fulfill. Put into simple terms, the goal is to lower an adversary $\mathcal{A}$'s abilities, where $\mathcal{A} \succeq \mathcal{A}(local, \star)$, such that $\mathcal{A}$'s traffic level converges to the $\mathcal{A}(blind, \star)$.[1] Thus the according security property that camouflage algorithms need to satisfy in order to induce $\mathcal{A}$'s loss in traffic analysis power is:

> **Camouflage Security Property:** An adversary $\mathcal{A}$ satisfying $\mathcal{A} \succeq \mathcal{A}(local, \star)$ cannot distinguish a hot node from a cold node with greater probability than p_i at any time t.

The lower bound on p_i is $\frac{h}{n}$, which is obvious but will be shown later on in detail. The investigation as to how effectively the camouflage algorithms of this chapter fulfill this property will be investigated in the Simulations Chapter, where also the methods for evaluating the techniques are introduced. This chapter focuses on the presentation and explanation of the camouflage algorithms along with some cost evaluations and empirical insights into other security issues. The algorithms presented here are mostly presenting already developed algorithms in other works, however the Local Camouflage is original work created for to camouflage special patterns that can be observed in sensor networks. Furthermore, the end of this chapter also contains a possible improvement to all camouflage concepts currently available, but due to its nature it cannot be investigated thoroughly as will be explained in the according section.

3.1 Global Anti-Traffic Analysis Algorithms

The global camouflage that this section intends to present is concerned with how the essential communication that needs to take place among sensor nodes, base stations and particularly hot nodes can be modified to give less evidence to adversaries regarding the location of gathered data in the network. The emphasis in global anti-traffic analysis is that the camouflage is applied to communication between distance nodes in the network. A simple example that gives an intuition of how communication in the network can be exploited to get to hot nodes is an adversary $\mathcal{A}(local, \star)$ that observes a certain fraction of the network and recognizes that a certain node is more frequently participating in message exchange than other nodes, thus is more probably harboring data that needs to be queried often. Similarly, the same adversary $\mathcal{A}$ can observe a legitimate user querying

[1]The additional advantage that an active adversary might obtain through malicious nodes acquired in the network is here ignored.

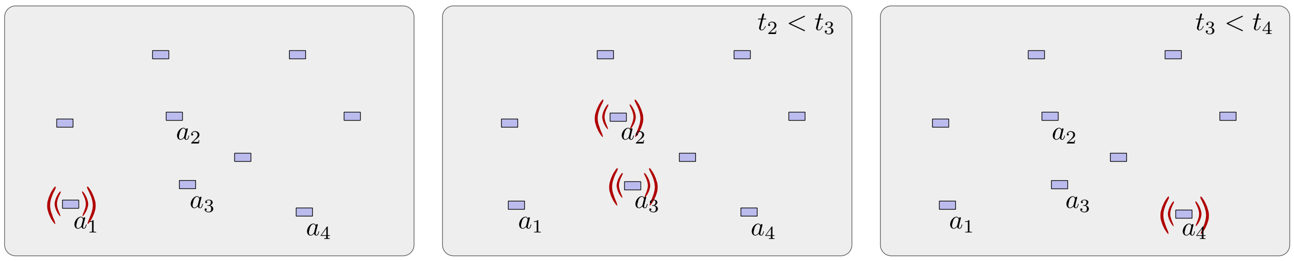


Figure 3.1: An Illustration of Time Correlation

a couple of sensor nodes for data, which then initiate communication with either a base station or hot nodes directly. Hence, $\mathcal{A}$ can try to follow the message trace eventually leading to an interesting node. Thus it is obvious that camouflaging messages sent in a sensor network, i.e. provide forms of global camouflage, can provide significant advantages for protection of data.

This section begins with a short description of Time Decorrelation, which is a very basic and well known technique to complicate tracing of single messages through the network. This is however given only for completeness and will not play any role in the sections to follow. Furthermore, the Fake Packets algorithm is used to show a notion that is global, but not application oriented. However the main techniques this section provides are *Multipath Routing*, *Random Walk* and *Fractal Propagation*, which can be seen as stepwise refinements of each other: Multipath Routing being the most basic, followed by Random Walk and finally resulting in the Fractal Propagationnotion. Those three approaches were introduced in [1]. Yet the work done and presented in [1] was concerned with protection of base stations only. Thus the algorithms presented here are variations of those presented for node to base station message exchange only, in the sense that they allow the camouflage techniques to be applicable to arbitrary communication parties, i.e. node to node for example. The changes that allow the three algorithms to be used in more circumstances are based on the assumption of a coordinate based routing scheme, whereas the ones in [1] used the notion of *parent nodes*, which are special nodes that arbitrary sensor nodes can send messages to, if wanting to communicate with base stations. Each node in the sensor network in [1] is assumed to have knowledge of two parent nodes, which also limits the notion of Multipath Routing. The Multipath Routing algorithm provided in this work alleviates this restriction. As a side note it should be said, that the generalization of the Fractal Propagation notion is introduced not in this chapter, but as contents of the Simulations Chapter and the respective section in the Appendix. Hence, the algorithms presented here, which are connected to the Fractal Propagation notion, are simply modified versions of the original ones, which can also be regarded are generalizations, but not as significant as the one presented in the appendix.

3.1.1 Time Decorrelation

The technique of Time Decorrelation addresses a simple scenario: a message *msg* is send by a source node to some destination node. The source node initiates a local broadcast, triggering a node in its vicinity to forward *msg*, which also issues a local broadcast to trigger another node. This process repeats until the destination is in the vicinity of one of the nodes, thus ending the chain of broadcasts. An adversary observing the fragment of the network where this chain propagates through can make the assumption, that the nodes forwarding *msg* will do so immediately after having completed receiving *msg*. Thus the adversary can use time to distinguish nodes of the chain from other nodes performing broadcasts in the same time interval, resulting in possibly successful identification of the destination node.

This tracking based on time is illustrated in figure 3.1, which shows a part of a sensor network that is presumably monitored by an adversary $\mathcal{A}(local, \star)$. The leftmost picture of the figure shows a node a_1 that commits a local broadcast, sending some data one its neighboring nodes. $\mathcal{A}$ can observe and record this event. Shortly afterwards node $a_2 \in V(a_1)$, depicted in the middle picture, initiates a local broadcast and finally shortly afterwards node a_3 does the same. The time t_2 at which node a_2 started to broadcast is less than t_3 the corresponding starting time of a_3. Having monitored those events along with the times, adversary $\mathcal{A}$ can infer that either the message node a_1 sent passed through a_2 and then further to a_3 or that one of the two nodes, a_2 or a_3, started its own transmission. However, due to the observations $\mathcal{A}$ can make on the network, he can have an accurate estimate how long messages usually remain at a node before they are forwarded, thus allowing him to decide which case is at hand. Furthermore, in the right picture of figure 3.1 node a_4 participates in the message exchange, yielding that it somehow was triggered by the broadcasts initiated previously. With the aforementioned accurate estimate of time delays in nodes' forwarding behavior, an adversary could deduce the correct path of a message from a_1 over a_2 and a_3 to a_4. Hence, for instance, conjecture that a_4 might know contain interesting information. To avoid such scenarios the adversary has to be kept from accurately correlating broadcast events he observes with each other. One way to do break the rather straightforward correlation is to introduce randomness into the forwarding process, which is what the Time Decorrelation algorithm does does.

The notion of Time Decorrelation is to introduce randomness into the forwarding mechanism. This is done by forcing each node that has to forward a message to wait a random time interval before actually doing so. Algorithm 3.1 shows how this idea can be formulated in pseudo code. The node s is assumed to have just received a message *msg*, which is brought to its attention by making the **upon event** statement true. First, the message is checked if it might already be at its destination. If so *msg* is provided to another layer for processing. In the other case, the algorithm generates a time interval of random length, which can be at most *time_max*, and forces s to retain *msg* until the chosen time period has passed. Then algorithm 3.1 passes *msg* to the routing mechanism for usual forwarding. The result is that an adversary cannot correlate messages sent by nodes in his vicinity, making it hard to trace single messages through the network.

```
Participants:
    s           - the node that receives a message to forward
Parameters:
    msg         - the data message to be forwarded
    time_max    - defines the maximum time to wait before forwarding msg

    s:          upon event receiving message msg do
                -     if msg needs not to be forwarded then
                -         process message msg
                -     else
                -         randomly generate a real number $r \in [0,1]$
                -         wait $\lceil r \cdot time\_max \rceil$ time units
                -         forward message msg
```

Algorithm 3.1: Time Decorrelation

The algorithm 3.1 has to be seen as being executed before the underlying routing scheme can forward the message. The algorithm therefore modifies the behavior of the routing scheme, which would normally go on and directly forward the message without any delay. Algorithm 3.1 intercepts a message and checks whether an application is waiting for this message or if it needs to be further forwarded, where it delays the process before giving control back to the routing scheme by calling **forward message** *msg*.

3.1.2 Fake Packets

The Time Decorrelation approach aims at hindering an adversary from pursuing messages through the network by randomizing the time delay messages experience when being forwarded at a node. Another, possibility is to send *fake* messages that do not bear any useful contents at all. Hence, the notion of Fake Packets presented in this section, should be a simple introduction into this form of camouflage, which does not involve modifying the time delay of the routing mechanism used. Furthermore, fake messages or fake packets address another phenomenon that is observable in a sensor network: an adversary $\mathcal{A}(local, \star)$ can observe the network and keep track of the amount of messages passing through nodes in his vicinity. This observation allows $\mathcal{A}$, not to track messages, but to recognize routes or paths that are frequently used by messages over time. Furthermore, patterns emerge that reflect a myriad of actual messages, which can also indicate locations of interesting nodes $\mathcal{A}$ could be interested in finding. To make those mentioned patterns more imaginable, a base station can be thought of that is assumed to be frequently messaged by other nodes in the network, hence an emerging pattern might consists of several paths that all lead to a single base station, making the nodes surrounding a base station hot spots for message forwarding and $\mathcal{A}$ should be able to clearly distinguish those active nodes close to the base station from ones that are not close at all.

```
Participants:
    s          - node that executes the algorithm
    a          - some node that is chosen by s
Parameters:
    p_f        - probability to send a fake message
    time_int   - time interval that specifies when p_f is evaluated
    fake_msg   - the fake message to be used

    s:         foreach time_int do
               -     with probability p_f do
               -         randomly choose an arbitrary node a
               -         encrypted point-to-point fake_msg to a
```

Algorithm 3.2: Fake Packets Algorithm (FakeP)

Hence the idea of fake messages is to send messages that do not have any particular purpose, thus shifting the attention away from real patterns, that might lead to valuable nodes, to fake patterns, that lead to dead ends or uninteresting nodes. The most basic way to achieve this is Fake Packets, which is executed on all nodes in the network and allows each node to send a fake message to an arbitrary other node in the network, with a wisely chosen and sufficiently low probability p_f. The corresponding pseudo-code is given in 3.2. The probability p_f directly influences the number of fake messages generated by the approach. A high value for p_f will yield a large number of fake messages being send though the sensor network, resulting in rather fast energy exhaustion, but also in a higher overall entropy or randomness of the observed network. High entropy guarantees that an adversary cannot obtain any useful information from traffic analysis. Nonetheless, high entropy, i.e. high message activity, everywhere in the network is too energy consuming. Even more, fake messages trying to increase entropy in the system will be harmful in case the present adversary $\mathcal{A}_b$ falls into the $\mathcal{A}(blind, \star)$ model. In case of an adversary $\mathcal{A}_l(local, \star)$ high entropy in parts of the network that are not in $\mathcal{A}_l$'s vicinity, will be wasteful too. This is one deficiency of the easy Fake Packets algorithm, which blindly tries to camouflage the complete sensor network, which might result in rather awkward behavior, like camouflaging an inactive part of the network whereas at the same time $\mathcal{A}_l$ can freely analyze messages patterns in his vicinity, which happens to not be performing any camouflage. Thus more application oriented camouflage that acts in the presence of actual traffic is more useful and appropriate, then the not application oriented Fake Packets algorithm.

It is crucial realize how a node knows that a fake message has arrived, but an adversary cannot determine this easily. This is due to the assumed encryption scheme that is used and assures that an adversary cannot determine what actual data a message contains, only the destination identification might be accessible to him. Furthermore, the size of the fake messages is also a crucial part of appropriate camouflage, hence the Fake Packets algorithm allows for different fake messages to be used, as shown in algorithm 3.2. Nonetheless, the mentioned drawbacks clearly opt for more application oriented algorithms that will be discussed subsequently.

3.1.3 Multipath Routing

The Multipath Routing notion was introduced in [1], and similar ideas, but in the context of routing only, can be found in [2] and [20] respectively. However, the Multipath Routing notion this section introduces is based on the one presented specifically for anti-traffic analysis in [2], with the addition of more flexibility by usage of the assumed coordinate based routing scheme.

Normally, the association with routing in a network is that a message is forwarded from one node to another, forming a path through the network until reaching the destination. There are vast amounts of routing algorithms used in ordinary networks and several in sensor networks, where in the latter a routing path mostly resembles a single line, which remains fixed over the lifetime of the sensor network [2]. One such example, which also can be found in sensor networks, is the *Greedy Forwarding Scheme*: each node forwards data to the node that is closest to the destination, which is also the scheme assumed to be used in the coordinate based routing available in the sensor model presented at the beginning. Using the same single path is convenient as there do not seem to exist plausible reasons to do otherwise. However when reflecting on the assumptions made on the environment of sensor networks, where adversaries pose threats to data integrity and even data routing, several reasons for a different routing notion come to mind. Hence the question is what scheme addresses such threats? Looking at a scenario where an adversary placed a malicious node in the network that can manipulate traffic passing through arbitrarily, provides a justification to route data not along a fixed single path to its destination but along one of a set of possible paths. Similarly, but more important for the aspect of possible traffic analysis is that employing a routing scheme that sends messages from a source node to a destination node over mostly the same path, invites an adversary $\mathcal{A}_l(local, \star)$ to perform traffic analysis and quickly recognize that very path as an emerging pattern. Of course, such a pattern allows him to easily follow data from the source to the destination. In order to avoid the formation of such patterns, the number of possible paths to reach the destination needs to be increased, thus allowing for a more disperse pattern that naturally does not allow $\mathcal{A}_l$ to easily recognize a path pattern to the destination.

The problems and possible approaches lead to the notion of p-Multipath Routing, as shown in figure 3.2 yielding different possible paths to the destination, where the continuous line and the dashed line are two possible paths of which only one is used at a time. To realize this idea, each node s is assumed to have at most p different neighbors to send its data to and those neighbors actually are helpful in forwarding the data, i.e. they lie on the way to the destination or reduce the hops to the destination. Naturally, the integer p depends on $|V(s)|$, the number of nodes in the vicinity of s; to be more precise p cannot exceed $|V(s)|$ and in realistic scenarios p can only be a fraction of $|V(s)|$. The obvious advantage of Multipath Routing is that instead of having a single choice at each hop (as it is in Greedy Forwarding), this number is increased to p at each hop. Hence, if the node to forward a message to is chosen randomly among the p possible nodes then a larger dissemination of the paths used for communication can be reached. Furthermore, the notion of p-Multipath Routing fits nicely into the Greedy Forwarding that is mostly used with coordinate based routing: setting p to equal 1 results in choosing the closest node to the destination.

Algorithm 3.3 depicts the modification that needs to be made to the routing scheme in order to achieve the p-Multipath Routing scheme. The algorithm is similar in structure as Time Decorrelation, with the difference that in case the received message *msg* needs to be forwarded, the node calculates the set $F := \{v_1, \ldots, v_p\}$, which contains the p closest nodes to destination node *dest* of the message *msg*, and randomly chooses one node from F to forward *msg* to. [3]

As can be seen in Algorithm 3.3, 1-Multipath Routing will result in a Greedy Forwarding approach that tries to get to the destination as fast as possible, whereas the 2-Multipath Routing allows at each node that forwards the message either the closest or the second closest node to be chosen for forwarding. This 2-Multipath Routing case is most similar to the usage of two parent nodes, which was employed in the original introduction

[2]If failures and dislocations are ignored.

[3]As with the Greedy Forwarding notion, Multipath Routing can suffer from the Void Problem, where all p possible nodes are further from the destination than the leading node s. In such a case the Perimeter Rule can be applied just as with Greedy Forwarding.

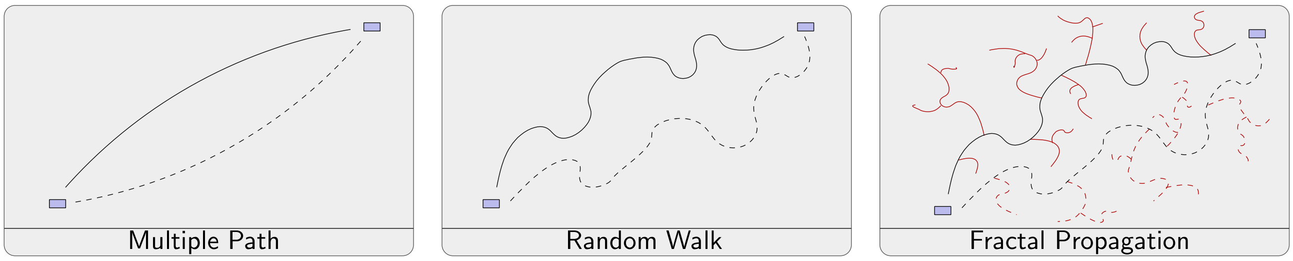


Figure 3.2: Refinement Steps for Fractal Propagation

```
Participants:
    $s$        - the node that executes the algorithm
    $v_i$      - nodes in the vicinity of $s$
Parameters:
    $msg$      - the data message to be forwarded
    $dest$     - the identification of the destination node to receive $msg$
    $p$        - the maximal number of neighbors to consider for forwarding

    $s$:       upon event receiving message $msg$ do
               -     if $msg$ needs not to be forwarded then
               -         process message $msg$
               -     else
               -         calculate closest nodes $F := \{v_1, \ldots, v_p\} \subseteq V(s)$ to $dest$
               -         randomly choose a node $v \in F$
               -         local broadcast $msg$ to $v$
```

Algorithm 3.3: Multipath Routing (MPR)

of this path dispersion.

Now that Multipath Routing is fully specified, a short look is taken at what advantages it offers in the presences of adversaries. Although, a detailed evaluation will be given as soon as simulation data is analyzed in the Simulations chapter and mostly only as part of the Fractal Propagation algorithm. Multipath Routing causes messages sent by a node to some destination node to take different paths. This makes it difficult for an $\mathcal{A}(\star, active)$ adversary that installed malicious nodes to obtain all messages that were sent, as the probability is high that only a fraction of the messages will actually pass through the malicious nodes. Nonetheless, the $\mathcal{A}(\star, active)$ adversary can manipulate those messages that will be routed through his malicious nodes thus posing a severe problem for the correctness of data. Thus an extension of Multipath Routing will send a message along one of the possible routes and several (substantially smaller) checksum messages, that contain a computed checksum of the message to be send, along different possible routes to the same destination. Thus the destination can check by using the checksums it obtained, if the message was not modified by a malicious node that happened to lie on the way. And in case of doubt request a retransmission, which will most probably go along a different path. This mechanism that was just describe, however, goes more into the direction of secure routing and is thus out of the scope of this thesis for a more detailed elaboration. The main advantage that Multipath Routing offers is in the case of an adversary $\mathcal{A}_l(local, \star)$. $\mathcal{A}_l$ looses his ability to recognized communication patterns in his traffic analysis, as the path can vary substantially, depending on the p parameter. A large value for p will make it difficult to recognize a single path that messages from their source to their destination take, however as Multipath Routing chooses among the closest nodes to the destination, it is supposed to stay focused around the single path that a message would take, if Multipath Routing was not employed. Thus p is chosen to be far smaller than the number of nodes in the vicinity of the current node s executing Multipath Routing: $p \ll |V(s)|$. Carelessly, taking p too large will result that messages will stray in the network with very little advancement to the actual destination node. Hence, in order to alleviate the focused forwarding of Multipath Routing the following notion of Random Walk provides a controlled way for deviation from the focus exhibited by Multipath Routing, besides careless increasing of p.

3.1.4 Random Walk

As referred to in the previous section, at this point the notion of Random Walk is introduced. It represents a refinement of the Multipath Routing notion of the last section. As mentioned in the explanations of Multipath Routing the mechanism chooses nodes that are closest to the destination of the message to be forwarded and thus resulting in a rather focused path towards the destination. Hence, the idea is to allow for controlled deviation from the focused direction of the paths of Multipath Routing: instead of always following one of the closest nodes, a certain probability causes an arbitrary node in the vicinity to be randomly chosen for forwarding. Hence the result is what can be depicted as a wiggly line, see figure 3.2 (middle picture), which is no longer a focused path, but a path that can deviate even in completely opposite directions. The figure also shows an alternative path, the dashed line, that could be chosen due to the underlying Multipath Routing algorithm. However, such deviation is controlled by a wisely chosen probability, in order to avoid an aimless wandering of messages in the network, which could be caused by using too high values for p in Multipath Routing.

Participants:	
s	- the node that executes the algorithm
v	- a nodes in the vicinity of s

Parameters:	
msg	- the data message to be forwarded
$dest$	- the identification of the destination node to receive msg
p	- the maximal number of neighbors to consider for forwarding
p_r	- the probability influencing the randomness of the routing path

```
s:    upon event receiving message $msg$ do
-         if $msg$ needs not to be forwarded then
-             process message $msg$
-         else
-             with probability $p_r$ do
-                 invoke MPR(msg, dest, p)
-             else
-                 randomly choose a node $v \in V(s)$
-                 local broadcast $msg$ to $v$
```

Algorithm 3.4: Random Walk (RW)

Algorithm 3.4 shows the pseudo code of Random Walk. The algorithm assumes that node s received a message msg and is either the destination or is supposed to forward msg to the destination $dest$. In case s needs to forward msg it calls the Multipath Routing algorithm with probability p_r, and therefore forcing the message to stay on a focused way to its destination. The crucial parameter, hence, is p_r, which defines the probability that a packet is forwarded to the node Multipath Routing chooses. If it was not for this probability p_r, Random Walk would behave the same as Multipath Routing. However the essential difference is that Random Walk allows msg to be forwarded to an arbitrary node in the vicinity of s with a probability of $1-p_r$. The parameter p_r therefore controls how strongly msg can deviate from the focused path Multipath Routing would suggest.

The advantage is that with the probability p_r the deviation from the actual path to the destination can be controlled and thus causes patterns that an $\mathcal{A}_l(local, \star)$ adversary observes to be more disseminated and ambiguous. Naturally, such an advantage comes with a certain cost: the Random Walk walk path deviates from a simpler, shorter and therefore less message intensive path that Multipath Routing would suggest. Hence the question is how much longer do the paths formed by the Random Walk notion become. Assuming that the length of a path is measured with the number of hops it takes a message to be transmitted from its source to its destination, then let k be the number of hops a message takes without applying a random walk and k' the number of hops with a random walk. The cost measure C is defined as the ratio of the number of hops without a random path and with a random path: $C = \frac{k'}{k}$. For a simple estimate of the incurred overhead, it is assumed that only the path suggested by the Multipath Routing algorithm reduces the number of hops to the destination and that any other deviation from such a path does not change the distance. Thus the relation $k \cdot p_r = k'$ applies, yielding as cost[4] $C = \frac{k'}{k} = \frac{k \cdot p_r}{k} = \frac{1}{p_r}$. This nicely reflects how the probability p_r controls the deviation and the therefore increased amount of hops: for $p_r = 0.5$, i.e. it is equally likely to take a step on the focused path or to deviate, the incurred cost $C = 2$, meaning that the path in such a case can become twice as long due to deviation. A more restraining value of $p_r = 0.8$ yields a cost of $C = 1.25$, which is less but also causes forwarding to be more focused. An exact evaluation of the effectiveness of the parameters can be looked up in [1].

3.1.5 Fractal Propagation

Fractal Propagation is the final step in the refinement that started with Multipath Routing and continued with Random Walk. The basic idea is to add fake branches to the wiggly pattern exhibited by the Random Walk notion. This is depicted in the figure 3.2 (right picture), where the actual message takes the path outlined by the black line, and randomly fake paths emerge as the message is forwarded (red lines). [5]

How this addition of fake branches to the actual message path can be realized, is described in Algorithm 3.5. The algorithm is no longer concerned with forwarding some message msg that is send from a source node to a destination node, but assumes that such messages are forwarded by using the Random Walk notion. The algorithm involves three kinds of parties, denoted s, v and v' that can stand for arbitrary nodes that happen to be able to satisfy the appropriate condition in their respective **upon event** lines. However the naming of those nodes serves a purpose: the s can be taken to initiate the whole process of Fractal Propagation, whereas v is assumed to lie in $V(s)$ and respond to the initiation of s, and finally $v' \in V(s) \cup V(v)$ can be seen being triggered by the fake message of v. Of course, other constellations of messages that trigger the described behavior of s, v and v' are possible, but for the explanation of Fractal Propagation the chosen naming is most conventional. The algorithm starts with a node s overhearing the transmission of a real message, which is assumed to be performed by Random Walk. With a probability of p_c the node s will start to generate a fake message of size $fake\_size$ and forward this fake message to some randomly chosen node $v \in V(s)$. Receiving such a fake message, the node v will itself forward the fake message to some random node from its vicinity with probability p_f. Those two actions form the basis for the Fractal Propagation, however another possibility needs to be considered, the branching from a fake branch, i.e. the creation of subbranches from other branches. This behavior is described by the v'', which essentially does the same as the s, with the difference that v'' is triggered by a overhearing a fake message and also having a different probability, namely p_t, determining the success of the branching attempt. Of course, over the course of fake message propagation a variable is carried around and decremented, such that the fake branches cannot penetrate the network arbitrary, but are guaranteed to disappear at a depth defined by the K parameter.

The algorithm uses several probabilities that need to be chosen wisely and have substantial influence on the efficiency of the algorithm. Probabilities that are too high, can cause too many fake messages to be send and thus waste energy; too low probabilities can endanger the whole attempt to hinder an adversary from recognizing patterns of communication in the network. Furthermore, finding the right value for the depth of the branches is also crucial, for the same reasons that probabilities have to be chosen after careful consideration. The best way to

[4]A similar argumentation is used in [1] for the node to base station case.

[5]The dashes lines depict what alternate route the message, and thus the fake branches, could take due to the fact that Fractal Propagation is based on the Multipath Routing notion.

Participants:

s	- current node
v	- arbitrary node in the vicinity of s
v'	- arbitrary node in the vicinity of s or v

Parameters:

p_c	- the probability influencing the generation of fake packets
p_f	- the probability influencing the forwarding of fake packets
p_t	- the probability influencing the generation of fake packets triggered by passing fake packets
K	- a propagation constant that controls the depth of the propagation (value is preset and known to all nodes)
$fake\_size$	- fake package size: size in bits of the fake packets to generate

```
$s$:    **upon event** overhearing a message $msg$ forwarded by some $v \in V(s)$ **do**
        -    **with probability** $p_c$ **do**
        -        **randomly choose** a node $v \in V(s)$
        -        **randomly generate** bits of length $fake\_size$ and store as $fake\_package$
        -        **point-to-point** message $(fake\_package, K)$ **to** node $v$
$v$:    **upon event** receiving a fake message $(fake\_package, depth)$ **do**
        -    **if** $depth > 0$ **and** $depth < K$ **then**
        -        **with probability** $p_f$ **do**
        -            **randomly choose** a node $v'' \in V(v)$
        -            **decrement** $depth$
        -            **point-to-point** message $(fake\_package, depth)$ **to** $v''$
$v'$:   **upon event** overhearing a fake message $(fake\_package, depth)$ **do**
        -    **if** $depth > 0$ **and** $depth < K$ **then**
        -        **with probability** $p_t$ **do**
        -            **randomly choose** a node $v''' \in V(v')$
        -            **decrement** $depth$
        -            **point-to-point** message $(fake\_package, depth)$ **to** $v'''$
```

Algorithm 3.5: Fractal Propagation (FP)

support the needed consideration and thus the right choice of values, simulations or field experience are helpful. For more detail concerning the message overhead incurred by the notion or the best choices for the probabilities parameterizing the Fractal Propagation notion, the original source [1] can be consulted. However the discussion of this approach is not complete without mentioning a variation of the Fractal Propagation algorithm presented in algorithm 3.5 that imposes certain restraints on the probabilities that control the algorithm. The first variation, named Differential Fractal Propagation, addresses the forwarding rate that a node encounters: if a node a forwards more packets/messages than a certain predefined threshold then both probabilities p_t and p_c are lowered depending on the encountered excess rate relative to the threshold. Hence, in the Differential Fractal Propagation variation, nodes that are exposed to frequent forwarding of messages will have a lower probability to actively send fake packets. More details on the Differential Fractal Propagation modification as well as another variation, which will not be explored here, can be found in [1].

Now that the algorithm is explained, a comment on the security properties it provides is needed: the Fractal Propagation notion is effective in case of an $\mathcal{A}_l(local, \star)$ adversary. $\mathcal{A}_l$ can observe a fragment of the overall traffic found in the sensor network. Thus when $\mathcal{A}$ follows messages through the network, he could do this fairly good with Multipath Routing and even Random Walk being used. However, Fractal Propagation will eventually cause a fake branch to emerge during the forwarding of messages. At such a point $\mathcal{A}_l$ has to decide which way to follow, the real message path or the fake branch that Fractal Propagation caused. Following the fake path, the adversary will eventually end up in a dead end and thus, due to his local abilities of traffic analysis, also lose track of the actual message path. A more strong adversary matching a $\mathcal{A}_g(global, \star)$ however can monitor the whole network and hence have a better possibility to recognize dead ends, especially when the depth K is less than the actual path length between the source and the destination, which is being camouflaged by the fake branches.

This section finished the techniques addressing global camouflage and presented Fractal Propagation as an effective and application oriented approach[6]. The following section introduced the notion of Local Camouflage and the respective algorithms.

3.2 Local Anti-Traffic Analysis Algorithms

The previous section has put great emphasis on work that is currently pursued in the anti-traffic analysis field, meaning that direct communication between for nodes that communicate is tried to be camouflaged by splitting paths and making it hard for an adversary to identify the real message path. The category into which the algorithms of that section fall was denoted by the term global anti-traffic analysis. This section takes a more specialized or local attempt to camouflage, in the sense that the techniques do not concentrate on participants that are located at great distance from each other, but rather close to each other, even only located in the vicinity of each other. Thus the here presented and emphasized category of Local Camouflage or Local Anti-Traffic Analysis is focused on different communication patterns arising in sensor networks than those addressed by global camouflage.

Particularly such local approaches can be used to equip and enhance algorithms with camouflage that work by having a leading node s communicate with its neighbors by several local broadcasts. The two akin approaches of Repeated Firework and Wandering Repeated Firework are useful for both camouflage of single messages and of patterns that an adversary of $\mathcal{A}(local, \star)$ or higher can recognize from longer observations. The following two algorithms presented here are original work and have not been seen in any similar form in the literature yet.

3.2.1 Repeated Firework

This section introduces an algorithm that addresses the issue mentioned at the beginning of the Local Anti-Traffic Analysis section: repeated local broadcasts of one node and the corresponding responses of its vicinity. To be more detailed, a node s passes through a number of communication rounds with the nodes found in its vicinity $V(s)$ and the nodes in vicinity of s respond to the messages s broadcasts, of course for the purpose of Repeated Firework the messages sent are all assumed to be fake messages. Similarly to direct message exchange between any two (distant) nodes, there is a need for somehow add fake message to that process or allow fake messages to complete imitate such a communication pattern as just described. Hence, Repeated Firework is an approach to fake several rounds of communication between a node s and its $V(s)$, which can either be used as a enhancements to algorithms or stand-alone to irritate an adversary that looks for the repeated local broadcast patterns in the network's traffic.

The pseudo code for the Repeated Firework is shown in Algorithm 3.6. The two parties involved are a leading node s that performs the local broadcasts to nodes in its vicinity and the nodes in the vicinity of s, referred to by the variable v. The leading node s starts off with a simple initialization and transmission of the first $fake\_msg$ (which is generated to fit the parameterized size for this message) that is local broadcast to nodes in its vicinity. After that s updates how many rounds are still left and waits for a response of an estimated number of nodes $v \in V(s)$ or for a $time\_out$ to kick in, which ignores the nodes that have not respond to the $fake\_msg$ yet. Once either all replies are received or the timeout is reached, s checks if still rounds are supposed to be carried out and, if applicable, initiates another round with a fake message. The nodes in the vicinity of s, i.e. the v node in the pseudo code, await the fake message and respond with a $fake\_reply$, whose size is also variable. This response however is only sent with probability p_r, which allows for a randomization of

[6]In this case: arbitrary communication between two nodes in the network.

Participants:	
s	- the leading node
v	- arbitrary node in the vicinity of s

Parameters:	
r	- specifies the number of repetitions
len	- specifies the size of fake message
$reply\_len$	- specifies the size of fake reply message
p_r	- specifies the probability of a vicinity node to participate in a firework round
$time\_out$	- specifies the probability of a vicinity node to participate in a firework round

```
s:    upon event initialization do
      -     set $cur\_round$ to $r$
      -     invoke trigger a camouflage round
s:    upon event trigger a camouflage round do
      -     randomly generate bits of length $len$ and store as $fake\_msg$
      -     local broadcast $fake\_msg$
      -     decrement $cur\_round$
s:    upon event receive $fake\_reply$ from $\lfloor p_r \cdot |V(s)| \rfloor$ nodes or $time\_out$ is reached do
      -     if $cur\_round > 0$ then
      -         invoke trigger a camouflage round
v:    upon event receive $fake\_msg$ do
      -     with probability $p_r$ do
      -         randomly generate bits of length $reply\_len$ and store as $fake\_reply$
      -         point-to-point $fake\_reply$ to $s$
```

Algorithm 3.6: Repeated Firework (RFire)

the overall patterns and surely extends the applicability of the algorithm. The cycle ends ones all rounds have been performed.

The usage of this camouflage is, as it has been already mentioned, local, allowing, for instance, a node s to camouflage communication with its neighboring nodes and especially adding fake messages as responses, because those responses could aid an adversary $\mathcal{A}_l(local, \star)$ to easily identify nodes $v \in V(s)$ that take a special role in the algorithm by responding to messages locally broadcasted by s. This implies that the *triggering of a camouflage round* can be understood to be triggered by message send in other purposes, by other algorithms[7]. Thus allowing this algorithm to be seen as an addition to arbitrary algorithms that exhibit a need for this sort of fake packets. Naturally, the approach can also be seen as a stand-alone algorithm that tries to fool an $\mathcal{A}_l$ adversary into believing that an algorithm having the same repetitive message pattern is executed somewhere, whereas it actually consists only of fake messages. This application has to be seen from the perspective that nodes might start algorithms with such patterns and thus pinpoint an $\mathcal{A}_l$, that observes such a pattern, a node that distinguishes itself from other nodes in some way (depending on his knowledge of the algorithms used in the network, this distinction can be crucial to security). Hence, applying Repeated Firework as a stand-alone algorithm in the proximity of such a special node and such an adversary, can lead the adversary to the node performing the fake rounds and leave the actual one untouched. Of course, increasing the number of these stand-alone fake patterns, will decrease the chances an adversary has to pick a real one, but also increase the number of messages used for this camouflage algorithm.

As the last argument implies, it is important to have an approximate estimate of the number of messages that the Repeated Firework notion incurs when it is executed. Looking at algorithm 3.6, the leading node s starts

[7]This calls for a slight modification of the Repeated Firework algorithm, namely the **initialization** event needs to drop the initial camouflage trigger, as well as the omission of the *fake_msg* as this is covered by an according modification of the other algorithm that is enhanced by Repeated Firework.

by sending a single $fake\_msg$ to its vicinity and causes each node therein to reply with an other fake message. Thus a single round incurs $r+1$ many messages, where $r := |V(s)|$ is the cardinality of the vicinity set or the number of nodes in the vicinity of s respectively. This patterns is repeated until the variable cur_round reaches zero, then the complete process comes to an end and the algorithm can be seen as having terminated. Thus the number of total messages is $cur\_round \cdot (r+1)$, giving an approximate message complexity of $\mathcal{O}(cur\_round \cdot r)$ for the Repeated Firework algorithm 3.6. Of course, because of the unreliability of the local broadcast primitive, assumed in the Data Security and Sensor Networks Chapter, or the usage of the p_r probability set to less than 1, the actual number of replies in each round will be approximately $p_{lb} \cdot r$ or $p_r \cdot r$ respectively. But it is assumed that the $\mathcal{O}$ notation accommodates for that fact, as the order of magnitude which is dominated mainly by r and cur_round.

3.2.2 Wandering Repeated Firework

The camouflage notion of Repeated Firework presented in the last section allows for a simple refinement similar to the one from Random Walk to Fractal Propagation, done in the section on Global Anti-Traffic Analysis techniques. The idea is to allow the firework of local broadcasts to wander in the network. This is realized by having the hot spot of the communication rounds, the leading node s, described in the previous section trigger nodes in its vicinity to start fireworks of their own. Naturally, such an approach needs to be controlled to avoid arbitrary dispersion of such fireworks in the network. Hence, a certain spreading factor $spread$ is used to define the dispersion of the Repeated Firework repetition and to limit the depth of the dispersion the maximum depth is given by K.

```
Participants:
    s               - current node
    v               - arbitrary node in the vicinity of s
Parameters:
    spread          - a spreading factor that specifies how many nodes should continue the firework
    K               - specifies the number of delegations (i.e. how deep the into the network the wandering can
                      get)
    r(d)            - specifies the number of repetitions, which can depend on the d
    len(d)          - specifies the size of fake message, which can depend on the d
    reply_len(d)    - specifies the size of fake reply message, which can depend on the d

s:    upon event initialization do
      -     invoke RFire($r(d)$, $len(d)$, $rlen(d)$)
      -     if $K > 0$ then
      -         randomly choose set $S := \{v_1, \ldots, v_{spread}\} \subseteq V(s)$
      -         decrement K
      -         point-to-point INVOKE_WRFIRE(spread,K,r,len,reply_len) to all $v \in S$
v:    upon event receiving INVOKE_WRFIRE(spread,K,r,len,reply_len) do
      -     invoke WRFire(spread,K,r,len,reply_len)
```

Algorithm 3.7: Wandering Repeated Firework (WRFire)

This notion is put into pseudo code in Algorithm 3.7, which simply calls Repeated Firework and makes the leading node s randomly choose $spread$ many neighboring nodes to propagate the firework. Of course, the algorithm limits the depth of the propagation of the fireworks by stopping as soon as K many spreading cycles have been completed.

Now that the algorithm for the Wandering Repeated Firework notion is stated the question arises what concrete applications can be addressed with such a camouflage technique. As the algorithm suggests, this is a simple extension to the notion of Random Walk that addresses algorithms exhibiting intense communication patterns between a single node and its vicinity. Imagining such algorithms their purpose is mostly not only to get some information from nodes in the neighborhood, but is more far reaching, meaning that several hops away nodes can be involved too. For instance, the algorithm might need a neighbor to provide information about his vicinity or just cause a neighbor to repeat the same algorithm again. There are surely sufficient scenarios where the pattern of the Repeated Firework is delegated to other nodes in the proximity of a node starting such algorithms. Thus the Wandering Repeated Firework is a simple extension that allows for such a spreading pattern using fake messages. Of course, the main applicability of Wandering Repeated Firework will be more clear, once the Evasive Data Storage techniques are presented in the following chapters. As those exhibit such a wandering and repetitive communication pattern. Furthermore, for the application of the Wandering Repeated Firework Camouflage, the *spread* parameter will be set to 1 only, but other applications might desire a higher spreading factor, thus in order to allow Wandering Repeated Firework to be applicable to other algorithms as well, the option for increasing this factor is given through the parameter *spread*.

As with the Repeated Firework algorithm it is important to estimate the message overhead that by using this camouflage technique is inflicted on the sensor network. To do so the number of nodes that will eventually start a Repeated Firework needs to be specified. The first node will elect *spread* many nodes to start a Repeated Firework, each of which will select *spread* many nodes again. This is delegation is repeated K many times, thus yielding effectively

$$\sum_{k=0}^{K} spread = 1 + spread + spread^2 + \ldots + spread^K = \frac{spread^{K+1} - 1}{spread - 1}$$

many calls to the Repeated Firework algorithm, given that $spread \neq 1$ of course. In the latter case, $spread = 1$ causes only for K many interactions. Summarizing, the message complexity for the $spread \neq 1$ case is $\mathcal{O}(spread^K \cdot cur\_rounds \cdot r)$ and for the $spread = 1$ case it is $\mathcal{O}(K \cdot cur\_rounds \cdot r)$. Of course, the exponential bound for the *spread* can be quite concerning, however it has to be kept in mind that the algorithm does not check if the invocation send to each node in the set S actually arrives at any of those nodes, because of the unreliability assumed for the sending procedure. Adding this uncertainty into the picture alleviates the explosion associated with exponential bounds, at least as long as K is not chosen to be excessively large. Furthermore, if the unreliability of channels is not sufficient and more control needs to be obtained of the spreading process, a spreading accept probability p_{sa} can be used. This probability is evaluated by each node, before it commits to contribute to the spreading of the Repeated Firework, therefore allowing the exponential explosion to be controlled more strictly.

This finishes the Local Camouflage techniques that will be of concern in the subsequent simulations and leaves a notion that is not specified to its fullest, but given as an example for another possible local camouflage technique with a completely different approach than any of the previous ones.

3.2.3 Self Adapting Pattern Imitation

The notion behind Self Adapting Pattern Imitation is to create an anti-traffic analysis mechanism that is applicable to almost arbitrary situations. Such an approach should work such that it does need not to be told how to create camouflage, but can rather realize and learn on its own, what fake messages should be sent and where to. On the spectrum of camouflage approaches such a mechanism would be situated between the simple Fake Packets notion, where no application orientation is integrated, and the highly application oriented approaches of Fractal Propagation or Repeated Firework. The Self Adapting Pattern Imitation idea is to use pattern recognition algorithms to find patterns in the traffic of a network and then initiate imitation of those either at the same location or in other parts of the network.

The basic recipe for the Self Adapting Pattern Imitation algorithm for each node s in the network is to perform the following steps:

1. keep track of messages passing through the vicinity, meaning the nodes that forwarded messages

2. propagate local observations of the vicinity, including location information of the nodes involved in the observations to all nodes at a distance of maximal *depth* hops

3.* accumulate observations received from other nodes and merge them to obtain a map of an excerpt of the sensor network, whose size depends on *depth*

4.* use a pattern recognition technique to extract message patterns from the created local map

5.* instruct a random node from the local map to imitate the observed pattern

However, the steps marked with a star are assumed to be executed only by a minor amount of special nodes. This is important to avoid all nodes in the network to issue imitation initiations and even more importantly to conserve computational power. The steps above of the Self Adapting Pattern Imitation notion are put into an algorithmic notation in algorithm 3.8. There are two kinds of nodes that are distinguished: the node s stands for nodes in the network that also execute the steps labeled with a star in the above enumeration and the node a only execute the steps that are not labeled. Such a node a can be seen as only providing data to the active node s, such that s can apply its pattern recognition to a larger region than its limited vicinity. As said before, the Self Adapting Pattern Imitation approach is presented not to its fullest, but rather as a notion to be explored in work beyond this thesis, thus the algorithm 3.8 contains several lines that need to be specified in more detail. Hence the algorithm cannot be implemented until those parts are completely specified; those missing parts include, for instance, the precise pattern recognition technique, which strongly influences the efficiency and accuracy of the approach. Furthermore, the exact structures, that are used for accumulation of observations or for the local map mentioned, are left unspecified as well.

The most significant drawback of this notion is that accumulating observations can cost large amounts of storage, which is a scarce resource for sensor node. Similarly, due to computational limitations a pattern recognition algorithm might not work sufficiently successful, leaving only base stations as possible nodes for pattern recognition and imitation initiation. It will remain for future work to find concepts that are efficient and effective, in order to fill in the gaps of the algorithm 3.8. And therefore, allow for a thorough evaluation of its effectiveness.

3.3 Summary

This section not only introduced and modified camouflage techniques, mainly the Fractal Propagation algorithm, that have already been published in the scope of other works, but also described a set of new approaches to the camouflage realm, namely the Repeated Firework and Wandering Repeated Firework algorithms. In the course of this chapter, also some of the complexity issues of the algorithms as well as their advantages offered in presence of adversaries have been discussed. Although the Fractal Propagation algorithm has been presented here in its original form, the generalization mentioned before will be presented later in scope of the Simulations Chapter and the Appendix. Summarizing, this chapter presented various Camouflage algorithms and allows the next chapter to provide insights into a completely different approach to address node capture, namely to introduce the core part of the thesis the Evasive Data Storage notion. Both techniques will be evaluated in detail towards the end of the thesis in the Simulations Chapter.

Participants:	
s	- a node that can trigger an imitation
a	- arbitrary node participating in the Self Adapting Pattern Imitation mechanism
Parameters:	
$depth$	- specifies the depth of the graph that is build up for pattern recognition
msg	- message that is overheard
$time\_int_1$	- specifies time interval to wait before next update is done
$time\_int_2$	- specifies time interval to wait before next pattern recognition phase is started

```
          // SAPI - Local Phase Active Nodes
$s$:      upon event  initialization  do
          -    initialize structure for local map $map$ adhering to $depth$
$s$:      upon event overhearing message $msg$ do
          -    incorporate $msg$ into $map$
$s$:      foreach $time\_int_1$ do
          -    propagate local $map$ to nodes $depth$ hops away
          // SAPI - Incorporate adjacent observations
$s$:      upon event receive $map'$ from some node $a'$ do
          -    incorporate received $map'$ of $a'$ into local $map$
          // SAPI - Trigger imitation
$s$:      foreach $time\_int_2$ do
          -    recognize patterns in local $map$ and generate imitation instructions $Imitation\_Inst$
          -    randomly choose node $a''$ mentioned in $map$
          -    point-to-point $Imitation\_Inst$ to $a''$
          // SAPI - Local Phase Passive Nodes
$a$:      upon event  initialization  do
          -    initialize structure for local map $map$ adhering to $depth$
$a$:      upon event overhearing message $msg$ do
          -    incorporate $msg$ into $map$
$a$:      foreach $time\_int_1$ do
          -    propagate local $map$ to nodes $depth$ hops away
          // SAPI - Execute Imitation
$a$:      upon event receiving $Imitation\_Inst$ do
          -    send imitation messages as described in $Imitation\_Inst$
```

Algorithm 3.8: Self Adapting Pattern Imitation (SAPI)

Chapter 4

Data Oriented Techniques

This chapter introduces the main technique to address the Node Capture Problem from a Data Security perspective. After having seen the camouflage algorithms in the last chapter, which intend to hinder adversaries with traffic analysis abilities from *finding* interesting nodes, this chapter focuses on the other possible approach an adversary can take to obtaining valuable data: *remember* interesting nodes. This is done by introducing a completely new perspective on how data can be stored in sensor networks. The Data Security and Sensor Networks Chapter of this thesis gave a glimpse of the current approaches of data storage in sensor networks, which makes it easier to grasp the advantages of the new storage method when taking into account the presence of an adversary.

The chapter will start with a basic introduction into the notion of Evasive Data Storage along with the security property that will be used to judge the procedures developed in this chapter and then dive into the presentation of algorithms that abide by that notion. The emphasis is on structural development of the idea, meaning that a simple algorithm is taken, and then refined to address problems that emerge along the way. Besides, a mere presentation of the algorithm several aspects regarding correctness and security properties will be investigated such that less emphasis has to be laid on those, when the Simulation Chapter investigates the security property thoroughly.

4.1 Data Evasiveness Notion

The idea behind Evasive Data Storage can be put in quite simple terms. Distinguishing between the possibility of fixing the h hot nodes and enabling those h nodes to vary over time, gives rise to data evasiveness. However this is different from the movement of data seen, for example, in Data Centric Storage, where data is moved to other nodes for storage, because once data reaches that final storage node, the node will remain hot. Data evasiveness, on the other hand, is the process of intentional shifting of data as part of long term storage, i.e. hot nodes can loose their status of being hot as time passes by, in order to avoid the illegitimate retrieval of data. This is essential to address the problem of adversaries that have spotted and remembered hot nodes, for example by successfully doing data analysis or temporarily subscribing to a sensor network as a legitimate user. In conventional storage, like Data Centric Storage, an adversary that located hot nodes can repeatedly access those nodes in order to obtain updated data. Such an adversary will have a success probability very close to 1, as Data Centric Storage will always anticipate to store data of a certain type at the same node. Thus an adversary that wants to access updated data, he once found and remembered to be stored at a certain node, can do so with almost guaranteed success. The only problem, which might drop his probability of success, is when the node failed. However due to Data Centric Storage's redundancy mechanism, replications of data have been located at nodes closest to the failed node, thus again making life easy for the adversary, who can just pick close nodes and have a very high probability of being successful. Hence, an Evasive Data Storage algorithm alleviates that guaranteed success of an adversary that remembers hot nodes, by moving data around. As an adversary approaches a node that by his knowledge used to store data, the Evasive Data Storage algorithm will have moved data in several displacement steps away from that node, thus such an adversary will not succeed by simply trying out close neighbors of the former hot node. This substantially reduces the success probability of such an adversary to below 1. Of course, the exact details depend on the strength of an adversary and which of the presented Evasive Data Storage mechanisms is exactly used. The achieved advantage will be explained in some detail when the different mechanisms of Evasive Data Storage are presented in the following sections and most detailed in the Simulations Chapter, where the algorithms are analyzed thoroughly.

Although the notion of Evasive Data Storage has the obvious advantages described above, it is important to keep in mind that the storage algorithm will be used in sensor networks, which have severe limitations regarding available energy. Thus the Evasive Data Storage algorithms need to be efficient in terms of number of messages exchanged and computation needed. Of course, for the increased security offered by the Evasive Data Storage algorithms a price needs to be paid in terms of communication overhead. However, with wisely chosen parameters of the algorithms, a too wasteful overhead can be avoided (most easily by keeping the time intervals between displacement of data long enough). Further criteria for the design of Evasive Data Storage algorithms, are the stated advantage of adversaries being able to recognize hot nodes and provide persistency of data in the presence of node failure. Naturally, as Evasive Data Storage is a storage mechanism for sensor networks, it has to comply with the properties needed for a useful Data Storage that were explained earlier: **Availability**, **Correctness** and **Freshness**. Furthermore, another property is addressed by the notion of Evasive Data Storage which is the security that Evasive Data Storage wants to achieve. In more formal terms the additional property is:

- **Evasiveness Security Property:** If adversary $\mathcal{A}$ knows that node s was hot at time t, then $\mathcal{A}$ cannot identify a hot node in the region $R_{eds}(s, t')$ surrounding node s with probability greater than p_{eds} at any time $t' > t$.

This security property dictates that the storage algorithm does not allow an adversary $\mathcal{A}$ to easily distinguish between cold nodes and hot nodes and includes a bound in form of the probability p_{eds} into that property. Thus a storage algorithm wants to prevent an adversary from having success probabilities arbitrary high or equal to one. A quite significant role is the mentioning of the region $R_{eds}(s, t')$ in the Evasiveness Security Property, as $\mathcal{A}$ will naturally assume that at time t' the number of evasion steps is limited, thus yielding a limited region that can be ascribed to hold the evaded data. Hence, $\mathcal{A}$ can limit his efforts to a smaller region than after a larger period of time $t'' \gg t$, at which he would have to consider a larger part of the network instead. It should be clear that an adversary $\mathcal{A}$, that remembers that a node was hot at time t and the network does not evade data from hot nodes, can access that node and the data contained within it at any time $t' > t$ with a success probability of 1. Such an algorithm thus, certainly does not comply with the **Evasiveness Security Property** stated above, and it is important to mention that all conventional storage methods (like Local Storage, etc.) do have the tendency to grant access to data at any time $t' > t$ with probability 1. Assuming that the network harbors h many hot nodes, the adversary can always pick a node totally at random or equivalently guess a node, resulting in a minimum success probability[1] of $\frac{h}{n}$. Hence, an algorithm that anticipates to fulfill the **Evasiveness Security Property** needs to push the probability p_{eds} as close as it can to $\frac{h}{n}$, which forces adversary $\mathcal{A}$ to behave as if he had to guess which node is hot.

4.2 Simple Evasive Data Storage

The easiest approach that can be taken to Evasive Data Storage is to take a look at a conventional data storage algorithm, like Local Storage, and then introduce the controlled movement of data into the picture. A network can be assumed to have h hot nodes, which corresponds to h nodes that sensed data and stored it locally in their memory or (more realistically) h aggregator nodes that fused data of several local nodes and then stored it locally. Either way, there are h hot nodes that store data, and those nodes will remain fixed. However, adding controlled movement to the mechanism allows each of those h hot nodes to give up its hot status by displacing its locally stored data to another node. Assume $\tilde{a}$ is a hot node storing data D and decides to move D. Thus $\tilde{a}$ chooses a node $v \in V(\tilde{a})$, transmits D to v and deletes D from its storage. After that evasive step, the hot node $\tilde{a}$ becomes a cold node a and the cold node v becomes the new hot node $\tilde{v}$. At this point D resides at $\tilde{v}$

[1]This probability will be will be explained later in this section.

until the node decides to move D, this cycle is repeated indefinitely over the lifetime of the sensor network and causes D to wander in the network. This basic idea of how to create an Evasive Data Storage algorithm is fully developed in the Simple Evasive Data Storage algorithm, which will be described in the rest of this section in more detail.

Participants:	
s	- the node storing data to be moved (leading node)
Parameters:	
$time\_int$	- defines time interval between checks if to trigger the data evasion
p_e	- the probability that displacement of data takes place
D	- the data item which is to be moved

```
s:    upon event receiving or generating D  do
      -     start timer time_int
s:    foreach time_int  do
      -     with probability p_e  do
      -         invoke SEDS-NSNC(D)
s:    upon event successful evasion  do
      -     stop timer time_int
```

Algorithm 4.1: Simple Evasive Data Storage Algorithm (SEDS) - Displacement Trigger

The starting point for the description of Simple Evasive Data Storage is the node s that generated or received a data item D and at that point also started a so called Displacement Trigger, which is assumed to be run as long as s is storing D. The code for the Displacement Trigger is shown in algorithm 4.1. Besides the parameter that specifies which D the trigger is responsible for, two parameters are used for algorithm 4.1 and are controlling how often evasion of D occurs: the $time\_int$ and the p_e parameters. The $time\_int$ specifies a time interval that needs to elapse before the evaluation of p_e begins, which specifies how probable an evasion is. Both parameters are needed for the control of the displacement process and can be seen to constitute a time interval of random length to wait until evasion is attempted.[2] Thus once the trigger decides to attempt data evasion, algorithm 4.1 invokes the actual core of the Simple Evasive Data Storage algorithm, the New Storage Node Choice part. In case the evasion is successful, which is triggered by the New Storage Node Choice by raising a *successful evasion* event, the Displacement Trigger is stopped by stopping the timer that causes $time\_int$ to be evaluated.

Now that the trigger of the evasion has been clarified the next step is to describe in detail the actual evasion process, which Displacement Trigger invokes. First off, consider two fundamentally different approaches to this problem. On one hand, the node s, which currently stores the data item D, can choose which of its neighbors should be in charge of storing D, the term *local choice* denotes such a process. On the other hand, s can leave the choice to the neighbors themselves, thus allowing them to decide who will store the D, denoted as the *remote choice*. Intuitively, having s not know where the data actually was shifted to, might seem to be more secure, as an adversary cannot obtain information about the path by opening s and extracting all data stored, therein the information about where data was displaced to. However, this might to true to some extend, but following arguments advocate the local choice:

- *the identification* of the node where D was displaced to and which is known to s does not enable an adversary to directly associate it with a physical node.

- *non-malicious node* s does not diverge from the algorithm it should execute, therefore can be forced not to store any information related to the new data hosting node (for instance, its identification).

[2]This random interval is bounded from below by the $time\_int$ value.

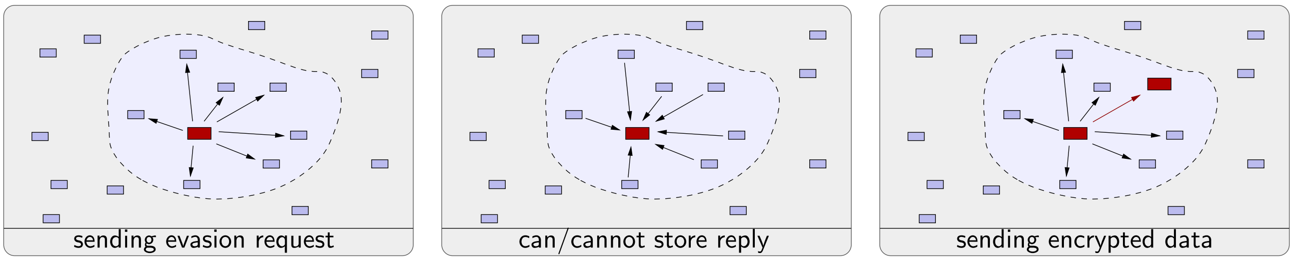


Figure 4.1: Simple Evasive Data Storage - New Storage Node Choice Steps

- *communication overhead* can be large when considering remote choice approach, as either the data needs to be send to all nodes in $V(s)$ unencrypted, simple broadcast to all, or it has to be send to each node separately but encrypted. The unencrypted possibility keeps energy consumption low, but leaves the data unprotected and an adversary could eavesdrop the transmission. The encrypted possibility is more secure, but burdens the energy household of the participating nodes strongly. Even when an encryption key is used that is known by all nodes, once the adversary successfully extracts all information contained in one node storing such a global key, the situation is the same as if no encryption was used.

Thus the Simple Evasive Data Storage uses the local choice, which makes the s the decision maker of where to displace its D. The algorithm does not need s to store the identification of the new node, where D was displaced to, hence s does forget the new node and the contents of D after the transmission is successful. The basic steps have already been described vaguely at the beginning of this section, now a detailed description is given. The three basic steps are depicted in figure 4.1, where the left picture shows the first one, the *Evasion Request*. This step consists of a local broadcast of the node s wanting to displace its data item D and intends to query which nodes are capable of storing D. This request also contains the size of the data to be stored, thus allowing each node $v \in V(s)$ to respond adequately in the second step, the *Store Reply* depicted in the middle picture of figure 4.1. The vicinity nodes answer the request of s by replying, if they can store or cannot store D. With that information s chooses one of the nodes that replied positively to its request and sends D encrypted to that chosen node, the *Data Transmission* step (right picture).

The pseudo code describing the New Storage Node Choice is presented in algorithm 4.2. The local choice process is started when the New Storage Node Choice is called by the Displacement Trigger. Upon invocation s sets up two sets, C that will contain the nodes that reply positively and M for the nodes that reply negatively to the *REQUEST_EVADE* that is locally broadcasted along with the size of the D that is to be displaced. Each v that receives the request and replies with *CAN_STORE*, which leads to the addition of v to the set C by s, or replies *CANNOT_STORE*, which leads to the addition of v to the set M. The displacing node s waits until all responses have arrived or until a timeout is reached, which is necessary because of the possibility that messages do not arrive either on the way to the neighboring nodes or during the reply stage. For the former, it is assumed that each node a has an adequate knowledge regarding which nodes belong it its vicinity $V(a)$, which is provided by the assumed underlying routing algorithm that supplies the node with both the neighbors and their respective location coordinates needed for adequate routing. Having reached this stage of the displacement process, s now has to choose from the nodes, whose positive reply was received. To do this, several possibilities are available, hence in algorithm 4.2 a parameter is used, denoted as *Choice*. This parameter is assumed to represent a choice function that, given a set of nodes, returns a single identification of a node from that set or $\times$, in case it does not make a choice (for instance, the given set is empty). This *Choice* function is called with the set C as parameter and, if successful, returns the new storage node ID, where the data D is transferred to. Upon receiving D, the new storage node ID replies with a *STORED* or *STORED_FAILED*[3] acknowledgment,

[3]This can be used if the node ID encountered an error and was not able to store D successfully, this can be considered redundant,

in order to inform the s of successful storage. In a positive case the s can dispose of D, as another node has stored it successfully. If the evasion of D is not completed, either due to a *STORED_FAILED* message or due a timeout, the Displacement Trigger is informed with a *failed evasion* event and therefore restarts the evasion attempt after some time interval. The important part of the algorithm 4.2 that is left unspecified, i.e. the *Choice* function, will be explored in more detail at the end of the section. For now, it suffices to assume that *Choice* selects a node from the set given to it.

Participants:	
s	- the node storing data to be moved (leading node)
v, ID	- arbitrary node in the vicinity of s

Parameters:	
Choice	- this function defines what node is chosen from $V(s)$ to hold the data
time_out	- timeout till nodes that did not respond are ignored

```
s:      upon event invocation do
        -     set $C = M = \emptyset$
        -     local broadcast $REQUEST\_EVADE$(sizeof D)
        -     start timer $time\_out$
s:      upon event receiving $CAN\_STORE$ from $v$ do
        -     $C = C \cup \{v\}$
s:      upon event receiving $CANNOT\_STORE$ from $v$ do
        -     $M = M \cup \{v\}$
s:      upon event $C \cup M = V(s)$ or $time\_out$ is reached do
        -     invoke $ID = Choice(C)$
        -     if $ID \neq \times$ then
        -         encrypted local broadcast $D$ to $ID$
        -         start timer $time\_out$
        -     else
        -         raise failed evasion
s:      upon event receiving $STORED\_FAILED$ from $ID$ or $time\_out$ is reached do
        -     invoke SEDS(D)
        -     raise failed evasion
s:      upon event receiving $STORED$ from $ID$ do
        -     raise successful evasion
v:      upon event receiving $REQUEST\_EVADE$(sizeof D) do
        -     if sufficient storage to store data then
        -         encrypted local broadcast $CAN\_STORE$ to $s$
        -     else
        -         encrypted local broadcast $CANNOT\_STORE$ to $s$
ID:     upon event receiving $D$ do
        -     if $D$ is stored successfully then
        -         encrypted local broadcast $STORED$ to $s$
        -     else
        -         encrypted local broadcast $STORED\_FAILED$ to $s$
```

Algorithm 4.2: Simple Evasive Data Storage Algorithm - New Storage Node Choice (NSNC)

Now that the core procedure of choosing where to displace a given data item to has been explored several issues still need to be addressed. The first one is directly concerned with the just described New Storage

as the timeout maintained by s could account for such a case, nonetheless this adds to the flexibility of the algorithm.

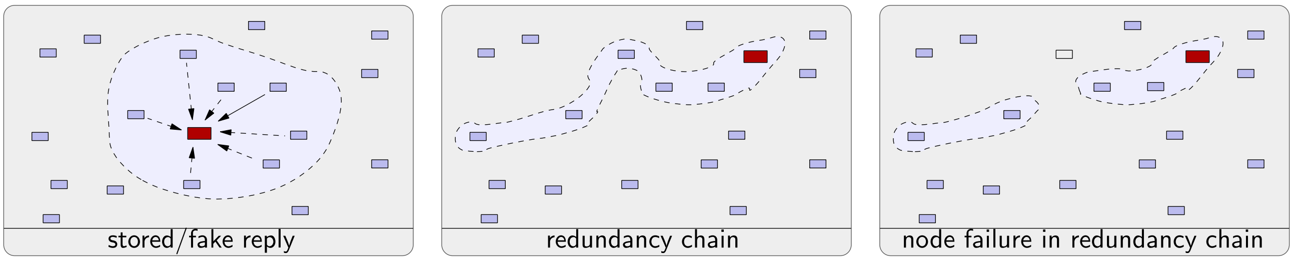


Figure 4.2: Simple Evasive Data Storage - Advanced Algorithm Parts

Node Choice algorithm. The last step where the chosen node ID replies to s whether it stored the data item D successfully is problematic, when imagining an adversary $\mathcal{A}_l(local, \star)$ that observes this data exchange. Without performing any sophisticated traffic analysis, $\mathcal{A}_l$ can identify the node that is now storing D, as it is the only node that performs a message transmission at this stage of the algorithm. Thus a camouflage has to be added. The algorithm 4.3 shows how this can be achieved, by proposing an addition to algorithm 4.2. This addition forces all nodes in the vicinity of s to send a *STORED_FAKE*, hence disallowing $\mathcal{A}_l$ to spot the new storage node ID easily. Instead of including this modification directly into the algorithm, it is possible to resort to a camouflage technique that has already been introduced in the preceding chapter. The Repeated Firework algorithm is useful in this context, the algorithm can be triggered by the s (one round only) as part of the displacement message, thus triggering all nodes in its vicinity to respond, which essentially leads to the same behavior that the algorithm 4.3 adds to the New Storage Node Choice algorithm. This camouflage enhancement is illustrated in the figure 4.2 (left picture).

```
Participants:
    s          - the node that displaces a data item
    v          - arbitrary node in the vicinity of s, unequal to ID

    v:         upon event overhearing completed displacement of data from s to ID do
               -     encrypted point-to-point STORED_FAKE to s
```

Algorithm 4.3: Simple Evasive Data Storage Algorithm - Camouflage Enhancement

The other issue that is important in context of Evasive Data Storage is the need to address an innate property of sensor networks: the failure of sensor nodes. The reasons for possible failure were described in the Background Chapter and currently only those are of concern that preclude adversarial intervention, i.e. the notion of **node failure** as introduced in the formal model. During the wandering of essential data items, like D, only the node s is assumed to store the data item. Thus if s happens to fail and thus looses D, there would be no way to retrieve D anymore. Therefore, some kind of redundancy mechanism is needed to provide Simple Evasive Data Storage as it is now with resistance against such failures.

A very straightforward way to achieve redundancy in Simple Evasive Data Storage is to force the node that generated a data item D to send it to several nodes in its vicinity that will then store the same data item using the algorithm presented so far. In such a scenario, there would be several nodes storing D and performing evasion, i.e. several *redundancy copies* of D will be wandering in the network. A more elaborate design is using the notion of a *redundancy chain*. A redundancy chain is a list of nodes $(a_1, \ldots, a_l)$ that have the common property of storing the same data item, where each a_i is assumed to have been chosen by a_{i-1} during the execution of the New Storage Node Choice algorithm. Hence, such a chain consists of nodes that are pairwise in the vicinity of each other and serve to store a data item with redundant copies. Figure 4.2 illustrates such a setup in the network (middle picture). This is similar to a notion developed in [21] for mobile agents. However, the redundancy chain investigated here is focused on protection of data only. The basic idea of a redundancy chain is to maintain l nodes that store the same data item D, such that in case a node fails, the network can still retrieve D through the redundant copies maintained by the chain. The working of the Simple Evasive Data Storage algorithm introduced so far, is that a node s that generated some data item D chooses another node v and displaces D to v. Afterwards s deletes D as the new node v is now in charge. Now in order to provide redundancy this approach is changed, and instead of allowing s to delete D it becomes part of the redundancy chain, thus storing D until the evasion is repeated l many times. After that many evasions, s is the last node in the chain, l other nodes provide sufficient redundancy, and hence in order to keep the length l of the redundancy chain constant, s is unlinked and allowed to delete D.

```
Participants:
    $s$           - a node participating in the redundancy chain
Parameters:
    $D$           - the data item that is protected by the redundancy chain
    $l$           - length of the redundancy chain, i.e. spots in $Chain$

    $s$:          upon event generating $D$ do
                  -     set $Chain = (ID(s), \times, \ldots, \times)$
                  -     set $D_c = (D, Chain)$
                  -     initialize SEDS Displacement Trigger
    $s$:          upon event invoke SEDS do
                  -     invoke $v = SEDS(D_c)$ restricted on $V(s) \setminus Chain$
                  -     if SEDS successful and $v$ not found in $Chain$ then
                  -         set $Chain = ID(v) \to Chain$
                  -         foreach $a$ found in $Chain'$ and $a \neq s$ do
                  -             point-to-point $NewHead(v)$ to $a$
    $s$:          upon event receiving $(D, Chain')$ do
                  -     set $Chain = ID(s) \to Chain'$
                  -     set $D_c = (D, Chain)$
                  -     initialize SEDS Displacement Trigger
    $s$:          upon event receiving $NewHead(v)$ from $\mathbf{H}(Chain)$ do
                  -     if $v$ not found in $Chain$ then
                  -         set $Chain = v \to Chain$
                  -         if $ID(s)$ not mentioned in $Chain$ then
                  -             quit participation in redundancy chain and free $D$
```

Algorithm 4.4: Simple Evasive Data Storage Algorithm - Redundancy Chain

The precise working of such a chain is presented in algorithm 4.4. This mechanism makes use of the New Storage Node Choice and Displacement Trigger algorithm respectively, to allow the chain to proceed through the network. Algorithm shows the additions that each node participating in such a redundancy chain needs to perform, where participation of a node a starts when data is displaced to a and ends with the evasion step at the time when a is the last link in the chain. The algorithm is written from the perspective of a participating node s, where special attention needs to be given the part that describes the actual setup of the chain during the generation of the protected data item D, i.e. the part that is only executed by the first node to store D. This part consists of setting up up the chain $Chain$ to a list $(ID(s), \times, \ldots, \times)$ that has l fixed storage spots, and as shown is initialized to contain the identification of the generating node as first element in the list. This spot, i.e. the head of the list, will be occupied by the node that is running the New Storage Node Choice and the Displacement Trigger part of the Simple Evasive Data Storage algorithm and will thus eventually move the chain one link further by evading D to a node node. Note that the algorithm prevents a node that is already in the chain to be chosen by New Storage Node Choice to become new head of the chain. If the new head is chosen, it obtains along with the actual data item D, the current chain $Chain$ and inserts its identification into the first spot of $Chain$. Analogously, the old head modifies the chain and propagates the new chain with the new head to all the nodes that actively participate in the mechanism. If a node participating in the redundancy chain receives a new $Chain$ from the old head, it checks if its own identification is still contained in the list, if not the node stops to participate in the chain and deletes the respective data item D. This algorithm thus provides a chain that moves forward in the network using the New Storage Node Choice and maintains its predefined length by automatically unlinking nodes.

```
Participants:
    $s$                 - a node participating in the redundancy chain
Parameters:
    $time\_int$         - time interval for heartbeat messages
    $time\_out$         - timeout till a node is considered to have failed in the chain ($time\_out << time\_int$)

    $s$:    upon event  initialization  do
            -    start timer $time\_int$,  set $connection\_check = false$
    $s$:    foreach $time\_int$ do
            -    if $connection\_check = false$ then
            -        search for $a$ such that $Chain = (\diamond, \ldots, \diamond, ID(a), ID(s), \diamond, \ldots, \diamond)$
            -        if $a$ is found then
            -            initialize $link\_checked$
            -            set $link\_checked = a \rightarrow link\_checked$
            -            point-to-point $HM$ to $\mathbf{H}(link\_checked)$
            -            start timer $time\_out$,  set $connection\_check = true$
    $s$:    upon event receiving $HM'$ from any node in $link\_checked$ do
            -    set $connection\_check = false$
    $s$:    upon event no $HM'$ is received from $\mathbf{H}(link\_checked)$ or $time\_out$ is reached do
            -    set $a$ equal $\mathbf{H}(link\_checked)$
            -    search for $a'$ such that $Chain = (\diamond, \ldots, \diamond, ID(a'), ID(a), \diamond, \ldots, \diamond)$
            -    if $a'$ is found then
            -        set $link\_checked = a' \rightarrow link\_checked$
            -        point-to-point $HM$ to $\mathbf{H}(link\_checked)$
            -        start timer $time\_out$
            -    else
            -        crop $Chain = (\diamond, \ldots, \diamond, ID(s), \diamond, \ldots, \diamond)$ to $(ID(s), \diamond, \ldots, \diamond, \times, \ldots, \times)$
            -        foreach $a''$ found in $Chain$ and $a'' \neq s$ do
            -            point-to-point $NewChain(Chain)$ to $a''$
            -        reinitialize as new redundancy chain head
    $s$:    upon event receiving $HM$ from $a'$ do
            -    if $a'$ contained in $Chain$ then  point-to-point $HM'$ to $a'$
            -    else  point-to-point $DROP\_OUT$ to $a'$
    $s$:    upon event receiving $DROP\_OUT$ from any node in $link\_checked$ do
            -    quit participation in redundancy chain and free $D$
    $s$:    upon event receiving $NewChain(Chain')$ from any node in $\mathbf{T}(Chain)$ do
            -    if $Chain'$ is sublist of $Chain$ then  set $Chain$ to $Chain'$
```

Algorithm 4.5: Simple Evasive Data Storage Algorithm - Redundancy Chain Timeout Extension

There is still an issue that has not been investigated, but is essential to ensure the right behavior of Simple Evasive Data Storage. The redundancy chain needs to address the possibility of failures of nodes participating in the chain correctly. This is particularly important when the head of the chain fails and thus could cause the movement of the redundancy chain to stop completely. A simple way to overcome this failure problem and ensure movement of the chain is the usage of heartbeat messages. Those messages are send by the nodes in the chain in order to keep track of the functioning of the links. For instance, if a node realizes that all upper links including the head of the chain have failed, it should take over as new head and proceed with the evasion process. The extension to the algorithm 4.4, which deals with such messages is described in detail in algorithm 4.5. The basic method employed is that each link in the chain s, except for the head, sends a HM message to the node that precedes s in the *Chain* and waits for a reply heartbeat HM'. In case a node s does not receive a reply it works its way up to the head of the chain sending HM messages to each node listed in its *Chain*. If s does not receive any replies, even not from the head of the chain, it elects itself to be chain head and begins to progress the chain on its own. This approach keeps the amount of communication down as the heartbeat messages are send to very close nodes, and even if a heartbeat does not arrive at the first attempt, a node needs at least several checks before it reaches the head and might choose to lead the chain on its own. Even in case that an undesirable split of the chain occurs, this will only increase the redundancy for the respective data item. This algorithm completes the needed ingredients for the Simple Evasive Data Storage algorithm.

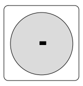
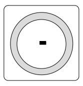
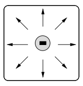
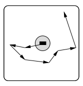
Choice Function	**Visualization**	**Description**
UVicinity		This choice function picks an arbitrary node in the vicinity, where each node has equal probability of being chosen.
UFurthest$k$		The choice function UFurthest$k$picks with uniform probability one node that is among the $k$furthest nodes located in the vicinity. In order to do so, the function puts the nodes in the vicinity in descending order of distance into a list and then picks form those nodes that find themselves in the $k$first spots.
GNodeFurthest		This choice function's goal is to move data as fast as possible from the node that generated it. Thus at each choice the node in the vicinity is picked that is the furthest from the initial node that generated the data. To do so, the choice function uses the location of the generating node, which is either send along or can be extracted from the fields of the data item (see *Gnid* field specified in the Data Security and Sensor Networks Chapter).
DirectionV		A more sophisticated choice function that surely needs to attach additional information to the data item it moves, is the DirectionV choice function. The notion is that the evasion process picks initially a random direction and pursues that direction thereafter. However, with a low probability p_c, the direction can be changed again in a random fashion. In case a direction cannot be pursued, for instance when hitting the border of the sensor network, a random node is picked.

Table 4.1: Choice Functions

The only issue that has been left out are the details regarding the *Choice* function that is one of the most important elements of the New Storage Node Choice algorithm and thus also of the other algorithms introduced as parts of Simple Evasive Data Storage, particularly the Redundancy Chain one. Table 4.1 shows several possible choice functions that will be used in the scope of this work, naturally the modular structure of the algorithms presented here allows for arbitrary extensions, in the sense of additional choice functions with varying properties. The only restriction is that choice functions need to select a node from the set passed to them and in case they cannot select a node (for instance when the set they were given is empty), they need to return $\times$. The table 4.1 gives the description, which is used to refer to a particular choice function, a visualization of the region that the choice function uses (it is assumed that the complete vicinity of a node is passed to the choice function, whereas in the algorithm discussed previously a subset of this vicinity is used, this accounts for more clarity in the description of choice functions) and a short description. The table shows two simpler mechanisms, the first two functions, and two more complex ones, the last two. Although the descriptions provided in table 4.1 might leave details out, the pseudo code is omitted, but interested reader can consult the implementation code used for the simulations, in case a more precise description is wanted. In order to simply obtain new choice functions, the notion of a simple combination mechanism will be given. When wanting to combine strengths of two choice functions or to alleviate a deficiency of another, it is allowed to form a combination of the functions given in table 4.1. The notation for a combination will be $(Choice_1, e, Choice_2)$, where the first and third component give the two choice functions that are to be combined, of course the order is important, meaning that the $Choice_1$ is used first and afterwards the $Choice_2$ is employed. The value represented by e specifies the number of times the first function is to be used and thereafter only the second one is employed. For instance, the choice function $(\text{GNodeFurthest}, 7, \text{UFurthest}k)$ will cause seven choices by the GNodeFurthest function, before the evasion is continued arbitrary long with UFurthest$k$. Of course, there are myriad of different combinations possible, but this thesis will only consider functions given in table 4.1 or the combinations resulting from the scheme just presented.

Now that all parts of Simple Evasive Data Storage have been introduced the question of data retrieval in this storage method is investigated in the next section. Afterwards the correctness and complexity are discussed in their respective sections as well as some of the security properties in presence of an adversary, which will finish the investigation of the Simple Evasive Data Storage algorithm in the scope of this chapter.

4.2.1 Data Update and Retrieval in Simple Evasive Data Storage

Now that the fundamental displacement of data and the inevitable data redundancy mechanisms are accounted for, the sole question that remains to be addressed is concerning data retrieval by a legitimate user. Data Evasiveness approaches have the intrinsic property of dislocating data over time. Therefore, there is no easy way to either access or update a certain data item, because its location can vary drastically. Approaches like *External Storage* or *Data Centric Storage* have the common feature that the location of data can be deduced with almost no effort, the Evasive Data Storage notion on the other hand does not specify the location where data can be found, but rather intentionally changes this location.

Hence, either retrieval and respectively update of data is simply realized by *flooding* the complete network to perform the operations or more elaborate solutions have to be thought of that must involve some kind of bookkeeping. There are two basic directions which can used to achieve this bookkeeping:

- *Base Station based Location Marking*: The base stations that are found in the sensor network are used to store the approximate location of an data item. Such an approach needs to take into account the frequent movement of data and hence involves numerous updates of the location of the evasive data. Doing so for every evasive step implies a rigorous monitoring of data items by the base stations, but also a quite large and inefficient amount of data exchange. Therefore, keeping in mind the redundancy approach taken in the Simple Evasive Data Storage Algorithm it is possible to just keep track of one link/node in the redundancy chain in order to guarantee data retrieval. To do so the base stations are informed about the current location of a data item every $f \approx l/2$ evasion steps, so that the base station has fairly good information about the residence of the redundancy chain and the data item it represents. Of course keeping the base station up to date can be done more frequently than $l/2$, but would also cause more message overhead. Naturally, there are different improvements possible for increasing protection against failures, for instance not only sending information to the base station about one member of the chain, but a triple consisting of several elements of the chain. Even a complete snapshot of the current chain list could be send to the base station, which consequently demands greater storage capacity on behalf of the base stations as well as greater communication overhead, but of course can increase protection against possible failures and thus unavailability of data.

- *Local Mark based Location Marking*: Certain nodes are chosen to become *local marks* that keep track of data motion by acting like intermediate nodes between base stations and hot nodes. Base stations query local marks instead of hot nodes and hot nodes contact local marks instead of base stations during displacement of data. Doing this creates a rather dense mesh of such marks which would keep the path length of location update messages, send by an data item's chain, small. Those local marks need to be set up before the network commences to operate. Furthermore, each node needs to know where the next local mark node is, in order to communicate with it. If a legitimate user queries the network, each of these local marks might store information about the current location of the queried data item, hence all base stations are equally required to know where the nodes functioning as local marks are located. Compared to the base station centered approach this one adds an intermediate layer of nodes, the local marks. Such a hierarchy can reduce the amount of communication required compared to the approach above, but only when the proportions of local marks and base stations are chosen correctly. A too dense deployment of local marks will result in almost flooding the network when a legitimate user queries for some data. A too sparse deployment of local marks will be essentially equivalent to Base Station based Location Marking. A major disadvantage is that local marks are just ordinary sensor network nodes, and hence subject to frequent failures. Those failures, even if occurring only infrequently to local marks can influence large parts of the network, namely the part that relies on that particular local mark, which then has to be covered by a neighboring local mark.

The first data query algorithm that is investigated is the simple notion of flooding. This is also the approach that is favored to be employed in case the Simple Evasive Data Storage Algorithm makes use of the notion of the *Redundancy Copies* redundancy mechanism that was mentioned earlier. Naturally, the flooding approach also applies to the notion of *Redundancy Chains* and guarantees that all nodes in a chain will be used to provide the data item, which can have advantages to the security of the data (this is similar to reporting the complete chain list to the base station and allow them to query every node of a chain for data). The basic idea of flooding is to have the base station BS that needs to obtain some data item of a given type $type\_spec$ to contact each node in the network, asking if it might have a data item matching the query. If several versions of the same data item are available, BS will receive a number of replies and needs to pick the freshest one.

Algorithm 4.6 shows the pseudo code for this data retrieval method. Note that all data retrieval algorithms are presented from the perspective of a single base station BS, of course the extension to arbitrary number of base stations is not problematic, but will not be of concern here. The algorithm makes sure the BS initializes a list structure DS that will store incoming data items matching the type specifier $type\_spec$ desired by the legitimate user LG. The base station initiates the process of data retrieval by **global broadcast**'ing the $REQUEST\_DATA(type\_spec)$ request that specifies the type of needed data such that nodes can respond only when they store a matching data item, meaning that the line forcing even cold nodes to respond with $\square$ can be dropped, if no particular reasons for not doing so are provided. A hot node $\tilde{a}$ that stores a matching data item D sends it back to the base station or can reply with a $\square$, if its data does not match. In any case, the base station waits until all queries have been answered or until a timeout is reached, which of course needs a carefully chosen timeout to work properly. After any of the conditions is met, all responses are evaluated according to their Ts field and the freshest data item received, F, is send to the legitimate user LG.

The next notion that is investigated is the *Base Station based Location Marking*. Here the assumption made is that the Redundancy Chain mechanism is used, thus it is assumed that a data item is stored in a chain wandering the network. However the approach can also be used in the Redundancy Copies mechanism, where at each evasion step an update is send to the base station, however in this case the pseudo code presented subsequently has to be slightly modified. The additional assumptions that are required but not included

```
Participants:
    $BS$           - base station issuing the query
    $LG$           - legitimate user querying for data
    $\tilde{a}$    - node that uses Simple Evasive Data Storageas storage mechanism
Parameters:
    $type\_spec$   - the type specifier that a legitimate user queries for
    $time\_out$    - time to wait for responses of nodes

$BS$:   upon event request for data of $type\_spec$ from $LG$ do
        -   initialize $DS$ for storage of data of type $type\_spec$
        -   global broadcast $REQUEST\_DATA(type\_spec)$
        -   start timer $time\_out$
$BS$:   upon event receiving $D$ matching $type\_spec$ do
        -   if $D \neq \Box$ then
        -       $DS = D \rightarrow DS$
$BS$:   upon event receiving from all nodes replies or $time\_out$ is reached do
        -   set $F = \Box$
        -   while $DS$ is not empty do
        -       if $F.Ts < \mathbf{H}(DS).Ts$ then
        -           $F = \mathbf{H}(DS)$
        -       $DS = \mathbf{T}(DS)$
        -   encrypted point-to-point $F$ to $LG$

$\tilde{a}$:   upon event receiving $REQUEST\_DATA(type\_spec)$ from $BS$ do
        -   if data $D$ matching type $type\_spec$ is stored then
        -       encrypted point-to-point $D$ to $BS$
        -   else
        -       encrypted point-to-point $\Box$ to $BS$
```

Algorithm 4.6: Simple Evasive Data Storage Algorithm - Flooding Data Retrieval

explicitly in pseudo code are two modifications to the Redundancy Chain algorithm: the first one supposes that an additional variable $DSRefresh$ is carried by each link of the chain. This variable is assumed to be initialized to f at the beginning of evasion process and decremented at each evasion step. The second is that each node has knowledge of the node from its chain that the base station knows about, this node if denoted by the $OldRep$ variable (in case of several nodes, a simple extension to a list of nodes is necessary). Both variables are referred to in the algorithm 4.7 that depicts the Base Station based Location Marking data retrieval notion.

The algorithm is similar to the flooding data retrieval algorithm, but avoids such a huge amount of messages being send by keeping the base station up to date regarding the location of the chains in the network. In order to do so, each chain maintains a $DSRefresh$ variable, and when this variable reaches zero the current head s of the chain sends an update message containing its identification, the identification of the old representative $OldRep$ and the stored data item's type specifier $type\_spec$ to the base station BS. The base station BS uses this information to maintain the $LOCS(type\_spec)$ list, where one list is available for each possible type. This list stores the representatives of chains that store data items of $type\_spec$ type. Thus when BS receives an update message $(newID, oldID, type)$, it updates the list $LOCS(type)$ by deleting $oldID$ and inserting $newID$. This ensures that for each type of data, the base station has a precise knowledge which nodes to query in order to obtain all data items of a specific type. This can be seen in case a legitimate user requests data of a certain type $type\_spec$: the base station finds the appropriate list $LOCS(type\_spec)$ and queries all nodes mentioned in this list, thus keeping the number of messages send drastically below the amount needed when flooding the network. The other parts, regarding replies to data requests and extracting freshest data, resemble the code of the flooding algorithm 4.6.

Participants:	
BS	- base station issuing the query
LG	- legitimate user querying for data
s	- node that participates in a redundancy chain

Parameters:	
f	- frequency, how many evasion steps to wait until to do an update
$type\_spec$	- the type specifier that a legitimate user queries for
$time\_out$	- time to wait for responses of nodes

```
$BS$:   upon event initialization do
        -      initialize $LOCS(type\_spec)$ for each possible $type\_spec$
$BS$:   upon event receiving $(newID, oldID, type)$ from $s$ do
        -      remove $oldID$ from $LOCS(type)$
        -      set $LOCS(type) = newID \rightarrow LOCS(type)$
$BS$:   upon event request for data of $type\_spec$ from LG do
        -      initialize $DS$ for storage of data of type $type\_spec$
        -      foreach node $a$ found in $LOCS(type\_spec)$ do
        -          point-to-point $REQUEST\_DATA(type\_spec)$ to $a$
        -      start timer $time\_out$
$BS$:   upon event receiving $D$ matching $type\_spec$ do
        -      if $D \neq \Box$ then
        -          $DS = D \rightarrow DS$
$BS$:   upon event receiving from all contacted nodes replies or $time\_out$ is reached do
        -      extract freshest data item $R$ from $DS$
        -      encrypted point-to-point $R$ to $LG$
$s$:    upon event $DSRefresh = 0$ do
        -      encrypted point-to-point $(ID(s), ID(OldRep), D.Ts)$ to $BS$
        -      set $DSRefresh = f$
$s$:    upon event receiving $REQUEST\_DATA(type\_spec)$ from $BS$ do
        -      if data $D$ matching type $type\_spec$ is stored then
        -          encrypted point-to-point $D$
        -      else
        -          encrypted point-to-point $\Box$
```

Algorithm 4.7: Simple Evasive Data Storage Algorithm - Base Station based Location Marking

The last elaborate data retrieval notion that was explained above is that of *Local Mark based Location Marking*, which assumes the presence of special nodes, called local marks, that are setup at deployment of the network. This notion is not provided in pseudo code for the Simple Evasive Data Storage algorithm, however the next section on Location Bound Evasive Data Storage will present a mechanism that is very similar to that of local marks.

In order to complete this section another issue needs to be addressed besides the presented data retrieval mechanisms. Introduced in the chapter on Data Security and Sensor Networks, a data item consists of a *generating node identifier*, a *timestamp*, a *type identifier* and *raw data*. The algorithms above and the specification of data storage referred to the freshness of a data item, this freshness addresses the tendency of sensor networks to change data of a specific type when new sensor readings are made, thus a user that queries for a specific item does not want to obtain data that is old and does not reflect the current status of the environment. Hence, the Evasive Data Storage as any data storage approach needs to provide facilities to ensure that data of the same type specifier is kept up to date, meaning that data items can be updated or *aggregated*. The problem intrinsic to the approach is that data wanders around the network and it is not quite clear where a data item with a certain type specifier is located (this is obviously very similar to the data retrieval problem). So merging data with new information of the same type specifier is a non trivial problem. In order to tackle this issue two directions can be pursued

- *Immediate Update*: Here the data item with the same type specifier has to be found in the network (most easily, but also most inefficiently by flooding) and is then updated with the new information immediately.
- *Deferred Update*: This update notion allows the data is be stored at an arbitrary location in the network not caring about the whereabouts of other data items with the same type specifier. Such a procedure does not induce a search for matching types of data, therefore saving lots of communication otherwise necessary. On the other hand, the network will have to store lots of redundant data items (of the same type specifier) where most of them are outdated and will have no use whatsoever. The merging of those data items of the same type specifiers is performed when legitimate user queries them. This is done either by the users themselves or in the base stations that accumulate all matching data before providing it to the user.

The explanations above reveal that the *Deferred Update* was assumed in the data retrieval algorithms above, but all of them would also work when the network used a form of *Immediate Update*. Fortunately, the similarity of the update and retrieval problems can allow Immediate Update to be performed the same way that data retrieval is done, with slight modification. Hence, this thesis will not go into more detail on updating data and keeping redundant data items in the network. Especially for the security aspects that are of interest the technicality of data update and the obvious possibility of adapting the algorithms for data retrieval for update purposes makes update problem be of less concern.

At this point all parts of Simple Evasive Data Storage and the respective data retrieval and update issues have been discussed. Thus the following sections that are still concerned with Simple Evasive Data Storage will investigate the correctness, complexity and security issues of this notion.

4.2.2 Correctness of Simple Evasive Data Storage

In order to discuss the correctness of the Simple Evasive Data Storage algorithm, each part of it that was presented in preceding sections will be investigated separately, and towards the end the satisfaction of the Data Storage properties, namely **Availability**, **Correctness** and **Freshness**, are addressed. The correctness will be argued in case of no node failures first, with several comments on the behavior when node failures are assumed to occur. The addition of malicious node failures will be explored in the last section that deals particularly with the security aspects of Simple Evasive Data Storage. Similarly, the satisfaction of the Evasiveness Security Property that Simple Evasive Data Storage should satisfy is discussed more detailed in the Simulation Chapter.

The first part of Simple Evasive Data Storage is the Displacement Trigger, whose purpose is to trigger the actual evasion process. Its correct behavior should be clear assuming that the local clock of the node does not fail as well as that the used random number generator is correct. The probability p_e and the time interval *time_int* guarantee that evasion will eventually be triggered, when appropriate values are assumed ($p_e \neq 0$ and *time_int* $\neq \infty$). Furthermore, the trigger only stops the timer for *time_int*, in case the New Storage Node Choice part has raised the *successful evasion* event, otherwise the evasion attempts will be repeated until data is displaced. As Displacement Trigger is not concerned with other nodes than the node it is running on, it does not need to deal with failures. Other parts of the Simple Evasive Data Storage need to be concerned in case the node that runs the Displacement Trigger algorithm fails.

The core of Simple Evasive Data Storage is the New Storage Node Choice Algorithm (algorithm 4.2), that chooses a new node for a data item D depending heavily on the parameter *Choice*. To prove the correctness of New Storage Node Choice algorithm it is necessary to argue that evasion of data will eventually take place with high probability. Suppose that New Storage Node Choice is invoked, then the node s running the algorithm will broadcast *REQUEST_EVADE* to all $v \in V(s)$ using the **local broadcast** primitive. Thus according to the specification each v receives the request, after time delay of Δt_{lb} at the latest, with p_{lb}, yielding that about

$p_{lb} \cdot |V(s)|$ many nodes receive the request after a finite time. Analogously, the number of nodes that successfully reply *CAN_STORE/CANNOT_STORE* to s after a time of $2 \cdot \Delta t_{lb}$ will amount to $r = p_{lb}^2 \cdot |V(s)|$. Due to the *No creation* and *No duplication* properties of the **local broadcast** primitive, all requests and replies will only occur once, hence, no vicinity node v receives the request twice or replies several times instead of just once. Similarly, the *Agreement* property ensures all nodes that receive a message from s in the respective time interval to receive the same request message. It is clear that those three properties will provide the same safety for communications for the coming steps as well, therefore they will not be mentioned explicitely and more emphasis will be put on the time delay and message omission probability. Setting the used timeout $time\_out \geq 2 \cdot \Delta t_{lb}$ will guarantee that all r nodes that successfully replied will be included in the following steps of the algorithm. Assuming that all nodes that replied are capable of storing the data item D that needs to be displaced, the set C will be of magnitude r and the *Choice* will return new node ID, given $r \neq 0$. Hence the question regarding the probability that $r = 0$ occurs, needs to be investigated: this happens when all request-reply communications between s and each v fail. The probability for successful request-reply is given by p_{lb}^2, as both local broadcasts need to be successful. To model this communication process, a random variable X which counts the number of successful request-replies between s and its vicinity $V(s)$ is introduced. X thus is the number of successes in $m := |V(s)|$ many trials. Hence, the distribution that applies is the binomial distribution, and the probability that k many successful request-replies occur is given by:

$$P(X = k) = \binom{m}{k} \cdot (p_{lb}^2)^k \cdot (1 - p_{lb}^2)^{m-k}.$$

The probability that not a single request-reply comes through and thus $r = 0$ occurs is

$$P(X = 0) = \binom{m}{0} \cdot (p_{lb}^2)^0 \cdot (1 - p_{lb}^2)^{m-0} = 1 \cdot 1 \cdot (1 - p_{lb}^2)^m = (1 - p_{lb}^2)^m$$

yielding the following probability that at least a single request-reply comes through:

$$1 - P(X = 0) = 1 - (1 - p_{lb}^2)^m.$$

Those probabilities thus describe the success of the classification of the vicinity $V(s)$ into the sets C and M in algorithm 4.2. Although the vicinity of s can be assumed to be quite small compared to the overall size of the network, it is of sufficient magnitude to make the probability that case $r = 0$ occurs particularly close to zero, thus, guaranteeing such that the set C passed to the *Choice* is not empty. Hence the sets C and M allow s to decide a new storage node ID, to which s sends D to. Receiving the D, the new node ID sends an acknowledgment back, hence the data transmission step succeeds with probability p_{lb}^2 after $2 \cdot \Delta t_{lb}$. Summarizing, the complete evasion step succeeds after a maximal total time[4] of $2 \cdot \Delta t_{lb} + 2 \cdot \Delta t_{lb} = 4 \cdot \Delta t_{lb}$ with probability

$$p_n = \left(1 - (1 - p_{lb}^2)^m\right) \cdot p_{lb}^2.$$

Furthermore, modeling with random variable X_n the number of repetitions needed until the New Storage Node Choice algorithm successfully evades D, it is obvious that X_n is geometrically distributed and has expected value of $E(X_n) = \frac{1}{p_n}$. Hence the Displacement Trigger will need to trigger the New Storage Node Choice algorithm on average $E(X_n)$ many times. In the same sense, the expected time until the New Storage Node Choice algorithm chooses a new hot node and evades data to it will occur after approximate delay of

$$E(X_n) \cdot 4 \cdot \Delta t_{lb} = \frac{4 \cdot \Delta t_{lb}}{p_n}$$

To further increase the probability of evasion and decrease the number of repetitions the probability p_{lb} of the local broadcast primitive can be increased as described in the Data Security and Sensor Networks Chapter. This

[4]This, of course, assumes that the computational time, medium access delay and so forth all sum up to zero and occur instantaneously.

will not only increase the communication overhead, but also the time delay Δt_{lb} for a successful transmission with the anticipated probability. Hence, the finite termination of the algorithm is guaranteed by the assumed finite transmission delay and the employed timeouts that address the possibility of both failures of vicinity nodes and loss of messages by the communication primitive. In case the node running New Storage Node Choice and Displacement Trigger respectively fails the correctness of the Redundancy Chain mechanism provides the anticipated continuation of data movement.

Thence the correctness Redundancy Chain needs to be investigated, which features the algorithms 4.4 and 4.5 in the subsequent discussion. To prove the correctness it is necessary to prove that the chain maintains its links and thus its predefined length of l with high probability. As seen in the correctness argument of algorithm 4.2, the probability of successful advancement is given by p_n, hence as redundancy chain makes usage of algorithm 4.2 the same probability applies.[5] This is also true for the time delay encountered in the Redundancy Chain, which is given by $4 \cdot \Delta t_{lb}$ as proved above. Similarly, the expected time until evasion is also inherited from the New Storage Node Choice algorithm. That said, the time delays the behavior after the New Storage Node Choice algorithm step succeeded needs to be investigated. In case evasion is successful, the chain needs to maintain its length l, thus the old head s of the chain sends a $NewHead(a)$ to all nodes mentioned in the old list $Chain$. The first probability that needs to be investigated is the arrival of the updated $NewHead(a)$ to the node a that has dropped out of the chain, due to the addition of the new head. If the message does not arrive, the node will not be unlinked properly from the chain, therefore arrival of the message can be considered crucial. The transmission will succeed with probability p_{pp} or more precisely with probability p_{lb}^{l}, which can attain a low value, if no retransmission is employed. Similarly, the maximal time delay encountered is given by Δt_{pp} or more precisely $l \cdot \Delta t_{lb}$, but does not play a major issue for neither security nor proper functioning. Back to the probabilities, even in case a does not receive $NewHead(a)$ and quit participation in the redundancy chain, it will continue to send HM messages to the $l-1$ many other links of the redundancy chain it knows of and eventually receive a $DROP\_OUT$ message causing it to quit the chain even in case of missing the original message from the head. Hence, what is left is to evaluate the probability that not a single $DROP\_OUT$ message arrives: in order to evaluate the probability that no drop out message arrives the same approach needs to be made as in the case of the New Storage Node Choice evaluation. Let X be a random variable which counts the number of successful HM requests and HM' replies between a and the $l-1$ many other nodes know to a from its local $Chain$ list. X thus stands for the number of successes in $l-1$ many trials. The variable satisfies the binomial distribution, and the probability that k many successful heartbeats occur is given by:

$$P(X = k) = \binom{l-1}{k} \cdot (p_{pp}^2)^k \cdot (1 - p_{pp}^2)^{(l-1)-k}.$$

The probability that not a single heartbeat succeeds amounts to

$$P(X = 0) = \binom{l-1}{0} \cdot (p_{lb}^2)^0 \cdot (1 - p_{lb}^2)^{(l-1)-0} = 1 \cdot 1 \cdot (1 - p_{lb}^2)^{l-1} = (1 - p_{lb}^2)^{l-1}.$$

Putting the mentioned probabilities together the probability that neither the initial $NewHead(a)$ nor a single $DROP\_OUT$ message arrives, gives the following probability:

$$p_s = (1 - p_{pp}) \cdot (1 - p_{pp}^2)^{l-1}.$$

This probability is denoted as split off probability p_s, which quantifies the occurrence of node a considering all other chain links to have failed. Under those circumstances a will, in order to provide subsequent functioning of the chain, split off and create a new chain, leading to another redundant chain for the same data item in the network. This is actually a protection mechanism, but can happen unintentionally and undesirably with the probability p_s. However, even for very unreliable channels $p_{pp} = 0.5$ and a chain length $l = 10$, such an unwanted

[5] The restriction of the New Storage Node Choice Algorithm to nodes not contained in the redundancy chain does not affect the p_n significantly, as a small $Chain \cap V(s)$ can be assumed.

split happens only with probability $p_s = 0.028$. For realistic channels with $p_{pp} = 0.8$ the split probability is even lesser: $p_s = 2.92 \cdot 10^{-5}$. Hence, the redundancy chain propagates through the network maintaining its length with sufficiently high probability $(1 - p_s)$. In order to complete the picture of the process of selecting a new head and getting rid of the oldest chain link a, the time delays need to be looked at. As argued above a might not receive the new head message and therefore assume that it is still part of the redundancy chain. The heartbeat messages, however, provide, along with the *DROP_OUT* messages, the appropriate mechanism to address this issue. In order to behave appropriately the used timeout *time_out* has to be set to a value greater than Δt_{pp} or $l \cdot \Delta t_{lb}$, otherwise the node might jump to false conclusions about the liveness of the other links. As the algorithm forces the node to work its way through the chain up to the head, the complete time delay until a finally receives a *DROP_OUT* or splits off the chain is bounded by $l \cdot \Delta t_{pp}$ and $l^2 \cdot \Delta t_{lb}$ respectively. Summarizing, the crucial point is to set the *time_out* value such that nodes are not unnecessarily assumed to have failed and therefore cause unneeded messages to be sent. It is also shown that a split off does not only happen with a low probability, but also after a comparably long period of time. Even in case of unfortunate message loss leading to a split of the chain, data is not lost, but redundancy is increased. The investigations regarding the heartbeat messages and the possible occurrence of a split off do not only apply to the oldest node in the chain, but also to all links. Letting $\bar{a}$ stand for such a node, excluding the head, and letting q stand for its position in the chain, i.e. $1 < q <= l$, the split off probability $p_s(q)$ depends on q and, derived analogously as described in the special case above, amounts to:

$$p_s(q) = (1 - p_{pp}^2)^{l-(l-q+1)} = (1 - p_{pp}^2)^{q-1}.$$

This obviously shows that the diminishing low split $p_s = p_s(l)$ probability derived for the case of the oldest node that needs to be unlinked from the chain, reaches different and dangerous dimensions in case of nodes that are closer to the head node. To be more concrete, assume the same values as above: let $p_{pp} = 0.5$ and $l = 10$. Then for the node just preceding the current head, i.e. $q = 2$, the probability for a split off now amounts to $p_s(2) = (1 - (0.5)^2)^{2-1} = 1 - 0.25 = 0.75$. Even the next node, $q = 3$, still yields a probability of $p_s(2) = (1 - (0.5)^2)^{3-1} = (1 - 0.25)^2 = 0.562$, which is also a very high value for such an unwanted event. Obviously, this forces the **local broadcast** and **point-to-point** primitives to use retransmissions or some form of acknowledgments to increase the fundamental p_{lb} and p_{pp} probabilities. Fortunately, it can be left as a side note for the implementer of the Redundancy Chain algorithm to use retransmissions or can take form of a simple modification of the redundancy chain algorithm, which causes nodes at position q in the chain to repeat the sending of heartbeat messages depending on their position, for instance $l - q$ many times. Such a simple modification will cause the nodes closer to the head, i.e. smaller q, to retransmit heartbeat message, and therefore make splitting particularly improbable. Summarizing, the analysis above showed that the mechanism has to address issues that can arise due to the unreliability of channels, and either solve them with brute force, meaning a flat retransmission convention to increase the overall communication reliability, or in form of small enhancements in the mechanism itself, which of course yields better message efficiency. In the end, the Redundancy Chain algorithm does wander successfully through the network with both acceptable time delays and sufficiently high probabilities. Another matter is how the redundancy chain copes with real failures of links: assume that a link a' in the chain fails, then the node a preceding a' in the chain, i.e. the current chain is assumed to have the form

$$Chain = (\diamond, \ldots, \diamond, ID(a''), ID(a'), ID(a), \diamond, \ldots, \diamond),$$

will not receive a reply to its HM it sent to a' after time delay Δt_{lb} and thus run into the timeout $time\_out \geq \Delta t_{lb}$. This causes a to send a HM to the next node in $Chain$, i.e. node a'', which will reply after a maximum delay of Δt_{pp} as it is assumed to not have failed. Hence, the message delegates heartbeat messages to nodes higher in the chain, allowing the complete chain to slowly proceed without taking any particular measures and eventually reach its full length of l again by leaving the failed node behind. A special case is the failure of the head of a redundancy chain: the node s' preceding the head in the $Chain$ will run into a timeout and as there are not more nodes above the head in $Chain$, s' will become new head by informing all nodes of the chain and starting its own Displacement Trigger. Furthermore, the redundancy chain does not allow for data to vanish, as nodes only delete data, when successfully instructed to do by the head of the chain, which happens only when l other nodes exist that are storing the data item as well, guaranteeing the **persistency** of data, in case the amount of failures is lower than l, the number of links in the redundancy chain. To finish the discussion of the Redundancy Chain algorithm, it should be noted that only few references where made to the timing delays involved in the algorithm. This is due to the fact that, except for the timeouts used, the timing of the redundancy chain does not play a major role. There is no need for exact evaluation of delays involved with heartbeat messages or split offs, as long as the functioning is guaranteed with deliberately chosen timeouts.

Now that the main parts of the Simple Evasive Data Storage algorithm have been analyzed to perform correctly, the question remains whether the proposed algorithm along with the two data retrieval algorithms does satisfy the Data Storage properties. Hence, the rest of the section investigates the question, if the Simple Evasive Data Storage Algorithm in cases of both the Flooding Data Retrieval and the Base Station based Location Marking Data Retrieval approaches, manages to satisfy **Availability**, **Correctness** and **Freshness**. The first one evaluated is the Simple Evasive Data Storage with Flooding Data Retrieval case:

Availability: To prove that this property is satisfied by the Simple Evasive Data Storage algorithm and the Flooding Data Retrieval Data Retrieval mechanism two basic issues need to be addressed: the time delay it takes for the data item to be retrieved and the probability involved. The first one evaluated is the time delay incurred when a legitimate user requests data. This is done by assuming perfect circumstances, i.e. disregarding the unreliable channels, which are then included in the probability evaluation later on. Assume that a legitimate user LG queries a base station BS for a data item of type $type\_spec$ at time t. In the network m redundancy chains of length l are assumed to exist and to store data items $D_1, \ldots, D_m$ such that $D_1.Type = \ldots = D_m.Type = type\_spec$ holds true. Furthermore, all the nodes participating in the chains are assumed to work correctly. At this stage, the Flooding Data Retrieval algorithm initiates a **global broadcast** consisting of a request for the data type queried by LG, which due to its properties will reach all nodes after Δt_{gb} at the latest. Thus at time $t + \Delta t_{gb}$ all nodes of the redundancy chains will have received the $REQUEST\_DATA(type\_spec)$. As all are assumed to work correctly and unreliability of channels is disregarded, all $m \cdot l$ hot chain nodes will reply with at time $t + \Delta t_{gb}$ either with matching data items or $\Box$. Thus the base station BS will have received all the replies before or at time $t + \Delta t_{gb} + \Delta t_{pp}$. If the transmission of the found freshest data item F from BS to LG is included into the picture the overall delay after which data is provided amounts to:

$$\Delta t_{da} \leq \Delta t_{gb} + 2 \cdot \Delta t_{pp}.$$

Now that the time delay involved in the availability property the neglected property of unreliable channels has to be taken into account. Although, the argumentation above ignored unreliable channels the time delay will remain valid as long as at least some of the nodes respond. Thus what needs to be evaluated is the probability that at least one data item is successfully provided to the legitimate user: again assume a legitimate user, LG, queries a base station BS for data of type $type\_spec$. Furthermore, assume that m redundancy chains store data items $D_1, \ldots, D_m$ with $D_1.Type = \ldots = D_m.Type = type\_spec$. The Flooding Data Retrieval will then send a $REQUEST\_DATA(type\_spec)$ by using the **global broadcast** primitive. In case a node participating in any of the chains receives the request it will respond with its data item. The base station accumulates all received data items and provides one of them to the LG or in case a timeout is reached and no data items are received at all, BS sends $\Box$ to LG. Now the probability that at least one data item D_i is provided needs to be investigated: the probability that the request does reach a link of the chains is p_{gb}, similarly the probability that a link does respond to BS is p_{pp}, yielding for the complete message exchange a failure probability of $(1 - p_{gb} \cdot p_{pp})$ for each node. Again a binomial distributed random variable X is defined that counts the number of successful responses to the BS. The term defining the probability of k successful trials is:

$$P(X = k) = \binom{m \cdot l}{k} \cdot (p_{gb} \cdot p_{pp})^k \cdot (1 - p_{gb} \cdot p_{pp})^{m \cdot l - k}.$$

Thus the probability for successful arrival of at least one data item of the requested type is:

$$p_{da} = 1 - (1 - p_{gb} \cdot p_{pp})^{m \cdot l}.$$

Even in unfavorable circumstances, taking $p_{gb} = 0.3$, $p_{pp} = 0.5$, for five available chains $m = 5$ each with length $l = 5$, the probability for the BS to receive at least one data item is $p_{da} = 0.982$. Even in case of f failures as long as $f < m \cdot l$ sufficiently many nodes will respond (however such a bound does not guarantee freshness, as argued below). Furthermore, if the redundancy mechanism that was used is not Redundancy Chain, but Redundancy Copies the same argumentation applies by setting the the parameter l equal to 1. To finish the discussion of the probability involved in the Availability property, it should be shortly stated how to increase the availability probability p_{da}. The most fundamental is to increase both the p_{gb} and p_{pp}, by increasing the repetitions at the local broadcast level; the dependency between those primitives has been shown in the Data Security and Sensor Networks Chapter. Besides this elementary approach, increasing the l influences the availability positively. To sum up, both the Δt_{da} and the p_{da} have been investigated and found to be satisfied by the Simple Evasive Data Storage and the Flooding Data Retrieval algorithms. The last note that needs to be made is addressing the $time\_out$ used in the Flooding Data Retrieval algorithm. In order to not miss any responses and maintain the bound given above the value of $time\_out$ has to satisfy $time\_out \approx \Delta t_{gb} + \Delta t_{pp}$, which is assumed to be calculable given knowledge about the communication primitives.

Correctness: The data provided is correct, assumed that there are no malicious nodes in the network and that nodes do not unintentionally provide wrong $Type$ fields in data items, which they are supposed to check before sending any data out to the base station. Otherwise a simple voting mechanism can be used to ensure Correctness in case malformed data exists in the network.

Freshness: As with the Availability property there are two issues involved: timing and probability. Assuming the configuration of redundancy chains as stated above and that the legitimate user queries for a specific data item of type $type\_spec$ at time t. Then, at that point in time, one of the stored data item is the freshest and let Ts_f be the according timestamp. Ignoring the unreliable communication primitive, the base station BS receives data items $D_1, \ldots, D_m$ from all chains in the network that stored data items matching the $type\_spec$ at time $\Delta t_{gb} + \Delta t_{pp}$, among which $D_i.Ts = Ts_f$ for some i. As such a data item was assumed to be stored and ready for query at time t the legitimate user contacted the base station. However, during the period from t to $t + \Delta t_{gb} + \Delta t_{pp}$, one or several updates might have occurred and stored for query. This would then, of course, cause at least one of the received data items D_j for some $j \neq i$ to satisfy $D_j.Ts > Ts_f$. Under any of the circumstances, the base station will return the data item with maximal timestamp, hence a data item $D \in \{D_1, \ldots, D_m\}$ with $D.Ts \geq Ts_f$, which is what the Freshness property needs the algorithm to satisfy. Naturally, this is only true as it can be assumed that the timestamps $D.Ts$ of the all received data items D are correct (based on the assumption that nodes generating data do not intentionally store false information and on correct local clocks). Summarizing, the retrieval algorithm will provide the freshest data item according to the Freshness Property, when the unreliability of the communication primitives are left out of the picture. Of course, due to our system model, this cannot be done, and therefore the according probability for successfully delivering the freshest data item has to be analyzed. The problem of wrong freshness can arise when not all redundancy chains that store matching data items reply to the $REQUEST\_DATA(type\_spec)$ request sent by BS. In such case the base station BS might provide a data item D' to the legitimate user LG even though a redundancy chain existed at time t of the query that stored D with $D'.Ts < D.Ts$. Thus it is necessary to evaluate the probability p_{fr} that data provided is actually the freshest data in the network. This is different from the probability assessed in the Availability property, because there only a single node from any of the chains had to answer, whereas to guarantee freshness at least a single node from each of the chains needs to answer. Hence what must be evaluated is the probability that at least one node from each redundancy chain responds to the base station successfully. Suppose there are m redundancy chains in the network storing data matching $type\_spec$. For each chain the random variable X_i is introduced and assumed to count the successful answers the base station receives from chain i:

$$P(X_i = k) = \binom{l}{k} \cdot (p_{gb} \cdot p_{pp})^k \cdot (1 - p_{gb} \cdot p_{pp})^{l-k}.$$

Hence yielding for the probability that at least a single node responds from chain i the probability:

$$1 - P(X_i = 0) = 1 - (1 - p_{gb} \cdot p_{pp})^l.$$

Now to answer the question of all chains having at least a single node transmitting data to BS, a random variable X is introduced that counts how many chains in the network respond, in the sense that at least one of its links responds:

$$P(X=k) = \binom{m}{k} \cdot \left(1 - P(X_i = 0)\right)^k \cdot \Big(1 - \left(1 - P(X_i = 0)\right)\Big)^{m-k}.$$

The case that is of interest is $k = m$, i.e. all chains have at least a node that responds, provides the wanted probability:

$$\begin{aligned} P(X=m) &= \binom{m}{m} \cdot \left(1 - P(X_i = 0)\right)^m \cdot \Big(1 - \left(1 - P(X_i = 0)\right)\Big)^{m-m} \\ &= 1 \cdot \left(1 - P(X_i = 0)\right)^m \cdot \Big(1 - \left(1 - P(X_i = 0)\right)\Big)^0 \\ &= \left(1 - (1 - p_{gb} \cdot p_{pp})^l\right)^m =: p_{fr} \end{aligned}$$

In the case of $p_{gb} = 0.6$, $p_{pp} = 0.8$, $l = 10$ and $m = 20$, the probability for the base station providing the freshest data item is $p_{fr} = 0.971$. Here this probability exhibits a very sensitive behavior to the l parameter. Fixing the other parameters as given and shortening l to 8, reduces the freshness probability down to $p_{fr} = 0.898$, a shortening to $l = 6$ even results in $p_{fr} = 0.670$, which surely is not an acceptable value. Hence in order to keep p_{fr} high, the l has to be as high as possible and, as with the Availability property an increase in the repetitions at the local broadcast level can yield higher probabilities of p_{gb} and p_{pp}, which also positively influence the freshness probability p_{fr}. Concluding, the Simple Evasive Data Storage and the Flooding Data Retrieval satisfy the Freshness Property, by providing a data item D with $D.Ts \geq Ts_f$ with a probability p_{fr}.

Finished with the correctness for Flooding Data Retrieval, the investigation of the Base Station based Location Marking retrieval is left. However, as the probabilities that result for availability and freshness are the same, the argumentation that to retrieval itself, regarding the update location and so forth, is omitted but can be easily found to be correct. This section showed that the Simple Evasive Data Storage Algorithm behaves correct and satisfies the Data Storage properties with either the Flooding Data Retrieval or the Base Station based Location Marking algorithms. Hence the next section deals with the message complexity of Simple Evasive Data Storage, leaving the security analysis for the section afterwards.

4.2.3 Complexity of Simple Evasive Data Storage

It is important to take a look at the complexity of the presented algorithm for Simple Evasive Data Storage, as sensor networks are sensible to the available resources. The complexity analysis conducted here will be from a qualitative point of view addressing all presented parts of the Simple Evasive Data Storage Algorithm as well as the two major costs in data storage, the event costs and the query costs (which already have been introduced in the scope of conventional storage methods). The metric that describes the complexity of the mechanism will be the amount of messages needed for a particular algorithm part to achieve its objective.

The first algorithmic part of Simple Evasive Data Storage is the Displacement Trigger, it does not incur any message costs at all, because it just exists to trigger the actual evasion of data. Thus the important part is the amount of messages needed by algorithm 4.2. The algorithm starts off with a single message of s, inducing all nodes in the $V(s)$ to respond. Setting $r := |V(s)|$, the resulting number of messages is $r+1$ for s to obtain the sets C and M. With that information the new node ID is chosen and the data item D is sent to ID, inducing ID to send an acknowledgment. This yields 2 additional messages, for a total of $r+3$.[6] If the New Storage Node Choice Algorithm is executed with the Camouflage Enhancement, algorithm 4.3, the acknowledgment does not consist of a single message, but of r many that are send, implying as total number of messages $2 \cdot r + 2$. To be more precise, the total number of message is influenced by the unreliability of the local broadcast primitive,

[6]Note that the size of the messages is being ignored.

which is successful only with probability p_{lb}. Thus the actual number of messages is $2 \cdot p_{lb} \cdot r + 2$. In any case the number of messages, depends only on the number of nodes in the vicinity, yielding for the New Storage Node Choice Algorithm a complexity of $\mathcal{O}(r)$. It can be argued that for efficiency the first round of the New Storage Node Choice Algorithm, i.e. the classification of the vicinity $V(s)$ into sets C and M can be dropped from the algorithm, because of the available acknowledgment and, in case of failure, the repetition of the mechanism. This would yield less messages, however due to the Camouflage Enhancement the overall order of magnitude would still remain $\mathcal{O}(r)$.

The next important part that needs to be investigated is the Redundancy Chain Algorithm. Which adds at each evasive step, messages to maintain the chain. When New Storage Node Choice chooses a new head, the old head sends to each node in the chain *Chain* a message, thus each adding additional $l - 1$ many messages. However, in order to maintain connectivity of the chain, heartbeat messages are sent every *time_int* interval. Those heartbeat messages (assuming no failure occurs) add another $l - 1$ many messages, as each node checks the next higher link in the chain (except for the head of the chain of course). If the amount of heartbeat messages is assumed to be constant between each evasion step, the total number of messages for Redundancy Chain including the New Storage Node Choice message overhead becomes $\mathcal{O}(r + l)$.

This finishes the message complexity for the Simple Evasive Data Storage Algorithm parts and leaves the investigation of the event cost and the query costs. The event costs for the Evasive Data Storage are no longer fixed but increase over time, this is an intrinsic property of Evasive Data Storage notion because it intentionally moves data in the network and thus causes messages to be exchanged. However, the message overhead incurred by evasiveness needs not to be exorbitant large compared to the fixed amounts of conventional storage methods. Looking at the life time of a sensor network, the choice of the essential parameters *time_int* and p_e for Simple Evasive Data Storage or to be more precise the Displacement Trigger, can be chosen to cause evasiveness infrequently, thus keeping the overhead compared to the conventional storage methods low, when looking at the complete life time of the network. Of course, those additional messages also incur better security properties, which cannot be obtained without additional work. Recapitulatory: the event costs for storing data are simply $\mathcal{O}(r + l)$ for each evasive step that is taken by the Simple Evasive Data Storage Algorithm.

Coming to the query costs, i.e. the number of messages that need to be exchanged in order to retrieve data that a legitimate user requested, the two approaches Flooding Data Retrieval and Base Station based Location Marking need to be considered. The Flooding Data Retrieval incurs costs of n many messages for flooding the complete network in order to find redundancy chains or redundant copies of data items, furthermore each hot node that stores matching data will need on average $\sqrt{n}$ many message to reply to the request. As the number of hot nodes matching a data item, say $\bar{h}$, can be assumed to be negligibly small compared to n, i.e. $\bar{h} \ll n$, the term $\bar{h} \cdot \sqrt{n}$ can be tolerantly considered to be n and thus yielding query costs for Flooding Data Retrieval to be $\mathcal{O}(n)$. The other possibility for data retrieval yields a different distribution of query costs. The Base Station based Location Marking data retrieval needs to maintain the location of a data item up to date at a base station or even all base stations in the network. Thus every f evasion steps the Base Station based Location Marking algorithm sends an update message to the base station, yielding $\mathcal{O}(b \cdot \sqrt{n})$ many messages, which will be considered part of the query costs. The obvious advantage is that at the time a user queries for a data item the base station BS knows where the $\bar{h}$ many hot nodes matching the desired type are located and can contact them directly with costs of only $\mathcal{O}(\bar{h} \cdot \sqrt{n})$.

Costs:	**Flooding Data Retrieval**	**Base Station based Location Marking**
Query Costs:	$\mathcal{O}(n)$	$\mathcal{O}(h \cdot \sqrt{n})$
Event Costs:		
- Every Evasion Step:	$\mathcal{O}(r + l)$	$\mathcal{O}(r + l)$
- Every f Evasion Step:	–	$\mathcal{O}(b \cdot \sqrt{n})$

Table 4.2: Simple Evasive Data Storage Data Retrieval Costs Overview

However looking the those costs (see table 4.2), especially the query costs, the problem seems obvious that the number of messages needed is large, for the Flooding Data Retrieval the number reaches $\mathcal{O}(n)$, which is utmost undesirable. The Base Station based Location Marking on the other hand has small query costs, but causes base stations to be contacted very frequently adding a substantial costs of $\mathcal{O}(b \cdot \sqrt{n})$ for each evasion step of every data item in the network. Thus a more elaborate approach for the storage of data in sensor networks is needed that reduces these large message overheads. The section on the Location Bound Evasive Data Storage notion will address this problem in an elegant way.

4.2.4 Security of Simple Evasive Data Storage

The objective of this section is to analyze what the odds for an adversary are to pick a hot node when the Simple Evasive Data Storage storage algorithm is used in the respective network. This is hence what the Evasiveness Security Property states and needs secure data storage algorithms to fulfill. However the contents of this section is more of a preparation for the simulative evaluation of the Evasiveness Security Property in scope of the Simulation Chapter, however several stronger adversaries will also be addressed that do not find any mentioning in the thorough investigation of the Evasiveness Security Property in the Simulation Chapter.

The assumption made is that the network consists of n network nodes that are all equally likely to store data. The evasion process does not prefer particular nodes in the long run (except the *Choice* does so), and the events that generate data can be assumed to be equally likely for every location of the network. Assuming that an $\mathcal{A}_{bp}(blind, passive)$ adversary enters a sensor network and wants to obtain a hot node, he has no choice but to guess which nodes are hot. Hence what is his probability of being successful in this case? Enumerating each node as $\{a_1, \ldots, a_n\}$ yields a probability of $P(a_i) = 1/n$ for a node a_i to be hot. As mentioned, this is reasonable for Simple Evasive Data Storage as data can wander anywhere in the network and does not prefer any particular nodes. Furthermore, it has to be ignored that events might exhibit some kind of patters and therefore break the uniform distribution assumption, at least at the start of data wandering, because as time passes by, evasion should cause the data to be equally likely in any part of the network. Having stated the assumption that each node is equally likely to store data and taking into account the presumption that there are h hot nodes in the network, the probability for the event that the adversary can pick a hot node at random from the network amounts to

$$P(adversary\ picks\ a\ hot\ node) = \frac{h}{n}.$$

This probability can be made more precise using the hypergeometric distribution. Let X_a be a random variable describing the probability that $\mathcal{A}_{bp}$ successfully finds l hot nodes when picking k nodes from the network. The probability is given by the following term

$$P_k(X_a = l) = \frac{\binom{h}{l} \cdot \binom{n-h}{k-l}}{\binom{n}{k}}.$$

Whereas the restriction applies that the number of successful hot picks l needs to be an integer from the interval $[\max\{0, k+h-n\}, \ldots, \min\{k, h\}]$. In order to get the initial probability of $\frac{h}{n}$, the probability for $k = 1$ (one pick) and $l = 1$ (the pick is hot) needs to be calculated:

$$P_1(X_a = 1) = \frac{\binom{h}{1} \cdot \binom{n-h}{0}}{\binom{n}{1}} = \frac{\frac{h!}{1!(h-1)!} \cdot \frac{(n-h)!}{0!(n-h)!}}{\frac{n!}{1!(n-1)!}} = \frac{h \cdot 1}{n} = \frac{h}{n}$$

Which naturally yields the probability argued less formally above. The important fact is that an such an adversary $\mathcal{A}$ cannot do better that this probability, when $\mathcal{A} \preceq \mathcal{A}(blind, \star)$ holds. Note that in the argumentation above the value of h stands for hot nodes, which can be ambiguous in whether it refers to distinct data items that are stored in the network or actual hot nodes regardless of the data item stored. The above makes use of the latter definition. However, taking the former as being described by h the argumentation above needs to be re-assessed, as redundancy caused by redundancy chains has to be considered. This is done by substituting the $h \cdot l$ for each occurrence of h in the formula for $P_k(X_a = l)$ above.

Now another case that concerns adversaries $\mathcal{A} \preceq \mathcal{A}(blind, \star)$ is investigated: assume $\mathcal{A}$ knows the location of a hot node $\tilde{a}$. Now if $\mathcal{A}$ wants to reaccess data from $\tilde{a}$, he has a success probability of 1, when a conventional storage method is employed (Local Storage, for instance). Even in case that $\tilde{a}$ fails, the redundancy approach of conventional storage methods will have located redundant copies in the close vicinity of $\tilde{a}$, allowing for a very high success probability for $\mathcal{A}$. How successful is such an adversary in case Simple Evasive Data Storage is employed? To simplify the analysis the redundancy nodes are ignored and only the current head of a redundancy chain is regarded as hot. Assume that since $\mathcal{A}$ accessed $\tilde{a}$ the last time, e successful evasive steps where conducted. The following investigates the probability for $\mathcal{A}$ to pick a hot node in the region surrounding the former initial hot node he remembered:

To evaluate the success probability of an adversary the approximate number of nodes that can be reached after e steps needs to be approximated. Let on average for each node a the cardinality of its vicinity be $|V(a)| = r$. Assuming that the vicinity has a radial form the radius can be described as $\mathcal{O}(\sqrt{r})$, hence after e steps the region R that covers all nodes that could store displaced data, which is also assumed to be in radial form, has an approximate radius of $\mathcal{O}(e \cdot \sqrt{r})$ yielding as approximate number of nodes[7] that possibly store data $\mathcal{O}(e^2 \cdot r)$. Thus if adversary $\mathcal{A}$ can estimate the values e and r, he will need to consider randomly picking a node from the region R described above, yielding a success probability[8] of $\frac{1}{\mathcal{O}(e^2 \cdot r)}$, which is assuming that each node in this region has equal probability of becoming the new hot node. Unfortunately, the assumption of an uniform probability distribution among the possible $\mathcal{O}(e^2 \cdot r)$ nodes is too optimistic and thus not quite correct. However, a exact distribution for each of the choice functions given in table 4.1 cannot be given, thus the Simulations Chapter includes a detailed analysis based on simulations, as to how the distribution of hot nodes in R depends on the choice function and which one reveals to be the most effective one.

Similarly, the main advantage that the Simple Evasive Data Storage mechanism provides, the decreased success probability for an adversary that tries to access the node $\tilde{a}$, he initial remembered to be hot has to be addressed. Taking only a single evasive step, $e = 1$, yields a probability for $\mathcal{A}$ to find data at $\tilde{a}$ to be zero. This is simply because in one evasion step the data item $\tilde{a}$ stored will move to one of its vicinity nodes, thus makes it impossible for the data item to remain at $\tilde{a}$. However, after one evasion step $\mathcal{A}$ still has a probability of $\frac{1}{r}$ to pick (with a single try) the new hot node by accessing any node in the vicinity of $\tilde{a}$. The highly anticipated drop, from having a success probability of 1, with conventional storage, can hence be achieved with Simple Evasive Data Storage, however for a complete satisfaction of the Evasiveness Security Property more detailed evaluation is called for. As after the next evasive step the initial hot node $\tilde{a}$ can have a quite high probability of being chosen to store the data item again. This probability is highly dependant on which of the choice functions presented in table 4.1 is used and is quite complex to analyze. However, the Simulation Chapter will investigate this success probability as well and show that the advantage gained by the Evasive Data Storage notion is highly satisfactory compared to the conventional approaches.

It has been investigated what chances an adversary has to pick a hot node in the regions surrounding a former node he remembered to store data. The main assumption made was that the adversary could not predict the choice the *Choice* made. Hence, the next step is in analyze the success probability given that adversary $\mathcal{A}(blind, \star)$ has obtained information about *Choice* used allowing him to predict the node chosen at each step with probability $\bar{p}$. Hence after e evasion steps adversary $\mathcal{A}$ will have a probability of $\bar{p}^e$ to pinpoint successfully the new storage node. It is clear that even high predictability of the choice function by an adversary will decrease quickly with each evasion step, thus yielding even for a high precision of $\bar{p} = 0.80$ a less than fair success probability of 0.107 after 10 evasion steps. Overall a significant advantage in terms of security compared to the properties of conventional storage approaches can be observed.

At this point the focus is directed towards several simple and distinct issues regarding the security of Simple Evasive Data Storage in the presence of stronger adversaries. The first issue is the redundancy chain

[7]To be exact, this approximation should be $\min\{\mathcal{O}(e^2 \cdot r), n\}$.

[8]Here of course, the same correction has to be applied: $\max\{\frac{1}{\mathcal{O}(e^2 \cdot r)}, \frac{1}{n}\}$.

mechanism and an adversary matching the strong $\mathcal{A}(\star, active)$ model. Such an adversary might attempt to install a malicious node in the network and wait for a evasively stored data item to come by. Naturally, such an adversary does not necessarily need to find hot nodes but can install several malicious nodes and wait for data to come in. This security concern will be addressed in the next chapter containing advanced Evasive Data Storage techniques. Furthermore, instead of just trying to obtain data, $\mathcal{A}$ can try to diminish a complete redundancy chain. However to do so the adversary cannot just capture the head and send appropriate messages to all links, but needs to either capture at least l many nodes, or impersonate the same amount. This is because for a node to quit participation in a redundancy chain the head of the chain needs to change to a new one not found in the chain before. Thus to force all nodes up to the last non malicious node to quit participation, the head needs to be changed l many times. Furthermore, assuming that a $\mathcal{A}(\star, passive)$ can change the data of up to t many nodes in a chain, in order to guarantee correctness of the provided data, the l needs to satisfy the inequality $l > 2 \cdot t + 1$ such that a base station obtaining data can decide for the correct version by usage of a majority vote. Another issue is to investigate the presence of an extended adversary $(\kappa, \Delta) - \mathcal{A}(\star, \star)$ that can be present in the network. Setting $\kappa = 1$ and assuming that the time interval Δ that restricts the adversary $\mathcal{A}$ starts at the time t the adversary successfully identifies a hot node $\tilde{a}$, then Simple Evasive Data Storage can guarantee (with high probability) that $\mathcal{A}$ will not be successful at obtaining the data stored at $\tilde{a}$, when the following inequality is satisfied:

$$\Delta > E(X_n) \cdot 4 \cdot \Delta t_{lb} = \frac{4 \cdot \Delta t_{lb}}{p_n},$$

where $E(X_n)$ represents the expected number of repetitions until evasion is successful. Thus when Δ of $\mathcal{A}$ is restricted as just stated, the data stored at $\tilde{a}$ will have been moved before the adversary successfully accesses the node. Of course, this assumes that no redundancy chain is employed, as such a chain would leave the data at $\tilde{a}$ until l many evasion steps have occurred, where l stands for the length of the redundancy chain. To guarantee that an $(\kappa, \Delta) - \mathcal{A}(\star, \star)$ adversary is not successful in presence of redundancy chain the restriction on Δ has to be even more severe[9]: $\Delta > l \cdot E(X_n) \cdot 4 \cdot \Delta t_{lb}$. Hence, it can be seen that the time delays involved in the mechanisms and the presence of adversaries that also obey timing restrictions can be used to provide higher security. However, as stated now the Simple Evasive Data Storage does not allow to influence the involved time delays directly, but this will be addressed in the Time Constraint Evasive Data Storage Algorithm in the following sections.

Looking back, the Simple Evasive Data Storage Algorithm presented in this section has introduced a very interesting approach to address security issues from a data security perspective, but also several concerns, especially concerning the efficiency of data retrieval and data update respectively. This is caused by the quite liberal way in which data is stored or wanders through the network, making it hard to control and keep track of. In most cases, a simple flooding does the job of locating data in the network, but being too wasteful, other ways are called for. The alternative to flooding, the Base Station based Location Marking approach for instance, incurs rather long distance communication at each displacement step, being too wasteful as well although having better message complexity properties when a legitimate users query data. Thus the following section offers a refinement of this simple approach which is tailored specifically to address the issue of data retrieval and data update respectively.

4.3 Location Bound Evasive Data Storage

Having seen not only the advantages of Evasive Data Storage proved in the last section, but also the disadvantages of how data needs to be kept track off in the network, this section aims to present a refinement to the Simple Evasive Data Storage algorithm presented in the previous section. Or to be more precise provides a framework where the Simple Evasive Data Storage Algorithm is used, but in a restricted form, yielding better properties when performing data retrieval and update respectively, but not loosing the security advantages

[9]This also needs to assume that the evasive redundancy chain does not behave unfortunate and return to the hot node $\tilde{a}$ during the l many evasions.

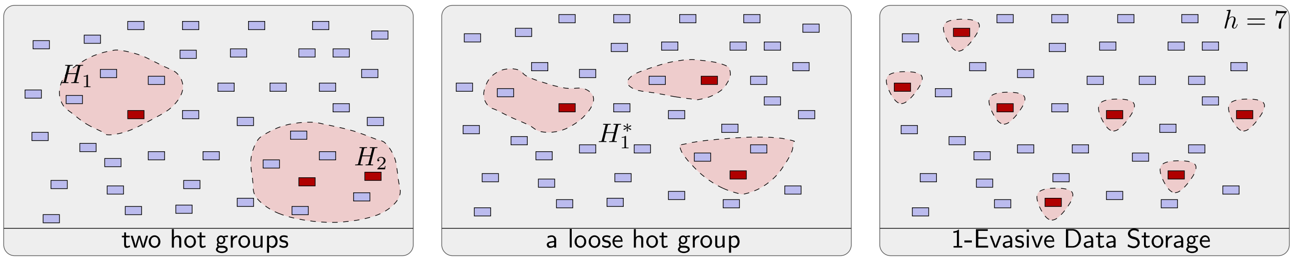


Figure 4.3: Location Bound Evasive Data Storage - Basic Concepts

inherent to Evasive Data Storage. The notion introduced will be referred to as Location Bound Evasive Data Storage or d-Evasive Data Storage. Unlike the conventional assumptions that the sensor network contains data stored in h distinct hot nodes (which remain hot over their life time), the d-Evasive Data Storage allows for h distinct hot regions that are responsible for storing the data gathered by the sensor network's nodes. Before completely diving into the notion of Location Bound Evasive Data Storage the concept of a hot region or *hot group*, as it will be referred to, needs to be introduced: A hot group H is a set consisting of d sensor nodes $\{a_1, \dots, a_d\} \subseteq \Pi$, such that for arbitrary pair $a'_1 \in H$ and $a_k \in H$ there is a way to exchange messages by using nodes from H only, i.e. a path $a'_1, \dots, a'_k$ with $a'_{i+1} \in V(a'_i)$ for all $i \in \{1, \dots, k-1\}$ and $a'_j \in H$ for all $j \in \{1, \dots, k\}$ holds. Thus, nodes in a group can move data around without having to rely on nodes that are not members of their respective group. A more lax notion is that of a *loose hot group* H^* where the restriction of group exclusive communication is dropped and hence data exchanged among members of such a group can also pass through arbitrary nodes[10] $a \in \Pi \setminus H^*$. Both concepts are depict in figure 4.3, two hot groups H_1 and H_2 are shown in the left picture and a single loose hot group H_1^* is shown in the middle picture. However, in this work the notion of a loose hot group will not find major attention and focus will center around the ordinary hot group notion.

Now that hot groups are a familiar term, d-Evasive Data Storage can be precisely defined. It forms an extension to the traditional notion of having h hot nodes in a sensor network by replacing single hot nodes with h distinct hot groups $H_1, \dots, H_h$, where each group has at most $d \geq 1$ members, i.e. $|H_j| \leq d$ for each $j \in \{1, \dots, h\}$ holds true, in order to store the gathered data. It is tried to keep the cardinality of each H_j to be as close to d as possible at all times, however due to failures this cannot be guaranteed every instance. Within each such group H_j data is assumed to be able to move around freely (an evasive storage mechanism is employed), but is not allowed to leave the group, therefore restricting the impact evasiveness has on the overall traffic within the network, but more importantly: restricting the location where data needs to be looked for.

The definition of d-Evasive Data Storage has the property of fitting nicely into the picture that has been presented up till now:

- setting d to be equal to 1, the d-Evasive Data Storage becomes the now widely assumed storage where data is fixed at h distinct hot nodes. This is depicted in the right picture of figure 4.3.

- at the other extreme, setting $d = n$, hence making the whole network available to evasiveness, the concept collapses to the Simple Evasive Data Storage Algorithm introduced in the last section. [11]

Note that there is no restriction in the definition of hot groups to disallow the possibility of having some node a with $a \in H_i \cap H_j$, for some $i, j \in \{1, \dots, h\}$. This is the reason why setting $d = n$ works out to be possible and results in the Simple Evasive Data Storage Algorithm from the last section. In most cases however, the setup of hot groups will avoid multiple membership of nodes.

[10]The symbol $*$ is used to explicitly distinguish between ordinary hot groups and loose hot groups.
[11]This presumes that the evasiveness approach used in a hot group is actually equal to Simple Evasive Data Storage.

In order realize the notion of Location Bound Evasive Data Storage it is assumed that the h hot groups are setup at pre-deployment. Hence this section will not be concerned with how to establish those hot groups autonomously without any manual pre-setup. However the Appendix carries a section titled Location Bound Evasive Data Storage - Maintenance Procedures which explains the needed procedures to use the Location Bound Evasive Data Storage notion without any pre-deployment setup. The therein presented algorithms focus on the following issues:

- *accretion mechanism*: describes how to setup nodes that form the members of hot groups.
- *member maintenance*: addresses the problem of possible node failures and aims at keeping the number of members d constant by checking for failures. The notion is to use heartbeat message and in case of failures invoke recruitment procedures to attain new nodes for the group. Furthermore, this mechanism can also take responsibility to avoid a hot group becoming a loose hot group, which can happen due to the frequent topology changes encountered in sensor networks.
- *hot group dislocation*: provides a mechanism to slowly move the complete hot group through the network, which can be seen as a swarm of hot group members moving through the network. This is essential as bounding evasiveness to a rather small region can lead to faster exhaustion of energy in those regions.

The algorithms are shown in the Appendix with pseudo codes and more detailed descriptions. But for the investigation needed from the security perspective those play a minor role and are therefore neither dealt with explicitly in this section nor in the Simulation Chapter.

Now an explanation of the details of this assumed pre-deployment setup of hot groups is needed. Basically, it is assumed that h hot groups are setup completely, satisfying the properties of the definition given for the hot group notion. Furthermore, each node is assumed to have knowledge about the location of the h hot groups, which is a similar assumption to the Global Hash Table, used in Data Centric Storage. Thus any node that wants to store data is assumed to be able to successfully transmit that data to the appropriate hot group for long term storage. Similarly, base stations have knowledge of hot groups and more precisely of the so called *head nodes*, that are assumed to exist in every hot group. These head nodes assume administrative task and will be denoted as $\hbar_1, \ldots, \hbar_m$. These nodes do not differ from ordinary sensor nodes, except that they are running a different set of algorithms and are ascribed additional knowledge about the hot group, for instance a list of all current members. There can be several head nodes in a hot group, however the base station needs only knowledge of a single one. Naturally, all members of a hot group are expected to know the head nodes located in the hot group. To obtain either a hot group or a head node respectively each node in the network can query a local database that will provide the needed information, like identification or location, to communicate with the desired party. Furthermore, each hot group is assumed to have a mechanism for the distinction of its members from other nodes in the network, here a simple identification mechanism is assumed that allows nodes from a hot group to determine which of their neighboring nodes are also part of their hot group. Thus it is assumed that each hot group is equipped with a unique identifier, referred to as $HGID$. Similarly to the assumption made about the available cryptographic primitives in sensor networks, hot groups are assumed to dispose of means allowing for secure communication among its members, for instance using a group wide encryption key.

The picture of a Location Bound Evasive Data Storage enabled sensor networks presented so far allows for the presentation of the essential distinction between the Simple Evasive Data Storage Algorithm that the previous section revealed and the Location Bound Evasive Data Storage notion this section is focused on: the modification of the New Storage Node Choice Algorithm. Algorithm 4.8 shows the pseudo code for the changes that need to be made to the algorithm 4.2 of the preceding section in order to be used for the d-Evasive Data Storage notion. The fundamental difference is that the set C that contains the nodes that are passed to the *Choice* is no longer containing the nodes that replied with a *CAN_STORE* message, but is further restricted by testing for hot group membership of the nodes that replied. This is achieved in algorithm 4.8 by sending, along with the usual *REQUEST_EVADE* message, a *GET_HGID* to each $v \in V(s)$. Each vicinity node that replies is now supposed to send its hot group identification back and therefore allow the node s put only neighboring nodes into the set C that match its own hot group identification. This simple change to the New Storage Node Choice mechanism restricts the evasion process to nodes that are members of the current hot group. The algorithm presented here bases itself on the assumption that the used Evasive Data Storage algorithm is the one presented in the preceding section, i.e. Simple Evasive Data Storage. However, the notion does not restrict itself to that single algorithm and in case other Evasive Data Storage are developed Location Bound Evasive Data Storage can be applied as well. However, for this thesis the combination of Location Bound Evasive Data Storage with the Simple Evasive Data Storage algorithm is assumed.

Participants:	
s	- the node storing data to be moved (leading node)
v	- arbitrary node in the vicinity of s
Parameters:	
$HGID(s)$	- specifies the identification of the hot group s is a member of

```
s:    upon event invocation do
  -       set $C = M = \emptyset$
  -       local broadcast ($REQUEST\_EVADE$(sizeof $D$),$GET\_HGID$)
  -       start timer $time\_out$
s:    upon event receiving ($CAN\_STORE$,$HGID$) from $v$ do
  -       if $HGID = HGID(s)$ then
  -           $C = C \cup \{v\}$
  -       else
  -           $M = M \cup \{v\}$
s:    upon event receiving ($CANNOT\_STORE$,$HGID$) from $v$ do
  -       $M = M \cup \{v\}$
v:    upon event receiving ($REQUEST\_EVADE$(sizeof $D$),$GET\_HGID$) do
  -       if sufficient storage to store data then
  -           encrypted local broadcast ($CAN\_STORE$, $HGID(v)$) to $s$
  -       else
  -           encrypted local broadcast ($CANNOT\_STORE$, $HGID(v)$) to $s$
```

Algorithm 4.8: Location Bound Evasive Data Storage Algorithm (LBEDS) - NSNC Modification

Having shown the modification of the New Storage Node Choice Algorithm, all other parts of the Simple Evasive Data Storage notion that were presented in the last section, i.e. the camouflage enhancement or the redundancy mechanisms, apply as well to the Location Bound Evasive Data Storage algorithm. Hence there is no need to repeat them. This is an advantage of the definition that allows for reusage of the algorithms as well as the results that were obtain in the last section. Now that the approach is clear, the next step is to investigate the Data Retrieval and Update in this storage notion and what advantages are entailed.

4.3.1 Data Update and Retrieval in Location Bound Evasive Data Storage

The Simple Evasive Data Storage incurred a large message overhead when performing data retrieval, which then lead to the introduction of the d-Evasive Data Storage notion of this chapter. Now it is time to show how data retrieval works in this notion, and subsequently the advantages yielded by the restriction in terms of messages needed as well as the persistency of security features that evasive data storage provides.

As with Simple Evasive Data Storage there are several possible ways to perform data retrieval in a sensor network that harbors hot groups. Hence, the simplest of those seen in the previous chapter can also be applied to the d-Evasive Data Storage notion, the Flooding Data Retrieval concept. Of course, the flooding in hot groups takes a different order of magnitude as the meaning of flooding here shifts from querying all nodes in the network, which is equivalent to a large message overhead, to all nodes in a hot group, which incurs a quite manageable amount of messages. To easily understand how flooding in the d-Evasive Data Storage notion works, the network in a network analogy of hot groups can be used: the head node $\hbar$ of a hot group becomes the base station and the nodes of the network become the hot group members, thus making a hot group an autonomous network performing flooding like in the Simple Evasive Data Storage flooding algorithm, but on a much smaller scale. Such a virtual network does not receive queries for data items from a legitimate user directly but form a real base station. Due to the assumptions made on the presetup of the hot groups in the network, a base station knows directly which head nodes it needs to contact in order to receive the data items wanted. The rest of the data retrieval is left to the head nodes.

```
Participants:
    $BS$            - base station issuing the query
    $\hbar$         - head node of a hot group $H$
    $a$             - node in the hot group $H$
Parameters:
    $type\_spec$    - the type specifier that a legitimate user queries for
    $time\_out$     - time to wait for response of head node
    $time\_out'$    - time to wait for hot group members to respond

$BS$:      upon event request for data of $type\_spec$ from $LG$ do
           -     find hot group that stores $type\_spec$ and its head node $\hbar$
           -     encrypted point-to-point $REQUEST\_DATA(type\_spec)$ to $\hbar$
           -     start timer $time\_out$
$BS$:      upon event receiving $D$ matching $type\_spec$ from $\hbar$ do
           -     encrypted point-to-point $D$ to $LG$
$BS$:      upon event $time\_out$ is reached do
           -     encrypted point-to-point $\square$ to $LG$

$\hbar$:   upon event receiving $REQUEST\_DATA(type\_spec)$ from BS do
           -     initialize $DS$ for storage of data of type $type\_spec$
           -     encrypted point-to-point $REQUEST\_DATA(type\_spec)$ to $\forall a' \in H$
           -     start timer $time\_out'$
$\hbar$:   upon event receiving $D$ matching $type\_spec$ do
           -     if $D \neq \square$ then
           -       $DS = D \rightarrow DS$
$\hbar$:   upon event receiving from all nodes replies or $time\_out'$ is reached do
           -     extract freshest data item $R$ from $DS$
           -     encrypted point-to-point $F$ to $BS$

$\tilde{a}$:  upon event receiving $REQUEST\_DATA(type\_spec)$ from $\hbar$ do
           -     if data $D$ matching type $type\_spec$ is stored then
           -         encrypted point-to-point $D$ to $\hbar$
           -     else
           -         encrypted point-to-point $\square$ to $\hbar$
```

Algorithm 4.9: Location Bound Evasive Data Storage Algorithm - Flooding Data Retrieval

Algorithm 4.9 shows how the flooding works in a hot group. As implied by the analogy above the difference is that flooding is performed by a head node $\hbar$ of a hot group H. For this to be initiated, the usual workflow is passed through: starting with a legitimate user LG who queries a base station BS for a data item of type $type\_spec$. The base station upon receiving this request finds in its local database (which is most likely realized by a global hash table) the hot group H that is responsible for storing the requested data type $type\_spec$. BS then sends a request for the respective data item to the head node $\hbar$ of H. If BS does not receive a reply within a certain time interval, it will return $\square$ to LG. In case the $\hbar$ receives the request it contacts all current members of the hot group H, i.e. it floods H with a $REQUEST\_DATA(type\_spec)$ message, which is send encrypted such that each node in the hot group can verify the authenticity of the request. If a hot node $\tilde{a}$ receives the request, it looks for a data item D matching the wanted type and, if doing so successfully, replies D to $\hbar$. The head node accumulates all the data items it receives and extracts the freshest according to the time stamp field each data item is assumed to have. [12] Then the head node sends the freshest data item it found, namely F, to the base station BS. Successfully receiving F, BS finishes the cycle by providing F to the legitimate user LG. Hence, a strong resemblance with the flooding algorithm for Simple Evasive Data Storage is unmistakable. However, the grave difference that needs to be stressed is that flooding a hot group H is less elaborate than doing so for the complete network.

Participants:	
BS	- base station issuing the query
$\hbar$	- head node of a hot group H
a	- node in the hot group H
Parameters:	
f	- frequency, how many evasion steps to wait until to do an update
$type\_spec$	- the type specifier that a legitimate user queries for
$time\_out$	- time to wait for responses of nodes

```
ħ:   upon event initialization do
     -    initialize $LOCS(type\_spec)$ for each possible $type\_spec$
ħ:   upon event receiving $(newID, oldID, type)$ from $a \in H$ do
     -    remove $oldID$ from $LOCS(type)$
     -    set $LOCS(type) = newID \rightarrow LOCS(type)$
ħ:   upon event request for data of $type\_spec$ from BS do
     -    initialize $DS$ for storage of data of type $type\_spec$
     -    foreach node $a'$ found in $LOCS(type\_spec)$ do
     -        point-to-point $REQUEST\_DATA(type\_spec)$ to $a'$
     -    start timer $time\_out$
ħ:   upon event receiving $D$ matching $type\_spec$ from $a$ do
     -    if $D \neq \Box$ then
     -      $DS = D \rightarrow DS$
ħ:   upon event receiving from all contacted nodes replies or $time\_out$ is reached do
     -    extract freshest data item $F$ from $DS$
     -    encrypted point-to-point $F$ to $BS$
a:   upon event $DSRefresh = 0$ do
     -    encrypted point-to-point $(ID(s), ID(OldRep), D.Ts)$ to ħ
     -    set $DSRefresh = f$
a:   upon event receiving $REQUEST\_DATA(type\_spec)$ from ħ do
     -    if data $D$ matching type $type\_spec$ is stored then
     -        encrypted point-to-point $D$ to ħ
     -    else
     -        encrypted point-to-point $\Box$ to ħ
```

Algorithm 4.10: Location Bound Evasive Data Storage Algorithm - Local Mark based Location Marking

The other Data Retrieval approach, referred to as Local Mark based Location Marking for Location Bound Evasive Data Storage, is similar to both of the elaborate notions defined in the previous chapter. The

[12] Algorithm 4.9 leaves the workings of the extraction part open, but they can be assumed to be equal to the mechanism the base station BS uses in the Flooding Data Retrieval algorithm for Simple Evasive Data Storage or in case a more security conscious approach is needed a majority mechanism can be applied.

resemblance with Base Station based Location Marking is that each node that displaces data in the hot group reports this to the head node $\hbar$, every time the assumed counter $DSRefresh$ reaches zero.[13] Thus the head node of a hot group takes the role of the base station making the hot group comparable to a network in a network. On the other hand the head node can be also seen as a location mark from the Local Mark based Location Marking notion, which knows where data is stored in the region it administrates and is directly queried by the base station, if data is needed. As the algorithm for Location Bound Evasive Data Storage seems to have more traits in common with the Local Mark based Location Marking concept, it is named after that notion. The pseudo code for Local Mark based Location Marking is shown in Algorithm 4.10. The code omits the interaction between the LG and the BS, as this is the same as stated in the other data retrieval algorithms, see algorithm 4.9 for instance. As indicated above algorithm 4.10 causes nodes in a hot group that store data to send a $(newID, oldID, type)$ message to the head node $\hbar$, such that $\hbar$ can track the motion of data in the hot group. Now, in case a legitimate user wants some data item D of type $type\_spec$, he queries the base station BS, which then uses a database (or global hash table) to pinpoint the hot group H responsible for storing the specific type $type\_spec$ and contacts the head node $\hbar$ that is responsible for H. Receiving a request from BS, the head node $\hbar$ can directly ask specific nodes for data matching $type\_spec$, because it constantly keeps track of them in the respective $LOCS(type\_spec)$ data structure. Once the nodes, $\hbar$ requested data from, answer, the head node can extract the freshest data item F among the ones it received and send F to BS. Obviously, the basic retrieval process is very similar to the Flooding Data Retrieval seen in algorithm 4.9, however the need for flooding is made superfluous, at the cost of more messages send at every evasive step.

Looking at both Data Retrieval approaches in the Location Bound Evasive Data Storage notion, it becomes clear that there is a huge advantage over Simple Evasive Data Storage in terms of quantity of messages needed: where Simple Evasive Data Storage needed n many messages for flooding, flooding in a hot group only takes d many messages. In order to see the difference between the two Evasive Data Storage notions more precisely the following sections take a look at the correctness and most importantly at the message complexities arising in the d-Evasive Data Storage notion compared to the complexity yielded by data retrieval of Simple Evasive Data Storage.

4.3.2 Correctness of Location Bound Evasive Data Storage

Due to the fact that Location Bound Evasive Data Storage is a refinement of the Simple Evasive Data Storage algorithms, many of the correctness results obtained for the Simple Evasive Data Storage case can be used in the argumentation for correctness of the Location Bound Evasive Data Storage notion. Starting with the New Storage Node Choice for Location Bound Evasive Data Storage it should be clear from the modifications shown in algorithm 4.8 that the only part that changes is the increased size of the request messages, as the $HGID$ needs to be requested and replied. Hence, the overall properties regarding time delays and probabilities are inherited from the unmodified New Storage Node Choice algorithm of the Simple Evasive Data Storage case. Similarly, the results obtained for the Redundancy Chain mechanism and the argumentation of all other parts of the Simple Evasive Data Storage algorithm that are used unchanged in the Location Bound Evasive Data Storage algorithm, naturally apply to the Location Bound Evasive Data Storage algorithm as well. Therefore, there is no need to restate the results of the preceding section again.

Thus what is left is to show that the Location Bound Evasive Data Storage and the two presented Data Retrieval approaches, Flooding Data Retrieval and Local Mark based Location Marking do satisfy the three basic Data Storage properties **Availability**, **Correctness** and **Freshness**. Starting with the Flooding Data Retrieval notion, the argumentation, that this data retrieval mechanism satisfies the mentioned properties, resembles to a high degree the one given for the respective Simple Evasive Data Storage case. The major change that needs to be taken care of is that probabilities and time delays have to be modified slightly because of the additional communication between the base station and the head node of the respective hot group. All other parts remain the same, as the working of the Flooding Data Retrieval is the same in the hot group as it

[13] See the Simple Evasive Data Storage section for details on this counter.

is in the whole network for the Simple Evasive Data Storage case. Because no fundamental changes occur, the implications regarding the appropriate choice of parameters and their behavior are the same as for the Simple Evasive Data Storage case and need not to be repeated. Similarly, a repetition of the exact formulas, that were obtained in the previous chapter is unnecessary and would not offer any new insights. However, what remains noteworthy is that the probability involved in the **point-to-point** primitive is less variable, due to the overall shorter distances that flooding in a hot group incurs compared to flooding in a network. This, of course, is only of minor concern as retransmission can handle the undesired effects of unreliable channels to a satisfactory degree, but will demand for a lower need for retransmissions compared to the Simple Evasive Data Storage data retrieval algorithm. Analogously, the time delay for the **point-to-point** is also more stable, than it would be when the whole network is considered, which is naturally caused by the shorter distances of nodes as well. The second notion that has to be addressed is that of Local Mark based Location Marking, which can also be used as a data retrieval approach in Location Bound Evasive Data Storage. However, with the same argumentation as with the Simple Evasive Data Storage for the Base Station based Location Marking data retrieval, an exact evaluation is left out, because the results would be too similar to the already presented analysis. The main issue however, that cannot be left out, is the comparison of the retrieval concepts according to the message complexities which they incur. Hence, more attention needs to be given the following section, leaving the formal correctness to be seen as a slight deviation from the thorough one provided in the Simple Evasive Data Storage section.

4.3.3 Complexity of Location Bound Evasive Data Storage

As Location Bound Evasive Data Storage is a refinement of the Simple Evasive Data Storage approach, which aims at bettering several unpleasant message complexity issues encountered in it, the complexity of Location Bound Evasive Data Storage should reveal the advantages that this refinement implicates. However, in this investigation the complexity of the predeployment setup procedures is ignored and only the algorithm parts used at runtime are evaluated.

The basic ingredients of Location Bound Evasive Data Storage are the same or similar to those that were used in the Simple Evasive Data Storage algorithm. The Displacement Trigger algorithm needs no modification, i.e. algorithm 4.1 works for Location Bound Evasive Data Storage, thus its complexity does not change. The important New Storage Node Choice algorithm, however, has to be modified (see algorithm 4.8 for the respective modifications) to be used in this context. Looking at algorithm 4.8, it becomes obvious that only the message sizes change, because of the additional hot group identification that is attached to some request and reply messages, in order to guarantee displacement of data to hot group members only. Hence, the complexity of $\mathcal{O}(r)$ obtained for the Simple Evasive Data Storage algorithm persists. Similarly, the Redundancy Chain notion is used insignificantly modified in the hot groups to provide resilience against failures and therefore also has the same message complexity of $\mathcal{O}(r+l)$ in the Location Bound Evasive Data Storage case. Those message complexities do not make any advantages of the location bound refinement obvious, but that should be clear as the true refinement has not changed those parts, but rather changed the location where they can be used. Thus the advantage of the Location Bound Evasive Data Storage refinement will be obvious in the subsequent analysis of the Data Retrieval complexities.

As with Simple Evasive Data Storage two Data Retrieval options are available and need to be investigated: the Flooding Data Retrieval algorithm, which in the simple evasive notion exhibited query costs of $\mathcal{O}(n)$ and event costs equal the redundancy chain costs of $\mathcal{O}(r+l)$, and Local Mark based Location Marking which is similar to the Base Station based Location Marking algorithm, that incurred query costs of $\mathcal{O}(\bar{h} \cdot \sqrt{n})$ and event costs summing up to $\mathcal{O}(r+l)$ plus additional $\mathcal{O}(b \cdot \sqrt{n})$ every f evasion steps. Representing the Flooding Data Retrieval notion, algorithm 4.9 allows a legitimate user LG to ask a base station BS for data of a specific type *type_spec*, which then contacts the head node $\hbar$ of hot group H responsible for storing type *type_spec* and waits for a single reply containing the data wanted. This request for data, incurs approximately $\mathcal{O}(\sqrt{n})$ messages, for both request and eventual response. As $\hbar$ is most probably not the storage node for the wanted data item, the node floods H in order to find the appropriate hot nodes, resulting in additional $\mathcal{O}(d)$ many messages accounting for the flooding. Putting those two actions together, the query from BS to $\hbar$ and the flooding of H by the head node $\hbar$, the query costs for Flooding Data Retrieval in Location Bound Evasive Data Storage amount to $\mathcal{O}(\sqrt{n}+d)$. As each hot group is assumed to use New Storage Node Choice in the modified version each evasive step adds $\mathcal{O}(r+l)$ messages to the event costs. However, before the location bound evasion can begin for a data item D, the generating node first needs to send D to the appropriate hot group, thus adding to the event costs $\mathcal{O}(\sqrt{n})$ messages. This is only done once and thereafter only the cost for an evasive step dominate the event costs. Comparing those complexities to the Simple Evasive Data Storage case, it is obvious that data query has become by far less demanding, as the flooding needs only d many messages compared to a (possibly) several magnitudes larger n number of messages.

The second Data Retrieval for Location Bound Evasive Data Storage is the Local Mark based Location Marking presented in algorithm 4.10. Here the same amount, namely $\mathcal{O}(\sqrt{n})$ is needed for the request from the base station to the head node and the later reply. However due to the fact that the head node $\hbar$ keeps track of the nodes in the hot group H that store data, the additional number of messages to obtain the data is $\mathcal{O}(\sqrt{d})$: no flooding is needed, $\hbar$ can contact the hot nodes directly, resulting in the according decrease in needed messages. Assuming that the hot group H contains $\bar{h}$ many hot nodes matching the needed type, the overall query costs for this Data Retrieval add up to $\mathcal{O}(\sqrt{n}+\bar{h}\cdot\sqrt{d})$. Hence the Local Mark based Location Marking notion obtains a better complexity over the Flooding Data Retrieval in the query case. The cost of $\mathcal{O}(r+l)$ for the evasive steps remain the same, as no part in the algorithm changes how data is displaced. However, in order to be able to avoid flooding and ask the right hot nodes for data, the Local Mark based Location Marking needs nodes to send their location to the head node $\hbar$, incurring costs of $\mathcal{O}(\sqrt{d})$ every f evasion steps.[14] Hence, this approach shifts the costs from flooding overhead to constant update messages overhead. The two notions and their respective message complexities are summarized in table 4.3, where it can be seen that the main advantage of Location Bound Evasive Data Storage compared to Simple Evasive Data Storage is that the dominating variable changed from a large n to small d.

Costs:	**Flooding Data Retrieval**	**Local Mark based Location Marking**
Query Costs:	$\mathcal{O}(\sqrt{n}+d)$	$\mathcal{O}(\sqrt{n}+\bar{h}\sqrt{d})$
Event Costs:		
- Initial:	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$
- Every Evasion Step:	$\mathcal{O}(r+l)$	$\mathcal{O}(r+l)$
- Every f Evasion Step:	$-$	$\mathcal{O}(\sqrt{d})$

Table 4.3: Location Bound Evasive Data Storage Data Retrieval Costs Overview

There does not seem to be an advantages of one particular Data Retrieval approach over the other, the Flooding Data Retrieval floods the hot group (or the whole network), but does not need additional messages to be sent at each evasive step, and on the other hand the Local Mark based Location Marking algorithm [15] removes the need for flooding, but adds additional update messages allowing the tracking of data. Thus non of those two concepts can be seen superior over the other, however if an estimate can be made over the dominating purpose of the network, or more precisely what operation will be performed more often, then a decision can be made as to which of the two algorithms is superior. Supposing that Q, the number of queries, is larger than the number of events E, then the Local Mark based Location Marking algorithm will be a better choice, because query costs do not involve any flooding and as the number of events is low the respective amount of additional messages is bearable. In the opposite case, where E dominates Q ($E > Q$), then the situation changes and flooding can entail less overall messages, as the larger number of events does not need to be chaperoned by additional tracking messages. Hence, the choice among the two approaches, both in the Simple Evasive Data

[14] For an explanation concerning the frequency f refer to the previous section on Simple Evasive Data Storage, where this variable has been described in more detail.

[15] In the Simple Evasive Data Storage this can be seen as the Base Station based Location Marking algorithm .

Storage and the Location Bound Evasive Data Storage case depends on the relation between the two values of Q and E respectively. Summarizing, the usage of the Location Bound Evasive Data Storage can yield more efficient storage costs than the usage of the Simple Evasive Data Storage algorithm can incur. Hence, the notion of hot groups can positively influence is overall applicability of the Evasive Data Storage notion. What still needs to be addressed are issues regarding the security provided by the Location Bound Evasive Data Storage notion, just as it has been investigated for the Simple Evasive Data Storage of the previous section.

4.3.4 Security of Location Bound Evasive Data Storage

As with Simple Evasive Data Storage it is important to investigate some of the security properties the Location Bound Evasive Data Storage algorithm exhibits, which focuses in this section mostly on the probability distribution for hot nodes in idealistic circumstances and on several short looks at the behavior in the presences of active adversaries. Of course the more interesting results will be obtained in the Simulation Chapter, where also the properties in context with Camouflage Techniques can be investigated.

One of the issues that can still be discussed beforehand, is the success probability an adversary $\mathcal{A}(blind, \star)$ has that has not gathered any information at all about the network or the respective hot nodes contained within it. Thus such an $\mathcal{A}$ is forced to choose a node completely at random. Assuming that each node has equal probability of becoming a hot group member and that each node in a hot group has an equal likelihood of storing data, the same probability for $\mathcal{A}$ results as in the Simple Evasive Data Storage case:

$$P(adversary\ picks\ a\ hot\ node) = \frac{h}{n}.$$

This is clear as the system harbors h many hot nodes and each of the n nodes in the network is ascribed an equal probability of becoming hot. This result can also be obtained more precisely using a hypergeometrical distribution, as done in the case of Simple Evasive Data Storage, but is omitted here.

The next step is to assume that an adversary $\mathcal{A}$ has knowledge about a hot node $\tilde{a}$ and investigate his success probability of obtaining data at a later point in time given such information. Making the same assumptions as in the Simple Evasive Data Storage case, the number of nodes that could store data of $\tilde{a}$ after e steps amounts to approximately $\min\{\mathcal{O}(e^2 \cdot r), d\}$, as the region that can be reached by the evasion starting at $\tilde{a}$ cannot exceed the number of nodes in the hot group. This is different to the Simple Evasive Data Storage algorithm, which did not pose such a restriction, except for the natural bound n, which cannot be exceeded. Hence, an adversary that identifies the boundaries of a hot group, can always find a hot node with a probability of $\frac{1}{d}$, or better in case $\mathcal{O}(e^2 \cdot r) < d$. Naturally, the estimates above assume that after e evasion steps the region of possible hot nodes exhibits an uniform distribution, however, just as in the Simple Evasive Data Storage notion, this cannot be assumed. A more detailed investigation will be provided in the Simulation Chapter, especially when the choice functions are looked at.

As with the Simple Evasive Data Storage notion a look at the behavior in presence of an adversary with strong intervention skills should be taken. Assuming an adversary $\mathcal{A}_g(\star, active)$, an important issue becomes the security differences between the flooding approach and the location mark approach for data retrieval in the sensor network. The adversary $\mathcal{A}_g$ can send arbitrary location mark update messages to the head node, when having taken over a node from the respective hot group. In the flooding approach such doing is not possible, as no such messages are needed. However when sending such fake location mark messages $\mathcal{A}_g$ can only add his own nodes into a particular tracking list of the head node $\hbar$ and delete nodes from the list $LOCS(type\_spec)$, if he knew the respective identifications. However, knowing these is only possible if $\mathcal{A}_g$ overtakes the head node $\hbar$ itself, thus the damage $\mathcal{A}_g$ can cause is limited to infiltrating the set of responses the head node $\hbar$ receives when requesting data from the nodes found in $LOCS(type\_spec)$, which can be avoided by a majority based modification of the data extraction $\hbar$ performs and the assumption that the number of malicious nodes $\mathcal{A}_g$ can put into the list $LOCS(type\_spec)$ is less than the majority. Several other issues that are found in the Simple Evasive Data Storage given a strong adversary, naturally apply to the Location Bound Evasive Data Storage notion as well, but a repetition of those is superfluous as it was already given in the previous section.

4.4 Time Constraint Evasive Data Storage

The last section introduced a refinement to the Simple Evasive Data Storage by restricting the region where the evasion process could propagate. The result is titled as the Location Bound Evasive Data Storage notion. The refinement process is taken another step higher in this section in form of the (d, t)-Evasive Data Storage notion. The idea is to add a restriction on the time that data can reside at a hot node until the evasion is forced. Up till now, both the Simple Evasive Data Storage and the Location Bound Evasive Data Storage algorithms used the Displacement Trigger algorithm part that determined the evasion of data parameterized by the two values of *time_int* and p_e respectively. Thus the evasion takes place after a random time interval. This can be seen as a restriction to the applicability of Evasive Data Storage in a sensor network, hence the (d, t)-Evasive Data Storage notion forces a maximal storage period of t time units, after which the evasion of data is forced to take place. The definition of this notion allows for a nice connection to the two notions of Evasive Data Storage presented so far: setting $t = \infty$ yields for (d, t)-Evasive Data Storage the notion of the last section, i.e. d-Evasive Data Storage. Similarly, setting $d = n$ and $t = \infty$ the resulting algorithm exhibits the properties of Simple Evasive Data Storage. Hence, (d, t)-Evasive Data Storage can be seen as a generalization of all the notions introduced so far and, of course, also of Non-Evasive Data Storage concept with h fixed hot nodes.

Participants:	
s	- the node storing data to be moved (leading node)
Parameters:	
t	- defines time interval that a data item can maximally reside at one single node
D	- the data item which is to be moved

```
s:    upon event receiving or generating $D$ do
 -        start timer $time\_int$, $t$
 -        set $force = false$
s:    foreach $time\_int$ do
 -        with probability $p_e$ do
 -            invoke SEDS-NSNC(D)
s:    upon event $t$ is reached do
 -        stop timer $time\_int$
 -        set $force = true$
s:    upon event $force = true$ do
 -        set $force = false$
 -        invoke SEDS-NSNC(D)
s:    upon event failed evasion and $t$ elapsed do
 -        set $force = true$
s:    upon event successful evasion do
 -        set $force = false$
 -        stop timer $time\_int$, $t$
```

Algorithm 4.11: Time Constraint Evasive Data Storage Algorithm (TCEDS) - Displacement Trigger

The technical realization is a minor modification to the Displacement Trigger algorithm part of Simple Evasive Data Storage, i.e. the algorithm 4.1. In addition to the time interval *time_int* that is part of the displacement decision, nodes keeps track of the passed time since the particular data item has been stored in the node. Once the storage time exceeds the time limit t, specified as a parameter, the node is forced to commence the displacement procedure, i.e. invoke algorithm 4.2 or 4.8. Naturally, before the t time units have elapsed there is still a probability, which depends on the probability[16] p_e, that displacement will take place even

[16]Refer to section on Simple Evasive Data Storage for details on probability p_e.

before the timeout t is reached. Of course, because the New Storage Node Choice algorithm does need some finite time to complete, the completion of the evasion is actually $t + \epsilon$ in the worst case. The modifications needed to achieve the (d, t)-Evasive Data Storage notion apply only to the Displacement Trigger algorithm part and are shown in algorithm 4.11.

Algorithm 4.11 is very similar to the one seen in the previous sections, but with the addition of t as parameter and the respective initialization of the timer for t. In the period before t is reached, the *time_int* and the p_e parameters govern the evasion and a *failed evasion* event does not affect the trigger, as the t has not elapsed yet. In case the *successful evasion* event is raised before t is reached, the trigger stops both timers for *time_int* and t respectively, in a sense terminating the algorithm. However assuming that t is reached, hence no successful evasion occurred in the time before, the behavior is different: the *time_int* timer is stopped such that no evasions are triggered anymore by it and a variable $force$ is set true. This variable indicates that evasion is forced, causing the New Storage Node Choice algorithm to be invoked until the evasion is successful. Hence, the maximal time for data to remain at a hot node running algorithm 4.11 is defined by t and by the time that the New Storage Node Choice algorithm needs to be successful, which can be seen as a rather small time addition of ϵ. Therefore the maximal time amounts to $t + \epsilon$. It can be seen that such a rather small change to one of the algorithm parts of the Simple Evasive Data Storage notion, can introduce a refinement that proposes an interesting security advantage in special cases.

The security advantage is especially interesting from the perspective of a $(\kappa, \Delta) - \mathcal{A}(\star, \star)$ adversary, who has a restriction on the time he needs to access the κ nodes. Hence such a slight shift in the behavior of the displacement decision can open up interesting aspects, e.g. when considering special restrictions on adversaries. A more detailed investigation of this notion is provided in the following section including comments on correctness and complexity. However, the Time Constraint Evasive Data Storage will not be investigated in any more detail in scope of the Simulation Chapter, as the results obtained there for the Simple Evasive Data Storage should be sufficient to indicate the advantage the Time Constraint Evasive Data Storage can offer.

4.4.1 Correctness, Complexity and Security

In order to better understand the Time Constraint Evasive Data Storage notion several of its formal properties need to be mentioned. However, as the notion is just a slight modification of the Simple Evasive Data Storage and the Location Bound Evasive Data Storage notion respectively, most of the results obtain for those are relayed to the Time Constraint Evasive Data Storage concept. Looking at the correctness of the algorithm, it is clear that all the Evasive Data Storage ingredients, like the New Storage Node Choice or the Redundancy Chain part, are correct, because they are used unmodified. The only difference is the modifications applied to the Displacement Trigger, which however can easily be seen as correct.

Similarly, the complexity properties are inherited from the refined notions and apply to Time Constraint Evasive Data Storage as well. Furthermore, the Displacement Trigger does not entail any communication messages, and thus needs no complexity evaluation. The complexities which apply are summarized in table 4.3 and 4.2 respectively in the preceding sections. The security properties of the parts other than the Displacement Trigger, are discussed at the security sections of the refined notions. The significant difference however arises when assuming an $(\kappa, \Delta) - \mathcal{A}(\star, \star)$ adversary. Having seen the working of the time restriction in algorithm 4.11, it is obvious that a $(\kappa, \Delta) - \mathcal{A}(\star, \star)$ adversary can be forced to be unsuccessful at every attempt he makes to access a hot node by making the relation $t < \Delta$ hold true. Such a setup will yield that even when an adversary tries to access a node that stores data, he will need too much time. Before the adversary is successful, i.e. after time Δ, the data stored at the node has already been displaced to another node, due to the restriction on the storage period which cannot exceed t.[17] Thus such an adversary can only achieve his goal of accessing data in a network running a Time Constraint Evasive Data Storage algorithm with the t parameter set to the right value, when he matches a $\mathcal{A}(\star, active)$ model and thus can take over node, which then wait for data to arrive and access data directly at its arrival time. There are several other variations, especially concerning the values for t and Δ

[17]This of course assumes that no redundancy chain mechanism is used, otherwise the restrictions on the time need to be made even more severe depending on the length of the chain.

that are of interest, but are left for future work and thus will not be investigated in the scope of this work. The argumentation thus shows that having a time restriction built into the Evasive Data Storage can offer interesting security enhancements, when adversaries obey similar restrictions. Thus making the Time Constraint Evasive Data Storage another interesting contribution to the family of Evasive Data Storage algorithms.

4.5 SUMMARY

This chapter revealed the notion of Evasive Data Storage and showed how the notion can be used to address the problem of an adversary remembering interesting nodes in the sensor network. The chapter introduced the Evasiveness Security Property which will be used in the Simulation Chapter to evaluate the Simple Evasive Data Storage and Location Bound Evasive Data Storage algorithms respectively. Of course, the approach of this chapter is different from the one presented preceding chapter, where the emphasis was put on the adversaries ambition of finding and recognizing interesting nodes. The three possible implementations introduced in this chapter for the Evasive Data Storage notion form a solid basis for a vivid analysis of the security properties entailed, but still leave some room for bolder extensions. Several of those extentions are shown in the following chapter. Besides, a mere introduction of the Evasive Data Storage algorithms, the chapter also investigated several of their properties given the model of sensor networks defined in the Data Security and Sensor Networks Chapter. Furthermore, a major contribution of this chapter is the introduction of the choice function concept and how the given four fundamental choice functions can be combined to form more complex ones. Hence making this chapter one of the most important ones of the whole thesis and providing sufficiently material for the following two chapters, regarding both extendibility and thorough analysis.

Chapter 5

Advanced Data Oriented Techniques

The last chapter gave a detailed description of the basic possibilities the Evasive Data Storage notion has to offer. It developed a very structured approach to implement this notion and gave several insights into the advantages as well as limitations. This chapter intends to continue to explore the notion of Evasive Data Storage in the scope of more elaborate and mostly experimental ideas. This chapter offers a variety of possible paths that can be taken in extending the evasiveness notion. Thus this chapter can be seen as presenting several hints as to where to go beyond what this work wants to present. [1]

5.1 Evasive Data Storage with Data Splitting

The notion of Evasive Data Storage incurs one particularly unpleasant point of attack that an adversary $\mathcal{A}(\star, active)$ can exploit to easily obtain data: with no need to conduct traffic analysis or look for data in the region surrounding a former hot node, $\mathcal{A}$ can take over nodes and wait for data to pass through when being displaced from one node to another on their journey through the network. This problem arises in all of the three notions described in the previous chapter. Of course, in case of the Location Bound Evasive Data Storage the adversary $\mathcal{A}$ needs to position his malicious nodes in hot groups, otherwise this approach will not yield any fruits. The notion of Data Splitting addresses this problem, by applying the simple idea of splitting a data item D into several subitems, referred to as *splints*, and storing those items separately instead of the original D. Assuming a conventional storage mechanism, Data Splitting would force an adversary to open up several nodes before successfully claiming a complete data item, which surely offers interesting advantages, this is however bought at the cost of additional redundancy and increased query costs. Similarly, the probability of accessing a hot node storing a splint will increase accordingly. In the case of any of the Evasive Data Storage algorithms seen so far, Data Splitting can also be applied. The main problem caused by doing so is how the costs for retrieval change under an increase of nodes that need to be queried. Fortunately, the query costs can be kept down as long as one of the appropriate Flooding Data Retrieval algorithms is used, thus making Data Splitting most efficiently usable in the Location Bound Evasive Data Storage notion. But does Data Splitting increase the security in an evasive storage algorithm? Of course, all the splints of a data item are still wandering through the network and thus will eventually be collected by the adversary's malicious nodes, however the time until all the splints of a single data item are collected can be significantly higher in comparison to the case where no Data Splitting is performed. In most cases the adversary will need to keep a lot of incomplete splints in a buffer until he finally can extract a single complete data item. This observation is especially useful when the adversary $\mathcal{A}$ is restricted with regards to the time he can maintain his malicious nodes: when $\mathcal{A}$'s available time is less than the expected time its takes to collect all splints of a data item, the approach does provide a significant enhancement.

The pseudo code that implements the notion of l-Data Splitting is presented in algorithm 5.1. The main parameter that influences the properties of the algorithm is l, the number of splints that the splitting is supposed to generate. The steps algorithm 5.1 performs are simple: instead of storing the data item D with an Evasive Data Storage algorithm, the data item is preprocessed, split and then each of the resulting splints is passed for storage with one of the Evasive Data Storage algorithm of the previous chapter. The preprocessing

[1] As the notions introduced here are intended to outline possible enhancements, a thorough analysis is omitted. In the same sense an investigation in the Simulations Chapter, except for the Evasive Data Storage with Data Splitting notion, is left out as well.

begins with choosing a permutation P, which can be imagined as any procedure that distributes the information contained in D as equally as possible among the l splints, such that after splitting a permuted data item a single splint does not provide any significant information to a potential adversary capturing it. It can be assumed that the symmetric encryption system that is used offers methods to generate such permutations with close properties to the ones wanted. The purpose of the permutation is to ensure that an adversary cannot extract easily any useful information from a single splint or several splints until he obtains all of them. Hence, once the permutation P is available, a new data item $D' = P(D)$ is generated, which is then divided into splints of equal size $D' = d_1, \ldots, d_l$.[2] One of those splints, by convention d_1, is augmented with the permutation P, in order to allow later reconstruction of the original data item D. Furthermore, the order of the splints needs to the taken into account too, otherwise the reverse process of obtaining the D from the splints is not guaranteed to work correctly. After the preparatory work the splints are stored evasively.

```
Participants:
    s      - node that wants to split data
Parameters:
    D      - the data item to be split
    l      - the number of splints the data item D should be split into

    s:     - upon event generating data item $D$ do
           -     choose some permutation $P$ and apply it to the data: $D' = P(D)$
           -     the resulting permuted data $D'$ is split up into $l$ smaller splints $D' = d_1, \ldots, d_l$
           -     one splint $d_1$ is augmented with the permutation $P$, resulting in $d'_1 = (P, d_1)$
           -     all of the splints $d'_1, d_2, \ldots, d_l$ are stored evasively
```

Algorithm 5.1: Evasive Data Storage with Data Splitting

How exactly the splints are stored is left open. In the Simple Evasive Data Storage there are no options other than treat each splint as a normal data item and allow for arbitrary wandering in the network. Of course, there need to be made several adjustments to the data retrieval procedures in order to handle the splints correctly, but as those modifications are minor they will not be investigated in detail. The Flooding Data Retrieval algorithm will reach all splints and provide them to the respective BS which puts them together to obtain the original item. The Location Bound Evasive Data Storage approach, however, offers several variations that can exhibit different security properties depending on the assumptions made. The most convenient is to store all the splints $d'_1, d_2, \ldots, d_l$ of a specific data item into the hot group matching the items type. It is obvious that the data retrieval on behalf of the head node of the respective hot group needs to be adjusted in order to put the splints together at query time of the base station. A variation is that the splints are separated to several hot groups instead of being located at a single same one. In order to do so, several hot groups need to intersect in their responsibilities regarding what data types they store. This is just a minor modification to the pre-deployment setup of the global hash table used by the base stations and sensor nodes respectively, to return a list of all hot groups matching a specific data type. Thus a node that generated splints $d'_1, d_2, \ldots, d_l$ can decided to store the splints in a several hot groups applying some distribution scheme, for instance equal distribution among all the hot groups matching the splints data type. The main advantage of this approach is that an adversary is very unlikely to have malicious nodes in all of the hot groups that are used for storing the splints. Hence, his efforts will be futile when he has not infiltrated all the hot groups responsible for a specific type, leaving him with several splints missing. Of course, such a sophisticated scheme entails additional costs at the time a query is initiated as now instead of a single hot group several have to be queried.[3]

Furthermore, Data Splitting has several advantages that fit nicely into the sensor network context: Data

[2]If the size of D is not a multiple of l then either some kind of padding is applied or the last splint d_l is left with a smaller size than the others.

[3]Note that the details of the data retrieval, which calls for additional information in each of the splints is left out, as it does not provide significant insights into the security properties of the notion.

Splitting does not require lengthy computations, but still can contribute enormously to the overall security. An adversary that wants to obtain split data needs to find and collect all l data splints in order to reconstruct the original data, which greatly complicates the adversary's task. In case of a restricted adversary of the form $(\kappa, \infty) - \mathcal{A}(\star, \star)$, a l parameter satisfying the relation $\kappa < l$ will cause $\mathcal{A}$ to be unsuccessful for sure, because he will never collect all of the necessary splints to obtain the original data item (when ignoring the active case and possible malicious nodes). Thus Data Splitting can not only make the adversary's task more time consuming, but also prevent him from being successful, when certain restrictions apply. Of course, in the course of the simulations to come such restrictions will be of less interest and the ones where the adversary has a greater probability than zero for obtaining the original data item will be analyzed.

Summarizing, the notion can offer several advantages and those will be evaluated in more detail in the Simulation Chapter along with the fundamental Evasive Data Storage algorithms of the previous chapter. It is noteworthy that this is also the only one of the advanced data oriented techniques that will be addressed in the Simulation Chapter, hence the following notions are purely for the exploration of ideas without any proof of their usefulness.

5.2 Evasive Data Storage with Authentication

As sensor networks evolve, more and more primitives will be available to be used in the design of applications and naturally in notions like Evasive Data Storage as well. Currently, most algorithms that demand security make use of basic encryption based on simple symmetric cryptography, [4] but currently another emerging primitive, which has been around for a long time in non-resource restricted networks, is beginning to be available in the sensor network realm: authentication. Work due to Benenson, Freiling (Gärtner) and Kesdogan in this area and their respective papers (see [7] as well as [8]) allow for an excursion on a possible incorporation of authentication within an Evasive Data Storage approach, which this section is devoted to.

The main part where authentication can be come in handy is at the core of all the presented Evasive Data Storage algorithms, the choice of the node where a data item is to be displaced to. Imagining a scenario where a sensor network is storing data and an adversary $\mathcal{A}(\star, active)$ captures up to t nodes and thus controls all data coming in and out of those malicious nodes, evidently such a scenario poses a vast obstacle for displacing data. Namely, the possibility that data falls easily into the adversary's hands by being displaced to a malicious node. This has been addressed in the context of the Data Splitting notion of the previous section, however the notion does not prevent it from happening but only can contribute to alleviate its impact. With the availability of authentication, however, adversary $\mathcal{A}$ cannot just wait until interesting data items are displaced to his malicious nodes. An effective authentication mechanism can provides a tool to pin down the malicious node (the nodes cannot authenticate themselves properly) and hence completely avoid displacing data into the adversary's palm.

In order to achieve this highly desirable authentication in combination with Evasive Data Storage an addition to the choosing procedure used in the New Storage Node Choice algorithm is called for. Of course, the actual authentication is assumed to be available as introduced [7]. The details are presented in the respective papers, here a procedure *authenticate* is assumed that takes two parameters: the first one is a vicinity set $V(a)$ of a node a that needs to be authenticated and the identification $ID(v)$ of the node v that wants to authenticate a. Thus the basic setup is for v to send an request to authenticate to node a, which then calls the *authenticate* procedure. The underlying authentication system is assumed to raise success or failure events, in case the authentication fails or node a does not respond. This includes a timing mechanism that guarantees one of the two events to be raised after some finite time. This is necessary to allow the connected evasion to proceed and not come to a standstill.

Algorithm 5.2 describes such an authentication, which is supposed to be called by the New Storage Node Choice algorithm. There are two basic ways to use the algorithm. The first one is to call algorithm 5.2 before the appropriate choice function is applied. Hence, setting $A = V(s)$ and k to some appropriate value. The algorithm 5.2 generates a set C of cardinality k, which consists of trustworthy nodes only and is then passed

[4]In most parts due to the influential work on the SPINS Protocol Suite, see [10] as reference.

Participants:	
s	- node needing to authenticate nodes
a	- arbitrary node in the vicinity of s
Parameters:	
k	- number of nodes that should maximally be authenticated
A	- set of nodes to authenticate nodes from, with $A \subseteq V(s)$
Returns:	
C	- set of nodes that are authenticated and can be used; possible that $\lvert C \rvert \leq k$

```
s:    upon event invocation do
      -     set candidate set $C$ equal to $\emptyset$
      -     set failure set $F$ equal to $\emptyset$
      -     randomly choose a new candidate $a \in A \setminus F \cup C$
      -     encrypted point-to-point $AUTH\_REQUEST$ to $a$

s:    upon event successful authentication of $a$ do
      -     $C := C \cup \{a\}$
      -     if $F \cup C \neq A$ then
      -         randomly choose a new candidate $a \in A \setminus F \cup C$
      -         encrypted point-to-point $AUTH\_REQUEST$ to $a$

s:    upon event failed authentication of $a$ do
      -     $F := F \cup \{a\}$
      -     if $F \cup C \neq A$ then
      -         randomly choose a new candidate $a \in A \setminus F \cup C$
      -         encrypted point-to-point $AUTH\_REQUEST$ to $a$

s:    upon event $F \cup C = A$ or $|C| = k$ do
      -     return $C$

a:    - upon event receiving $AUTH\_REQUEST$ from $v$ do
      -     invoke $authenticate(V(a), ID(v))$
```

Algorithm 5.2: Evasive Data Storage with Authentication

on to the used choice function, which can then pick any node from the provided set. However, this can be inefficient if the cost for the k successful authentications are large. Furthermore, the effectiveness of the choice function can be diminished if k is too restrictive. A different approach is to apply the choice function of the New Storage Node Choice algorithm first, thus obtaining the identification ID of the chosen node. Then algorithm 5.2 is called with parameter $k = 1$ and $A = \{ID\}$, causing it to attempt to authenticate the chosen node only and in case of failure returning $C = \emptyset$, which the New Storage Node Choice can use to allow the choice function to pick a different node. Of course, this is more efficient as only one node needs to be authenticated, but risks several repetitions in case the choice function does not pick non-malicious nodes.

It is clear that the approach does provide a significant advantage in terms of security as malicious nodes are almost certainly excluded from the evasion process. Of course, a perfect exclusion depends on the properties of the authentication primitive and surely cannot be guaranteed in all cases. A thorough evaluation of the combination of authentication and the Evasive Data Storage notion cannot be given, because no practical implementation is available. However, there should be sufficient arguments in favor of pursuing such a combination.

5.3 Partial Evasive Data Storage

Both of the modifications presented in this chapter so far, change the algorithms for Evasive Data Storage such that better security properties are achieved. Both of them do so by modifying the algorithms directly through additional primitives or changes to the data items themselves. However, a different point of view on the idea of evasiveness can also provide interesting properties without even needing any particular modifications to the algorithms of the previous chapter. The shift of perspective is to allow for both ideas the conventional storage and the Evasive Data Storage notion to cooperate and to be combined, which leads to the Partial Evasive Data Storage notion. To do so a predefined threshold can be used that specifies what percentage of gathered data should be stored evasive leaving the rest to be stored conventionally. Thus the network would consist of fixed hot nodes that do not change over time, providing the retrieval costs of conventional storage and similarly lower security, and hot nodes that displace their data according to an Evasive Data Storage algorithm. Hence, the threshold allows for controlled shifting between a sensor network with only fixed hot nodes and one where all the data wanders around. Mixing both strategies can add to the irritation of an adversary, because in homogenous cases where only conventional or Evasive Data Storage is employed, he can easier build a model as to how to interpret traffic in the network and exploit this to extract information as to where information is stored. Mixing both worlds imposes a more difficult model of the network, thus decreasing the advantages an adversary has in homogeneous networks employing one storage strategy exclusively.

The combination of conventional storage and Evasive Data Storage can also offer tremendous advantages when the Data Splitting notion from the beginning of this chapter is added into the picture. It can be argued that storing all the small splints that method generates separately in an evasive fashion can lead to a vast message overhead in the network, therefore draining the available energy and shortening the overall lifetime of the network. What is needed is to find a way to have splints in the network, but keep the overhead of evasiveness low. One obvious attempt to do so is do apply the Partial Evasive Data Storage idea and store only c splints evasively ($c \geq 1$) and the rest conventionally. More precisely, given a data item D, out of the resulting splints $d_1', d_2, \ldots, d_l$ generated by the Data Splitting algorithm, the first c splints $d_1', d_2, \ldots, d_c$ are given to the Evasive Data Storage used in the mixed storage approach and the remaining splints $d_{c+1}, \ldots, d_l$ are stored at fixed hot nodes. Doing so will make the retrieval more complex, but will not exhibit the message complexity of storing all the splints evasively. Furthermore, from the security perspective an adversary needs to find both types of hot nodes, fixed ones and evasive ones, which can be a complicated task as the evasive splints can wander in a completely other part of the network than the fixed hot nodes are located in storing the complementary splints. Effectively the threshold c controls the evasiveness of the network, setting c equal to 0 results in a network using conventional storage only and setting $c = l$ yields the original notion of Data Splitting in Evasive Data Storage as described at the beginning of the chapter.

There are several other ways of how Partial Evasive Data Storage can manage to combine both, increase of security due to Evasive Data Storage and less message overhead that is provided by the conventional mechanisms. However the detailed comparison and best values for the thresholds mentioned above will not be investigated in the scope of this work. The focus of the remaining chapters will be on the properties of the fundamental algorithms from the Camouflage and Evasive Data Storage realm.

5.4 Sensor Aware Evasive Data Storage

The last notion falling into the advanced modifications of the Evasive Data Storage approach calls for usage of a sensor node's basic capabilities: its sensing ability. Assuming that the provided sensors that each sensor node of the network disposes of can detect the motion of an adversary, the Evasive Data Storage can offer a highly interesting way to protect the data in the sensor network. A similar notion but from a quite different perspective has been explored in [22], where a pursuer and an evader are postulated to reside in the network. The pursuer tries to catch the evader without any direct connection to Evasive Data Storage or the goal of protecting data. The idea behind Sensor Aware Evasive Data Storage, however, is solely focused on the usage of Evasive Data Storage algorithms to protect data, which due to the assumed sensor capabilities also bases itself on a completely different model than the work found in [22]. Hence, the basic functionality of the Simple Evasive Data Storage is to use the sensors to gain knowledge about an approaching adversary and hinder him significantly from accessing hot nodes.

In order to do so several basic possibilities open up and should be discussed before more detailed options are focused on. Each adversary that enters a sensor network will be noticed by the nodes in a certain vicinity surrounding him. Those nodes therefore can propagate the knowledge about him to the other nodes in the network allowing for following strategies to be pursued:

- avoid data being displaced into the region surrounding the potential adversary,
- all hot nodes in the vicinity of the adversary displace their data until no data items are left in the endangered region,
- adjust vital traffic such that important messages are not routed through the vicinity of the adversary.

Naturally, all of the those strategies can improve the security of data in the network, however attention needs to be given to the possibility of a Denial of Service attack. Especially, in the last strategy which forbids for route traffic through the vicinity of an adversary can pose a threat to the correct functionality of the network. First, the detection of the sensors does not need to be perfect, i.e. an animal or other entity can be held for a possible adversary and having several of those in unfortunate locations in the network might lead to a partitioning of the network in terms of possible communication. Second, several adversaries can enter the network at the same time leading to the same problem of a possible partitioning. Luckily, the other strategies do not exhibit such straightforward disadvantages and can be thus be focused on in more detail.

When an adversary is detected to have entered the sensor network, a hot node in the vicinity of him can trigger the New Storage Node Choice algorithm in order to displace data away from the approaching adversary. Of course, this is only effective if an appropriate choice function is used: displacing data to a node closer to the adversary surely does not benefit the security of the network. Thus the employed choice function needs to base its choice on the location of the putative adversary. Fortunately, the GNodeFurthest choice function can be misused to achieve this: by changing the location of the generating node GNodeFurthest uses to guide its displacement choice to the location of the adversary, the choice function will choose a node that is located as far as possible from the location of the adversary. Such a behavior is precisely what is needed in order to keep data from the adversary's palms. An adversary that enters such a network will have little chances of capturing a hot node. Furthermore, the possibility to corner or surround a hot node, such that it cannot displace data away from the adversary is highly difficult, especially when only a single adversary is assumed. This notion is, of course, very similar to the one that forbids data to be displaced into the vicinity of an adversary, but calls for a different modification of New Storage Node Choice to disqualify nodes that detected an adversary from the evasion process. Here the same argument applies that surrounding a data item to capture the data is highly difficult.

Unfortunately, the interesting Sensor Aware Evasive Data Storage notion has to be tested for functionality in real world sensor networks and proof itself there to offer effective security enhancement. Furthermore, this notion does not provide any protection from malicious nodes that an adversary $\mathcal{A}_a(\star, active)$ might have employed in the network. Moreover, such an $\mathcal{A}_a$ can setup a net of malicious nodes somewhere in the network and force hot nodes through well directed motion to displace their data items in direction of the malicious net that waits to capture the precious data items. This is comparable to a fisher that sets up a fishing net in a lake and the tries to lure the fish to fall for his trap. Hence, a strong adversary $\mathcal{A}_a$ could use the Sensor Aware Evasive Data Storage notion to his advantage, though how effectively and costly such a setup would be for $\mathcal{A}_a$ remains to the investigated.

Having addressed the adversary exhibiting the strongest interaction skills, a look a quite weak adversary $\mathcal{A}_l(\star, \textit{limited passive})$ should be taken. Such an adversary cannot open up nodes in the network to retrieve data, but is assumed to carry them out of the network. Hence, a simple adaptation of the Sensor Aware Evasive Data

Storage idea can be used to address such adversaries successfully, even without a need for detection of $\mathcal{A}_l$ with motion sensors or similar. As each node is assumed to have knowledge about its vicinity, each node can detect when its vicinity is changing. Such an event is can either be caused by the topology changes all sensor networks are exposed to, or due to the fact that an $\mathcal{A}_l$ is carrying the node out of the sensor network. The former case, can be ignored assuming that topology changes are minor in the sense that only a fraction of the detectable vicinity nodes changes in a certain time period. An adversary carrying a node out of the network will cause for a different pattern in the change of the captured node's vicinity, thus will be distinguishable from a mere topology change. Assuming that a node successfully detected that is it being kidnaped, it can delete all data it possibly stored, which assumes that other nodes have redundancy copies. In such a case the adversary $\mathcal{A}_l$ would not find any useful data, although the node was hot at the time he picked it. Of course, there are a lot of possibilities and attacks against such a method, therefore a detailed investigation and thorough development of such a notion will not be given in this work, but will be left for future research.

5.5 Summary

This chapter introduced mostly advanced extensions to the Evasive Data Storage notion and thus functions to stress the richness the idea has to offer. Along with the extensions that are provided with no proof of efficiency or realistic implementation the Data Splitting notion is an exception. Data Splitting will be evaluated in more detail in the following Simulation Chapter and will be shown to significantly improve security when certain conditions are met. Furthermore, this chapter also discussed the possibilities that the combination of conventional data storage and Evasive Data Storage can offer, which was summarized as the Partial Evasive Data Storage notion. Hence, along defining the useful notion of Data Splitting this chapter also includes several interesting discussions of a possible combination of new and old data storage approaches as well as enriches the Evasive Data Storage notion with more bold extensions that still have to prove their usefulness in future investigations.

Chapter 6

Simulations and Comparisons

The last three chapters have focused on the presentation of algorithms of two classes, the first one were camouflage algorithms and the second were algorithms addressing the notion of Evasive Data Storage. Of those, a majority has been implemented in the scope of this work and thus can be used for different purposes. One main goal is to evaluate those approaches in a more realistic setting and in more detail then done before, especially from the perspective of the security properties mentioned in each of the chapters. Thus this chapter will show results obtained from running the implemented algorithms in a simulated environment resembling the properties assumed in the Data Security and Sensor Networks Chapter by usage of a discrete event simulator. This chapter addresses both Camouflage and Evasive Data Storage separately and their combined performance. First, the employed discrete event simulator is introduced and several details are given regarding the parameters that are determining how to run the simulations. Thereafter, the respective main topics, i.e. several of the algorithms found in the previous chapters are given along with figures providing insights into their behavior. In order to do so, for each algorithm, for a general technique or for a possible combination of algorithms, performance measures have to be introduced and explained. Those are necessary in order to obtain comparable data that indicates how effective the algorithms perform and most importantly, how well they satisfy the according security properties. Hence, before diving into simulating algorithms or techniques, those measures have to be introduced and explained.

6.1 Simulation Tools and Procedures

Before the results can be discusses, the used simulation environment and some of the used procedures or basic setup scenes should be investigated. This section therefore gives an introduction into the simulation tools that are used and how the model of the Data Security and Sensor Networks Chapter is implemented in the respective simulator. Furthermore, several explanations regarding the problems and issues that are essential are explained, which is also crucial to understanding the actual source code of the implementation of the Camouflage and Evasive Data Storage algorithms.

6.1.1 The Simulator

In order to simulate the behavior of the algorithms seen in the previous chapters a discrete event simulator is employed. Discrete Event Simulators offer the basic functionality to model entities that can exchange messages, which is basically what is needed for our model of a sensor network. The one used for the purposes of the implementation of Camouflage and Evasive Data Storage algorithms is SimJava 2.0 (see [23] for the website of the SimJava discrete event simulator). Of course, implementing the algorithms with such a simulative environment in mind has several consequences, for instance the programming language used might not be the best choice for embedded and resource constrained real systems available for sensor networks. However, due to the fact that the simulator has to cope with *all* sensor nodes in the network, which amounts to a quite large number, the implementation has to keep memory usage of each node low in order to allow for the simulation to execute. Hence, although the systems where the implementation code runs are substantially different regarding the available resources, both call for careful usage of memory. However, several issues regarding the simulator cause some deviation of the implementation given here to run on the simulator from one that would be written to run in a real network environment. Fortunately, those are just minor and can be easily changed in case the given implementation is to be used on a real network. Those differences also include structures that are included in the provided implementation for the collection of the results this chapter intends to attain, thus making the implementation code more complex than necessary. But this is unavoidable, when meaningful results are the primary goal of the implementation.

Although all those differences between the simulative implementation and a implementation on real sensor nodes exist, the presented simulative results can be used to study the intrinsic properties and can be used for obtaining insights into some of the behavior that any real life implementation would also exhibit. Furthermore, with the advances in technology that can be assumed to come, the simulative and a real implementation will converge more and more, in terms of what constructs and operations can be used. Summarizing, not all the restrictions posed by real life sensor networks have be addressed in the simulator, however the implementation at hand does provide sufficient resemblance to be used for evaluating the algorithms of the preceding chapters.

A final note should be given on the implementation. The used discrete event simulator, SimJava, uses the **Java** programming language and is used for most of the algorithms. However, due to the architecture of the simulator, where each sensor node is given its own *thread* to run, it becomes obvious that simulations impose a bound on the parameter n even on high end systems with large amounts of memory available. Hence, most of the simulations and the resulting figures are given for virtual networks of maximal size $n = 6400$. This, of course, is still a large and representable number, but it is clear that larger networks for real life applications are possible. In some cases, however, the need for each sensor node actually executing code in its own thread is not a necessity, thus allowing to bypass the bound on n and using another technique such that larger networks can be investigated. This is done in evaluating the properties of *Choice Functions* without a connection to a specific Evasive Data Storage algorithm. There, not the discrete event simulator is used, but a completely autonomous implementation is given, which is more suitable for running the choice functions and collecting respective data. The used language for this implementation is **C**$^\sharp$ and naturally the complete source code is provided along with the code for the Java implementation. That said, the parameters and the respective properties of the simulated algorithms can be explained in the subsequent sections.

6.1.2 Simulation Parameters

In order to carry out the algorithms in the discrete event simulator, several parameters regarding the setup of the simulator and hence the virtual sensor network as well have to be specified. The most basic parameter for such a network is the **number of nodes** or **size of the network**. Using the notation defined in the Data Security and Sensor Networks Chapter, n will refer to the number of nodes in the network. The simulations carried out in the following will have a n value ranging from several hundreds to several thousands, the exact value is specified separately for each simulation discussed. Along with the size of the network, the second most basic setup issue is the assumed **topology of the network**. This refers to the layout the nodes in the network are assumed to follow and the options for this parameter are as follows:

◦ *Rigid Grid*: This is the most basic layout that can be used to position the n many nodes in the network. Each node is positioned in a grid that is defined by the n and forms a square shape. For instance, a network of $n = 6400$ will result in a grid of dimensions 80×80, where each node has neighbors one unit apart to the left, right, top and bottom. Thus such a network exhibits constant density among all the nodes in the network, therefore all nodes except those at the border lines have the same number of nodes in their vicinity.

◦ *Random Rectangle*: This is similar to the Rigid Grid with respect to the shape: the network is assumed to have a rectangular border defined by some parameters for *width* and *height*. However, within those boundaries the n nodes are assigned random locations, such that no particular pattern should be preferred. Such a setup is more likely to be encountered in real life scenarios, as rigorously setting up several thousand of nodes according to some pattern is highly laborious.

- *Rigid Circular*: Equivalently to the Rigid Grid, a Rigid Circular layout can be thought of equally spaced nodes following a rigid pattern. Here, the nodes are not layout according to a grid, which can be seen as a layout in an euclidian coordinate plane, but rather in on concentric circles, which corresponds to a polar coordinate system. The idea behind this slightly different layout is to provide a form that corresponds to a network as seen from the perspective of a single node that is at the center of all the concentric circles ordering this rigid structure. Given that n many nodes are to form a Rigid Circular sensor network layout, the process fills up the concentric circles until n nodes where inserted, where the layout is influenced by parameters defining the stepwise increase of r and φ, which are used to describe points in polar coordinates.[1]

- *Random Circular*: Just like there is a random counterpart to the rigid rectangle, there is the analogous notion of a Random Circular layout. In order to drop the uniform density, which is naturally an inherent property of the Rigid Circular layout, the spots of the imagined polar coordinate system and filled in randomly until the desired n many nodes have been assigned some location. A simple procedure that chooses r and φ randomly and assigns the corresponding euclidian coordinates to the current node can be used to realize this layout. Of course, some upper limit r_{max} for the variable r has to exist in order to keep the circular shape recognizable.

The usage of the *Random* variants can be more compelling in terms of real life applications, but also poses some difficulties in form of gaps, partition of the network or similar. The assumption for this thesis was to disallow for a partitioned sensor network, as it is not of concern to deal with such things at the level of devising security related algorithms. The problem of gaps has been already mentioned in the scope of conventional storage methods, where it was referred to as the *Void Problem*. However, for the evaluation of the security properties of the implementations of the Camouflage and Evasive Data Storage notions those problems are less interesting and thus the prevalent topology that will be used (mostly to keep the evaluation simple) is *Rigid Grid*. Further parameters that can be considered concern the properties of sensor nodes communication primitives, where **transmission delays**, **unreliability probabilities** and **vicinity range** play the major role. Those will be given explicitly when they are needed, but will be omitted when they do not contributed significantly to the respective properties investigated. Furthermore, when unreliability probabilities are introduced into the simulation environment, they naturally will be given in form of the probabilities p_{lb}, p_{pp} and p_{gb}. At this point, there is a choice between two possible ways to handle those probabilities: either the same approach is taken as in the Data Security and Sensor Networks Chapter, where it was argued that in real sensor networks the communication primitives **point-to-point** and **global broadcast** need to implemented using the **local broadcast** primitive, thus making the respective probabilities p_{pp} and p_{gb} depend on p_{lb}, or a more independent approach is taken and each value for the p_{lb}, p_{pp} and p_{gb} is predefined without any actual connection between the other probabilities. However, as the issue regarding unreliabilities has been discussed in the correctness sections of the respective algorithms, the simulations here will not ascribe any significant attention to those probabilities. Similar to the unreliability of the communication primitive, the fact that sensor networks are subject to failures of its nodes can be included into the simulation as a parameter specifying how **failures** occur (see Data Security and Sensor Networks Chapter), how frequently those occur and how they are distributed. The simulation can implement these by a simple modification of the behavior that the failed node's thread exhibits, which can include a complete cut of participation in the simulation or only a temporal indifference to the messages send to it. Clearly, this depends on the wanted failure type. The malicious failure type, however, that includes the presence of an adversary $\mathcal{A}$ that matches one of the models defined in this thesis should be introduced not as a elementary parameter of the simulation environment, but be tailored to the algorithm that is evaluated at hand, as the actions of $\mathcal{A}$ ought to depend on the task the algorithm at hand aims to solve. A different type of failure that can also be included is that of **dislocation**. It has been argued during the introduction of sensor networks, that its nodes can be assumed to be moved due to changes in the environment,

[1]Of course, each node obtains ordinary euclidian coordinates, which can be easily obtained from the r and φ values of the polar specification.

which of course can have a substantial influence on the network and on the mechanisms used therein. The implementation can achieve this kind of failure by a exchanging the links a node has stored to represent its vicinity to a different set and in the same sense update the links of the node's neighbors as well.

Besides the elementary parameters that can be thought to be inherent to sensor networks or to the simulation environment respectively, the more important ones are the parameters that influence the algorithms themselves. Those are mostly algorithm specific and will be specified at the time the respective figures and results are discussed. However, the influence those algorithm specific parameters have on the behavior will not be repeated and should be looked up in the according chapters, if any doubt arises. Over the course of this chapter, the used choice function will emerge as one of the most important parameters, not only for the Evasive Data Storage notions, but also for the Camouflage algorithms.

6.1.3 Simulation Performance Measures

Now that the parameters for the simulations are explained, the other important parts that will be used extensively in this chapter are the *Performance Measures*. The chapters that introduced the various Camouflage and Evasive Data Storage algorithms provided security properties, that intend to capture what the actual objective of the class of algorithms actually is in terms of the increase in security they want to provide. Thus in order to evaluate how effectively the algorithms simulated and analyzed achieve those security properties, performance measures have to be defined that indicate the successful performance of the algorithms. The main motivation behind those is that the security properties are more of a qualitative statement that does not indicate directly how good behavior can be measured, therefore calling for the notions presented in this section. However, this section does not solely present the measures needed for the security properties, but also gives a survey over a broader range of such measures that will not necessarily be used in the sections to follow. Note that some sections define evaluation principles of their own, hence not using the ones given here.

The most basic performance measure that can be used to evaluate an algorithm or to compare different approaches that achieve the same goal is to measure the *number of messages* needed by the respective procedure. This measure is especially important in the case of sensor networks, as it indicates directly the influence an algorithm can have on the overall consumption of energy in the network and thus on the overall lifetime.

The next measure, which will be of substantial importance in the remaining part of the chapter is the notion of *entropy*. Entropy is a measure known from information theory and is used to describe the randomness of systems. Such a measure can be assumed to be linked directly to the Camouflage Security Property, as the more randomness a system has, the less probable it is that an adversary $\mathcal{A}$ can extract any useful information out of any traffic analysis he might be able to perform. This is rather counterintuitive when the general usage of the entropy concept is known, where higher entropy indicates that more information can be obtained. However, in scope of this work and in sensor networks in general, a higher entropy means that each nodes participates in a larger number of communications, which hence causes the recognition of patterns in the network through traffic analysis to be more complicated. In other words, entropy is inversely proportional to the degree $\mathcal{A}$ can restrain from being shifted towards a blind level of traffic analysis. Thus the concept of entropy will become especially important in the context of the investigated Camouflage techniques. The actual measure of entropy E can be defined for a subset $A \subseteq \Pi$ of the sensor nodes found in the network:

$$E(A) = -\sum_{a \in A} \frac{M_a}{M_A} \log_2(\frac{M_a}{M_A})$$

where M_a is the number of message the sensor node a forwarded and M_A is the total number of messages send in the region specified by A. [2] It is clear that to measure the entropy of the complete network $E(\Pi)$ has to be calculated.

[2]The negative sign is needed to get a positive entropy measure, as the values given to the logarithm in the definition of E are smaller than 1, thus yielding negative values by the logarithm.

Similar to the entropy measure the *distance measure* is one of the more complex performance measures used in the following. Its usage covers both Camouflage as well as Evasive Data Storage and intends to indicate how far part nodes that satisfy certain properties are in the network after some finite amount of time. Hence, the used property will vary for Camouflage and Evasive Data Storage respectively, but otherwise the intention of the measure is similar for both cases. As the measure indicates how far apart nodes are in the network, the fundamental building block is the distance two nodes a and a' share, which is calculated as follows:

$$d(a,a') := \sqrt{\left(LOC(a).x - LOC(a').x\right)^2 + \left(LOC(a).y - LOC(a').y\right)^2}.$$

In order to specify the complete measure the special properties mentioned above have to be included. Starting with the Camouflage case, the properties aim to distinguish nodes that forwarded fake messages and those nodes that did not participate in faking communication at all. Thus at a certain point in time there exists a set $F := \{a_1, \ldots, a_l\} \subseteq \Pi$, such that each $a_i \in F$ participated in some form of faking communication. Furthermore, as in most cases the fake messages are connected to some genuine communication, which is anticipated to be camouflaged. Hence, there also exist nodes that participated at least once in that kind of communication process and shall be denoted as $C := \{a'_1, \ldots, a'_k\} \subseteq \Pi$ Then the overall distance measure used for Camouflage is defined as follows:

$$D_{cam} := \frac{\sum_{a \in F} \min\{d(a,a') : a' \in C\}}{l}.$$

Looking at the definition of D_{cam} it becomes clear that the measure averages over the distances nodes involved in fake messages have to a certain node involved with the original communication process, therefore indicating how far apart those nodes are. The constrain that only the node in set C having the minimal distance is considered for each node in the set F, is grounded in the goal to avoid instability in the measure, which can occur when all distances to members of C are considered. Naturally, the sets C and F can be varied depending on the evaluated Camouflage. For instance, in the case of the Fake Packets Camouflage notion, the D_{cam} measure, as just defined, cannot be applied without modification. However in the case of Fractal Propagation the definition of D_{cam} is quite natural and as the D_{cam} will be used solely for this algorithm, there is no need to provide any other forms of the measure.

Continuing to define the measure for the Evasive Data Storage case, the property used here is to distinguish hot nodes from ones that do not store any data at a given point in time. Thus the set used is $H := \{\tilde{a}_1, \ldots, \tilde{a}_h \subseteq \Pi$, which contains all hot nodes in the sensor networks. A second set as needed for the Camouflage distance measure is not needed and the distance measure is given by:

$$D_{hot} := \frac{\sum_{i=1}^{h} \sum_{j=1}^{h} d(\tilde{a}_i, \tilde{a}_j)}{h^2}.$$

The distance measure for the Evasive Data Storage calculates the sum of the distances that separate hot nodes in the network, therefore indicates how spread the hot nodes are in the network.

Now that the measure and its calculation has been clarified for both cases, what remains is to shortly describe its qualitative purpose: In the case of Camouflage a high D_{cam} measure stands for a larger coverage of the network through fake messages, which is certainly preferable from a low measure which indicates that fake message do not successfully divert attention from what is needed to be undisclosed. This property can vary in degree of usefulness for different Camouflage techniques, however it finds a successful applicability in the techniques that will be evaluated shortly. In case of Evasive Data Storage a small value for D_{hot} means that data items are located rather close to each other, whereas on the other hand a large value for D_{hot} stands for a better distribution of the data items. Thus a large D_{hot} value indicates that a better better security can be guaranteed, because of the mundane fact that an adversary $\mathcal{A}$ will have to overcome larger distances starting at one hot node before reaching another hot one.

As seen in most of the security properties that were given for Camouflage and the Evasive Data Storage notion, the probability for an adversary $\mathcal{A}$ to achieve a certain malicious intend plays a crucial role in all of them. Thus *probabilities* or *probability distributions* of different kinds, for instance success probabilities of $\mathcal{A}$ for finding a hot node, have to be considered as performance measures as well. However, the precise specification of what probabilities are important will be given in the respective context, as they can vary largely.

Those measures above form the means that will be used in the following to evaluate the algorithms. However in some circumstances a section can introduce different measures or combine measures of this section to indicate performance of an algorithm. That said, the actual investigation of the algorithms can be approached in the remaining part of this chapter.

6.2 Performance of Camouflage Algorithms

In order to start with the evaluation of the algorithms of this thesis, the Camouflage notion will be addressed first in scope of this section. One reason for this is to adhere to the sequence of presentation seen in the chapters of this thesis, but more importantly is that some camouflage algorithms will be referred to in the evaluation of Evasive Data Storage. The Camouflage notion anticipates to hinder an adversary $\mathcal{A}$ with traffic-analysis skills to obtain information, which is what the corresponding Camouflage Security Property states in more formal terms. The goal is to reduce a local or global adversary to the traffic-analysis skill level denoted as blind, i.e. providing means such that $\mathcal{A}$ is forced to behave as if he disposed over a blind level of traffic-analysis skills. In order to evaluate the effectiveness of the Camouflage algorithms, a measure has to be used that indicates how well $\mathcal{A}$ can analyze the network. Hence, the prevailing measure for this section will be the entropy E, where a higher value for E indicates that $\mathcal{A}$ is forced more to behave at a blind level.

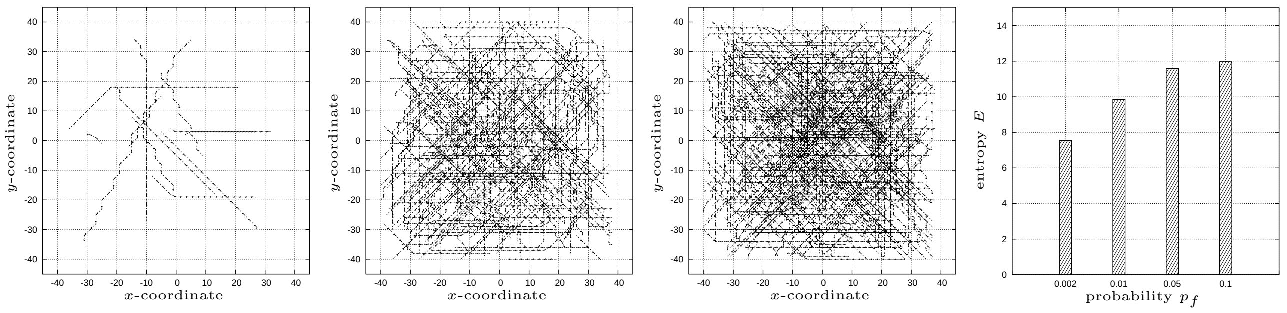
6.2.1 Fake Packets Global Camouflage

The Fake Packets notion stands for a simple approach to Camouflage and performs in a global context, basically implying that several sensor nodes that can be quite far apart in the network are involved in the technique.[3] To measure the effectiveness of the Fake Packets notion, the increase in the network's entropy shall be investigated for varying parameter p_f. For the sake of simplicity, the implementation of Fake Packets that is used for the simulations does not include a strict adherence to the size of $fake\_msg$ messages, as the actual algorithm 3.2 proposed. This is unnecessary for the calculation of the entropy and was therefore not included. For a actual implementation this should naturally be included though. The Fake Packets algorithm given in algorithm 3.2 does build upon the assumed routing scheme, which naturally can also take the form of Multipath Routing or Random Walk routing, but for the evaluation here a conventional routing scheme is assumed, for instance shortest path. Results yielded from using other routing mechanisms should take a similar form comparable to the ones the routing mechanism used here provides. Besides the p_f parameter, the Fake Packets notion is controlled by another crucial value, namely *time_int*. This parameter, however, is not considered for the simulation, as sufficient information about the behavior of the Fake Packets algorithm can be extracted from a single invocation, or (put in other terms) a single passage of the *time_int* time interval.

Now the basic simulation parameters and the results can be discussed. As virtual network a rigid grid is assumed with 6400 sensor nodes and hence having the size 80×80. Nodes in the network are assumed to have an approximate vicinity radius of 3, which yields that on average a node can communicate with about 28 neighbors. As already mentioned only a single valuation is looked at, which is basically the fake message traffic incurred after each time interval *time_int*. Figure 6.1 shows the traffic patterns incurred for probabilities $p_f = 0.002, 0.01$ and 0.05. The respective pictures show the network and fake messages sent, indicating that even very low values for p_f already yield a substantial message overhead.

Furthermore, nodes at the border of the network are less likely to participate in the fake communication. What is more interesting is the increase in entropy that is shown in the right diagram of figure 6.1. The obvious result is affirmed, which is that higher values for p_f yield a higher overall entropy in the network. Hence, it can be argued that employing the Fake Packets algorithm with a rather high value for p_f, say larger

[3]For more detail on the functioning of the Fake Packets algorithm refer back to the Camouflage Chapter.



$p_f = 0.002$ $p_f = 0.01$ $p_f = 0.05$ Entropy Values

Figure 6.1: Fake Packets for varying parameter p_f and Fake Packets Entropy

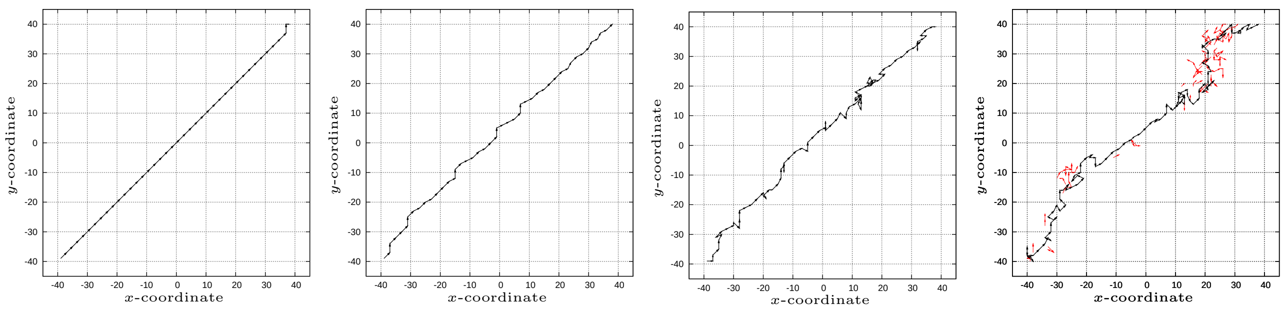
than 0.01, can successfully contributed to an adversary's failure to recognize traffic patterns, if the respective *time_int* is chosen to invoke the algorithm sufficiently frequently. However, high values for p_f cause almost the complete network to send fake messages, as seen in the respective figure, which also entails a very fast drainage of the energy available. Using low values for p_f surely is a wiser approach to the Fake Packets notion, as the network is burdened less, but will also not be as successful in the camouflage of traffic-patterns as desired. The reasons have already been discussed in the Camouflage Chapter and are grounded in the fact that Fake Packets is not connected to any communication primitive, but generates fake messages randomly. Summarizing, the Fake Packets notion is a very basic idea to provide Camouflage and can provide high entropy, but due to its indifference to actual, real communication Fake Packets needs large number of messages to achieve sufficient entropy and fulfill the Camouflage Security Property respectively.

6.2.2 Fractal Propagation Global Camouflage

Having seen the problem with the Fake Packets notion, this section explores the Fractal Propagation notion to perform Global Camouflage or, more specifically, to disguise communication between two particular nodes in the network. This notion can be seen as addressing the fundamental deficiency of Fake Packets, where the reference to actual communication of the network was missing. As the Fractal Propagation notion is achieved through stepwise refinement of more basic notions, it is clear that to start off a look needs to be taken at the behavior of those intermediate refinements. Thus the investigation begins with simulative results of Conventional Routing, Multipath Routing and Random Walk. Furthermore, it is important to keep in mind that the used simulation model for the subsequent simulations is a *Rigid Grid*, which implies a uniform density between nodes in the simulation network. This is also reflected in the resulting figures of the routing paths and naturally would take slightly different forms, if the uniform distribution would be dropped, for instance by employing a *Random Grid* or similar topology. However, for the purpose of a concise evaluation of the refinement steps and the final notion of Fractal Propagation the uniform distribution of nodes provided by a rigid grid is most suitable.

Before, the simulation results are presented and discussed in more detail the used performance measures have to be looked at. As with Fake Packets and the remaining simulations where Camouflage plays a vital role, the entropy measure E is of utmost importance, but also another measure that is obtained by combining two of the elementary ones presented at the beginning of this chapter is employed. In order, to take the spreading of fake messages in the network into account the D_{cam} distance measure is used and a combination measure obtained by forming the product $E(\Pi) \cdot D_{cam}$ is employed, which indicates how well an algorithm increases the entropy in the network and how well distributed the fake messages involved are. Those performance measures are important and will be used for the evaluation of the Fractal Propagation in this chapter, but another important performance measure has been introduced in scope of the original work [1] that provided the Fractal Propagation notion, is denoted as GSAT. The GSAT performance measure has the goal to evaluate how effectively an adversary $\mathcal{A}$ can identify the destination node of a communication established between two nodes. The idea is to let the adversary search (using his traffic-analysis skills) through the network until he reaches the destination. Thus a large search number returned by the GSAT measure indicates that an adversary is successfully hindered to track communication easily. However, the precise results obtained will not be repeated in this chapter, but can be found in [1] along with more details on the GSAT and its origins.

Hence, for the Fractal Propagation investigation the basic performance measures and their combination will be sufficient to understand the subsequent simulation results. The basic goal is the use entropy and related measures to investigate the usefulness of the camouflage algorithms in fulfilling the **Camouflage Security Property** stated in the Camouflage Chapter. Thus the basic setup for the investigation of the Fractal Propagation Global Camouflage will consist of a source node located in the down left corner of the virtual sensor network, which sends one or several messages to a destination node in the opposing corner. This can be seen as a quite simplified scenario, where protection is needed to hide the destination from an adversary $\mathcal{A} \succeq \mathcal{A}(local, \star)$, who observes the source node when it is sending data somewhere.



Conventional Routing Multipath Routing Random Walk Fractal Propagation

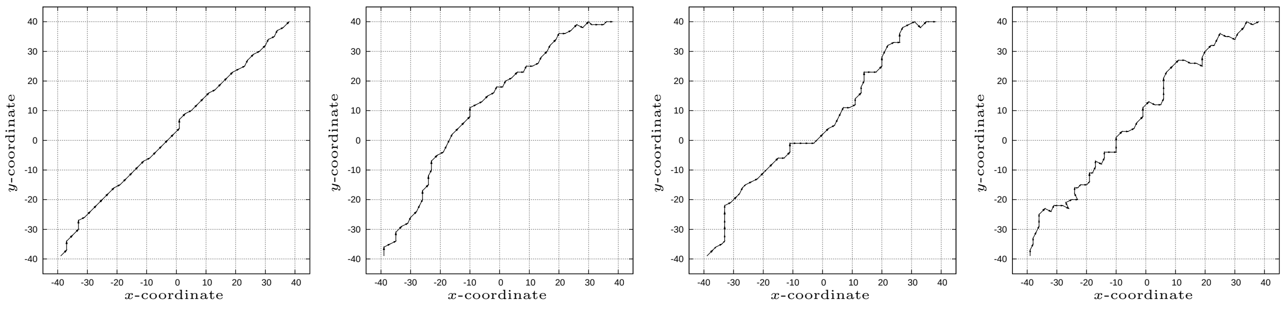
Figure 6.2: Stepwise Camouflage Refinements for Fractal Propagation

Before, the Camouflage Security Property is actually investigated the properties of the refinements and the influence the parameters have on those has to be looked at. This is not directly connected to the security issues, but is crucial in obtaining a thorough understanding of the algorithms up to the Fractal Propagation algorithm. Hence, the basic communication behavior of each of the refinement is looked at. Setting up a network with 6400 nodes, each node having a vicinity radius of 3, the figure 6.2 shows how the refinements, starting with conventional routing, lead to the Fractal Propagation notion and the behavior it yields. The left picture shows a path a single message takes when routed using a conventional shortest path routing algorithm. This is also what an adversary $\mathcal{A}$ with global communication skills would see,[4] where it is obviously that $\mathcal{A}$ will have no difficulties identifying the destination node in the upper right corner. The first refinement that is provided by the Multipath Routing notion is shown in the second diagram for a $p = 2$ parameter value. A slight deviation from the shortest path of the conventional routing is observable, but is still insufficient to be of any advantage in terms of hindering an adversary from following the message to its destination. The Random Walk refinement step however, offers the first major deviations from the shortest path, by allowing to also move backwards in the opposite direction with a certain probability. In the figure, the Random Walk communication path is shown with $p_r = 0.5$ parameter, which can yield a path that is up to two times as long as the shortest one. The right diagram of figure 6.2 reveals the final Fractal Propagation algorithm. The execution shown in the rightmost diagram is yielded by parameters $p_c = 0.01$, $p_f = 0.7$, $p_t = 0.01$ and $K = 5$. Thus generating fake branches with quite moderate probabilities, but forwarding them with high probability of 0.7 up to a depth of 5. In the diagram the fake packets that are sent are colored red, in order to distinguish them from the genuine communication that is to be camouflaged and is given a black color. In the subsequent figures where Fractal Propagation plays a role, the fake packets this algorithm generates will always be given a red color such that a

[4]Note that the same applies to a local adversary, when the communication takes place in his observable vicinity.

distinction can be made.

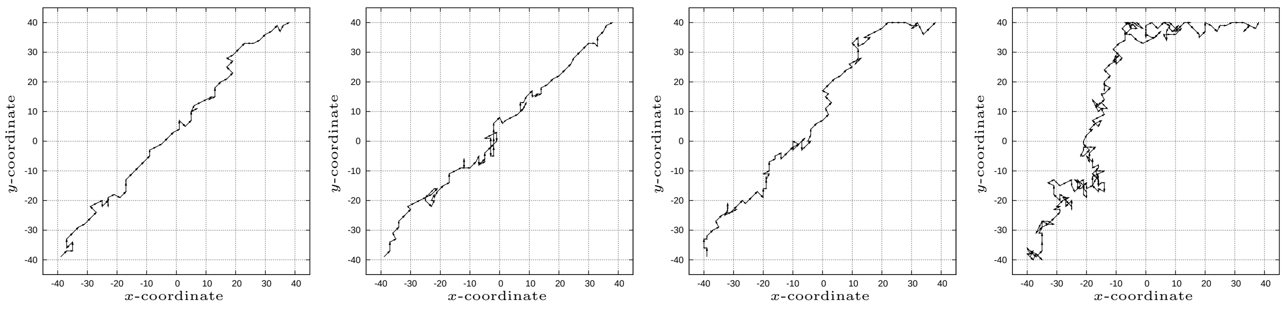
After having seen how the refinement steps yield the Fractal Propagation notion in simulated executions a detailed look ought to be taken at each refinement for varying parameters. The subsequent figures were yielded in the same virtual network setup as used in the the previous simulation and therefore will not be repeated.



Multipath Routing: $p = 2$ Multipath Routing: $p = 4$ Multipath Routing: $p = 6$ Multipath Routing: $p = 8$

Figure 6.3: Multipath Routing Routing for varying parameter p (1 message)

Starting with the Multipath Routing notion, figure 6.3 shows executions with varying parameter p. The figures are confirming the expected results, in that a large p parameter allows for a broader deviation from the shortest path, but it is important to stress the crucial difference to the next refinement, the Random Walk notion: the Multipath Routing routing notion always forwards the message towards the destination and does not allow for backward jumps in the path. Of course, this can be overridden by making the p ridiculously close to the number of nodes in the vicinity, but that will also cause the complete routing process to fail. The figures shown in 6.3 have to be regarded as sample executions, as repeating the sending of a message will yield different figures due to the used random choice among the p closest nodes. Still the figures of a single transmission give sufficiently intuition as to how strong p can cause the communication path to vary. To sum up, an increasing p parameter causes Multipath Routing to approach the Random Walk notion, but also causes routing to become more and more inefficient. Hence, the best results will be obtained with a low value for p and letting the deviation be handled by the Random Walk notion and its respective parameters.



$p_r = 0.8$ $p_r = 0.6$ $p_r = 0.4$ $p_r = 0.2$

Figure 6.4: Random Walk for varying parameter p_r and $p = 2$ (1 message)

In order to investigate the Random Walk algorithm, all the simulations are carried out once with a multipath parameter of $p = 2$ and once with $p = 6$. The first case is handled in figure 6.4 for varying parameter p_r, which influences the allowed deviation from the path suggested by Multipath Routing. As the figures show, high values for p_r cause to Random Walk approach in its exhibited behavior the Multipath Routing scheme. Letting, the probability p_r get lower the communication path can begin to deviate significantly from the shortest path and most crucially cause many seesaw changes on the way. Which can be ascribed to the event that a forwarding to a node happens in the opposite direction of the destination node with probability $1 - p_r$ and is directly followed in the next step by a forwarding in the right direction with a probability of p_r. This process can iterate. Such a seesawing of the forwarding process can be nicely observed in the rightmost diagram of figure **??**, where the p_r is sufficiently low. Now, the question that still remains is how the behavior changes when the p parameter is also chosen to be high.

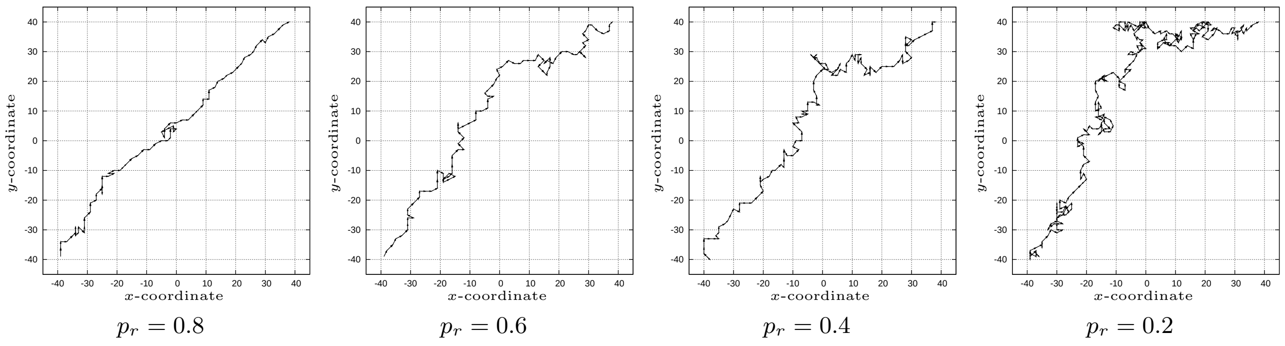
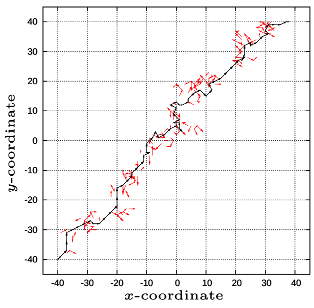


Figure 6.5: Random Walk for varying parameter p_r and $p = 6$ (1 message)

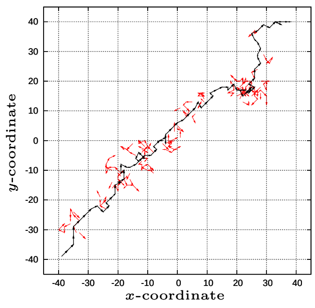
Figure 6.5 shows the executions of the Random Walk algorithm for varying p_r, but with a larger p value of 6. The figures support the statement made previously that a higher p yields behavior similar to that of a lower p_r. This can be seen in the diagrams of figure 6.5, where compared to the figures yielded by the same p_r values in the $p = 2$ case just shown, the higher p seems to shift the p_r probability towards lower values. In addition comparison of the figures indicates, that the lower the p_r itself becomes, the less relevant the used p value is. This should convey sufficient information about the relationship between the Multipath Routing and the Random Walk notions and their parameters respectively. What remains is a comment on the influence Random Walk has on an adversary that wants to follow the communication path. As seen already, Multipath Routing does not pose any problem to an adversary, Random Walk is similar in that respect, as the adversary can still easily follow the traffic-pattern yielded, but will take longer time as the traffic-pattern contains many seesaw parts on the way. These parts of the path can be effective in case other traffic occurs in the network, which can cause an adversary to loose track of the actual communication path. Thus Random Walk does improve security, even if it is only under certain circumstances.

The final stage of the refinement and the notion that can be used to fulfill the Camouflage Security Property, is the Fractal Propagation algorithm, which will be investigated just as the previous two refinements for varying parameters and a similar setup to the one used so far.

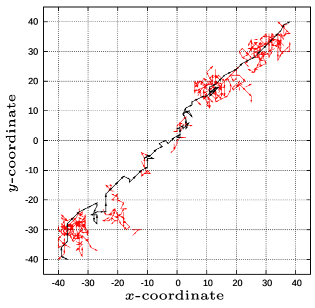
Figure 6.6 shows executions of the Fractal Propagation algorithm with underlying parameters $p = 2$ and $p_r = 0.5$. The virtual network used for the simulations consists of 6400 sensor nodes, each with a vicinity radius of 4. The resulting figures show how vital the right choice of the parameters is and that even rather low probabilities for the generation of fake branches can yield substantial overhead of fake messages. This is exemplified in the rightmost diagram of the figure, where low probabilities $p_c = 0.05$ and $p_t = 0.02$ yield quite a dense band of fake traffic around the communication path. This is due to the high forwarding probability of 0.9, as can be seen when compared to the cases where this probability is lower (the two left diagrams of figure 6.6). The figures indicate that choosing the right parameters is a crucial task and is mostly dependable on the properties of the network and the traffic the application of the network exhibits. However, the pictures shown in figure 6.6 reveal a strong deficiency of the Fractal Propagation algorithm as it is presented in [1]. The fake traffic, which was supposed for lure an adversary away from the actual communication path, remains close to the path even for large depths K. This observation leads to a natural generalization of the original Fractal Propagation notion, which then allows for a solution to the problem just described.



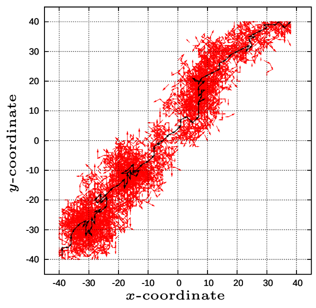
$p_c = 0.05,\ p_f = 0.3,$
$p_t = 0.01,\ K = 8$



$p_c = 0.02,\ p_f = 0.7,$
$p_t = 0.01,\ K = 5$

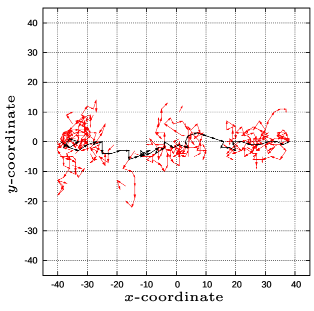


$p_c = 0.01,\ p_f = 0.9,$
$p_t = 0.01,\ K = 8$

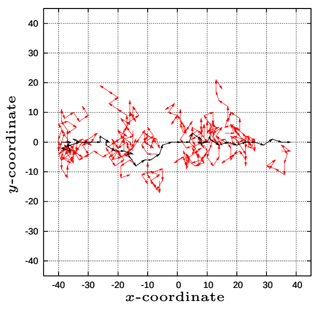


$p_c = 0.05,\ p_f = 0.9,$
$p_t = 0.02,\ K = 8$

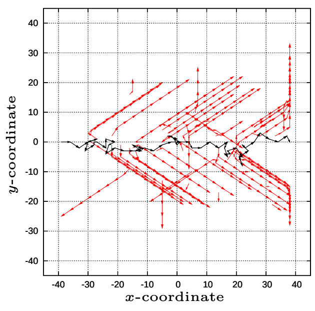
Figure 6.6: Fractal Propagation for varying parameters (1 message)



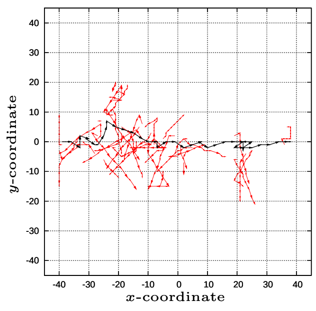
UVicinity



UFurthest$k$ with $k = \frac{|V|}{2}$



GNodeFurthest with $k = 1$



DirectionV with $p_c = 0.05$

Figure 6.7: Fractal Propagation with different Choice Functions

The generalization of Fractal Propagation is achieved by introducing the concept of the *choice function* presented in scope of the Evasive Data Storage notion to the Fractal Propagation idea. The Fractal Propagation algorithm presented in the Camouflage Chapter, algorithm 3.5 allows an arbitrary node to be chosen to forward a fake message to. This however is the main cause for the adherence of the fake messages to the original communication path. Thus a natural generalization is to introduce a choice function to guide the forwarding mechanism, just as a choice function guides how data is evaded in the world of Evasive Data Storage. Figure 6.7 shows the Fractal Propagation generalization for the four most basic choice functions that were presented in the Data Oriented Techniques Chapter. For a detailed presentation of the generalized Fractal Propagation refer to the Appendix, where also the modified pseudo-code is provided. The resulting figures for the generalized Fractal Propagation notion shown in figure 6.7 indicate that the work done in scope of Evasive Data Storage leads the way to a significant improvement of the Fractal Propagation method that originally was proposed in [1]. The original description can be seen to use as forwarding function the choice function UVicinity, thus yielding peeks of communication close to the spots where the probability p_c caused the initial fake branch to emerge. Hence, with the knowledge gained in the Evasive Data Storage chapter on how to evade data using different kinds of choice functions, the current approach of Fractal Propagation can be seen as a special case of a general class of Fractal Propagation propagation algorithms that are parameterized by choice functions, which guide the development of the fake branches. A thorough understanding of this family of Fractal Propagation algorithms must wait until the basic properties of the different kinds of basic choice functions and combinations of those will be discussed later on. However, it should be clear that the Fractal Propagation algorithm 3.5 can easily be modified to adhere to a choice function specified by some parameter and that different behavior will be yielded, not regarding the overall entropy values or number of messages, but the locations of fake messages become more controllable. This is seen best for the case of the GNodeFurthest choice function, depicted in the third diagram from the left in figure 6.7. Compared to the original Fractal Propagation algorithm's fake message location, the GNodeFurthest choice function forces that the branches do not concentrate around a single spot but penetrate the network and employ the K parameter nicely.

At this point a fairly detailed description has been given regarding the various parameters and the resulting figures or patterns that result in the network. This is regarded as quite important as it conveys a feeling as to what to expect from the algorithms and which parameters are important for constructing a valid effort to counter an adversary $\mathcal{A}$ with traffic analysis capabilities. Nonetheless, in order to complete the picture, the actual effect on $\mathcal{A}$ has to be investigated. Thus the remaining part of this section will be concerned with showing the effectiveness of the camouflage provided by the Fractal Propagation notion in presence of $\mathcal{A}$ by evaluating the respective Camouflage Security Property using the measures described at the beginning of this section.

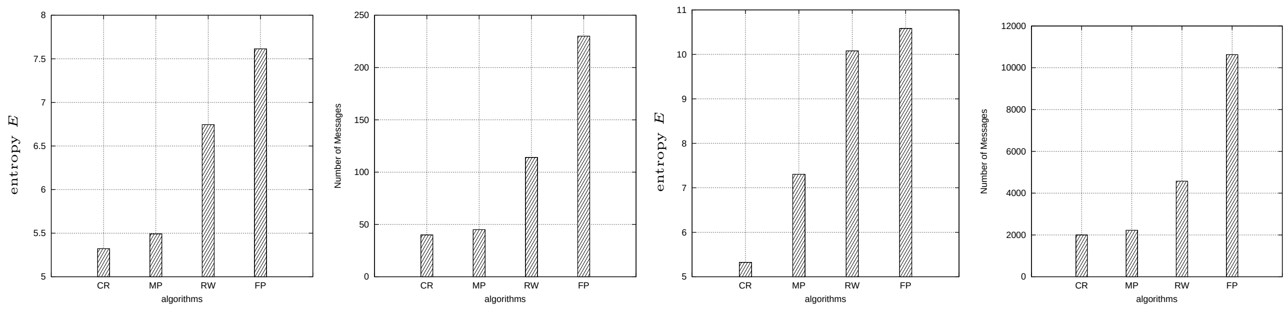
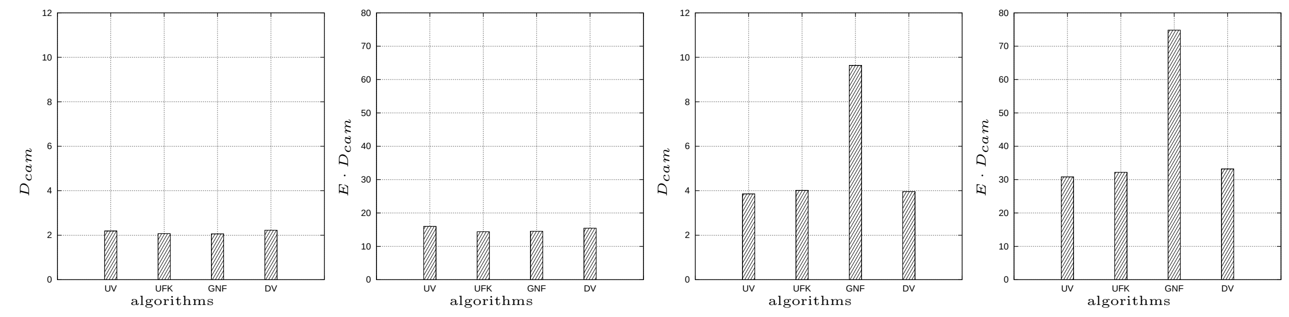


Figure 6.8: Fractal Propagation Refinements' Entropy and Number of Total Messages

Figure 6.8 shows both values for entropy and the number of messages needed by the approach, where the parameters for the algorithms used are equal to those employed in figure 6.2. The two left diagrams refer to the entropy and the number of messages or more precisely local broadcasts needed for a single message to reach its destination, where the setup of source and destination nodes is the same as the one used so far for the other simulations and only a single message is sent. The yielded results show that Fractal Propagation provides the highest entropy of all the refinement steps presented in this section. However, the number of messages is naturally also adequately increased. To refer to the Camouflage Security Property, it is clear that the high entropy E yielded by the Fractal Propagation algorithm does successfully contribute to shifting an adversary $\mathcal{A}$ with local or global analysis skills towards a blind level. The remaining two diagrams on the right of figure 6.8 show the entropy after 50 messages have been sent from source to destination. The observation is that the Random Walk algorithm approaches the Fractal Propagation in terms of entropy, however not quite reaching that same level. The main point of orientation for satisfaction of the Camouflage Security Property remains the result obtained for a single communication step, which clearly shows that Fractal Propagation does increase security for communication. The results the simulations yielded, are almost identical to the ones found in [1], hence can be used to argue that the generalization of the Fractal Propagation notion does indeed lead to even better results. The other performance measure, namely GSAT, which was introduced in [1], can also be used in context of the Camouflage Security Property, however the corresponding results will not be shown here. But they also indicate the superiority and effectiveness of the Fractal Propagation algorithm.

Now that Fractal Propagation in its original form has been investigated, what still needs to be evaluated are the properties of the generalization of the Fractal Propagation notion, which was described previously. To evaluate the advantages the generalization offers the respective simulations are carried out in a network of size 80×80 and assumed vicinity radius of each of the 6400 sensor nodes is 4. For the underlying Multipath Routing and Random Walk algorithms the respective parameters are $p = 2$ and $p_r = 0.5$. Hence, the same setup used as for the simulations presented in figure 6.7. The performance measures employed are D_{cam} as well as $E(\Pi) \cdot D_{cam}$. Both put an emphasis on the spreading of the fake packets, and therefore stress the advantage that the generalization of the Fractal Propagation allows for: controlling the distribution of fake messages and avoiding a conglomeration that is observed in the original Fractal Propagation or Fractal Propagation with choice function UVicinity respectively. The resulting values for the performance measures are given for varying depth parameter K, whose values are given in the respective figure 6.9.



$K = 2$ $K = 2$ $K = 8$ $K = 8$

Figure 6.9: Generalized Fractal Propagation evaluation using the Entropy and Distance Measure

The figure shows the values for the measures D_{cam} and $E(\Pi) \cdot D_{cam}$ obtained for generalized Fractal Propagation for each of the basic choice functions. The basic Fractal Propagation parameters used are $p_c = 0.01$, $p_f = 1.0$, $p_t = 0.002$ and $K = 8$, which obviously are chosen such that branches can develop to their full length stressing the problem solved by the generalization. The results show that for a low depth value the measures cannot determine any significant difference between the choice functions (see the two left diagrams in the figure). However, as soon as the depth is increased one of the choice functions reveals its superiority compared to the others. The GNodeFurthest does avoid conglomeration of the fake messages in the proximity of the original communication, thus providing a better deflection of an adversary's attention. Of course, more sophisticated choice functions can be employed for even better results, but the ones investigated here already reveal that the generalization does provide means to more effective algorithms satisfying the Camouflage Security Property. To get a deeper grasp of the properties that the choice function enhancement does offer, the choice functions themselves have to be evaluated thoroughly. This will be done in scope of the Evasive Data Storage section, but can be thought of being equally applicable to the generalization of Fractal Propagation.

6.2.3 Firework Local Camouflage

The previous two sections covered the Global Camouflage algorithms and therefore leave only the investigation of the second notion presented in this thesis concerning Camouflage: Local Camouflage or Local Anti Traffic Analysis Techniques. The Camouflage Chapter introduced two Local Camouflage notions, that will be investigated in this section: Repeated Firework and Wandering Repeated Firework. As with the Fake Packets notion, the implementation of the Repeated Firework and Wandering Repeated Firework notions does not strictly abide by the size of $fake\_msg$ messages, i.e. the parameter $reply\_len$ is assumed to be constant and of minute value. This can be done as there is no need for comparing different sizes of messages in the scope of the evaluation carried out within this section. However, it is clear that in real applications the presence of such a concept in the algorithm itself can be important in successful camouflage, especially when an adversary can compare the size of messages sent in the network.

As with the Fractal Propagation notion, both Firework Algorithms first are simulated for varying parameters in order to provide sufficient knowledge about the behavior of them and the influence of the respective parameters. Before, looking at the basic parameters and the simulation figures, it is important to note that the figures do not show all communication, but only the replies that nodes sent to the node executing the camouflage algorithm. This is done to avoid unnecessary clutter in the figures, which themselves are already hard to interpret. For the simulations a network with 3600 nodes is used, hence having a size of 60×60 with a rigid grid employed as topology. However, in several circumstances only a fragment of the network might be shown, in order to magnify the patterns yielded by the algorithms. The vicinity radius of each node is set to be 3.

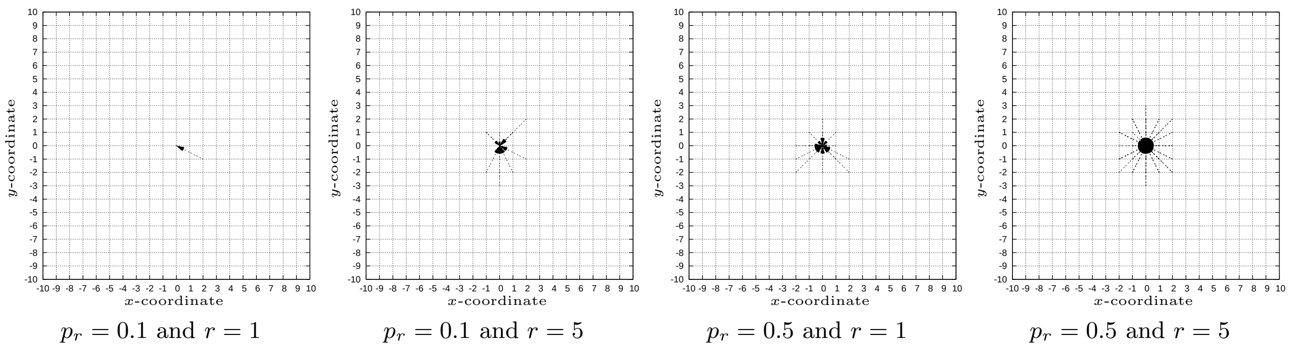
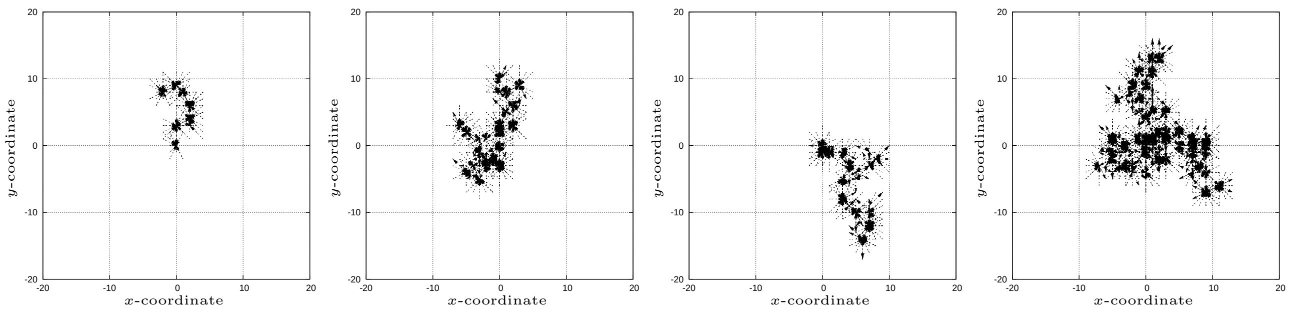


Figure 6.10: Repeated Firework for varying parameters r and p_r

Figure 6.10 shows the traffic patterns that can be observed for the Repeated Firework algorithm. The figures show only the replies of the Repeated Firework algorithm for varying participation probability p_r and number of rounds r. The leftmost picture shows the case where p_r is very low, and thus only a minute amount of nodes in the vicinity responses to the initial request of the node, which is positioned at location $(0, 0)$. However, repeating the process for 5 rounds the patterns become more dense and more similar to what is achieved with a higher probability p_r after a single iteration (see third picture from left). Of course, the choice of those parameters is depending on the communication pattern the Repeated Firework is supposed to camouflage, but the figures show that small increases in the parameters can lead to significant differences in the patterns yielded. Hence, the parameters should be chosen after detailed analysis of the patterns exhibited by the actual (to be camouflaged) algorithm. Furthermore, in case the unreliability is added into the picture, the actual p_r is even lower due to the possible failure of messages sent to the neighboring nodes.

However, the important algorithm to be used in context of the Evasive Data Storage algorithms is Wandering Repeated Firework. The simulations of this algorithm are carried out in the same setup as the Repeated Firework was and as the Wandering Repeated Firework can be seen as a refinement of the Repeated Firework algorithm, the respective parameters will be kept constant for the subsequent simulations amounting to $p_r = 0.2$ and $r = 5$.

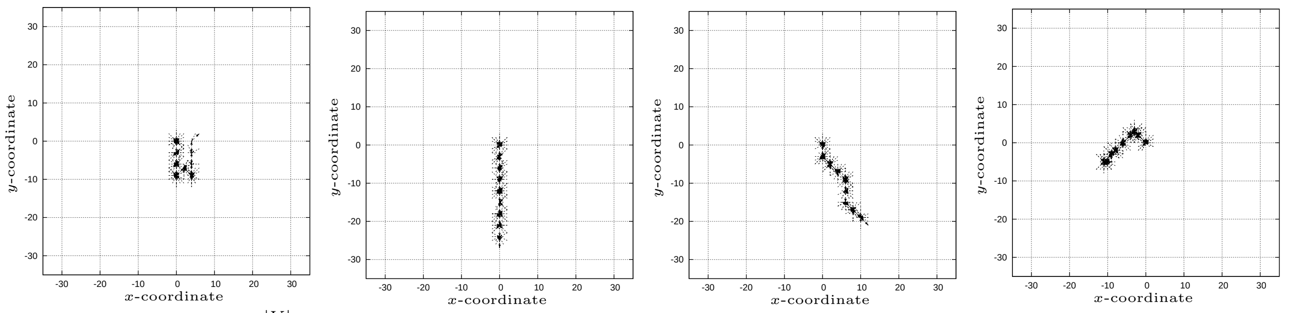
Figure 6.11 shows the behavior of Wandering Repeated Firework for varying parameters. However, in order to investigate the algorithm precisely another parameter is introduced: p_{sa}, which parameterizes with what probability a node continues to spread the wandering of the repeated firework. This can be seen as a better means to control the propagation of the wandering or as introduction of the unreliability of channels, however under a more suitable name. Either way, the leftmost figure shown in 6.11 shows the setup with will be most important for this work, i.e. where $spread = 1$ and $p_{sa} = 1.0$ holds. The precise reasons for that are simply that such a setup mimics the patterns observed in case of Evasive Data Storage algorithms and is hence used for their camouflage. The other pictures show the patterns yielded when the spreading is increased. And indicate that quite a clutter around the starting node is obtained, when spreading is high. This naturally, can be useful depending on the application, but would surely be more advisable if the underlying choice function used would be different (to use different choice functions only a minute change to the algorithm 3.7 has to be made, which is similar to the modification needed to obtain the generalization of Fractal Propagation; thus how to add choice functions to the Wandering Repeated Firework algorithm will not be given in detail). However, as there is only one case of interest, the results for different choice functions is only given for parameters matching



spread = 1, $K = 9$ and $p_{sa} = 1.0$ *spread* = 2, $K = 9$ and $p_{sa} = 0.7$ *spread* = 3, $K = 9$ and $p_{sa} = 0.5$ *spread* = 3, $K = 9$ and $p_{sa} = 0.7$

Figure 6.11: Wandering Repeated Firework for varying parameters *spread* and K

those of the leftmost diagram of 6.11 and are presented in figure 6.12.



UFurthest$k$ with $k = \frac{|V|}{2}$ GNodeFurthest with $k = 1$ GNodeFurthest with $k = 2$ DirectionV with $p_c = 0.05$

Figure 6.12: Wandering Repeated Firework with different Choice Functions

The figures reveal a substantial difference among the patterns resulting from using difference choice functions. The leftmost, showing the UFurthest$k$ is almost identical to the UVicinity case shown in figure 6.11 (leftmost picture) and causes the pattern to conglomerate around the initial node located at $(0, 0)$. The next two diagrams show the results for the GNodeFurthest choice function with varying parameter k, which allows the choice function to choose uniformly among the k furthest nodes available at each step. Here the patterns are what one would expect from such a camouflage algorithm. Similarly, the DirectionV choice function shown in the right diagram does not exhibit any kind of conglomeration, which is, however, heavily dependent on the probability p_c. To avoid the clutter of the UVicinity or UFurthest$k$ choice functions, p_c has to be chosen to be sufficiently low. Exact behavior of the choice functions will be investigated in the following Evasive Data Storage section and will provide more accurate investigation of the patterns that can be expected from the Wandering Repeated Firework Camouflage algorithm. Hence, the executions given so far shall be sufficient to understand the Repeated Firework and Wandering Repeated Firework at a qualitative level.

What has not yet been addressed is the effectiveness of the algorithms and most importantly the fulfillment of the Camouflage Security Property, which those camouflage algorithms naturally should adhere to. The reason for this, is that the Local Camouflage algorithms have to be put into context of the actual communication they intend to disguise. In the scope of this work, the actual communication will be the one exhibited by the Evasive Data Storage algorithms to follow, thus the respective analysis of the entropy and similar measures, incurred when using the Repeated Firework and Wandering Repeated Firework camouflage techniques are used, will be investigated along with those algorithms. Other cases, or general behavior of Repeated Firework and Wandering

Repeated Firework for arbitrary algorithms will not be considered, as the focus of the thesis is mainly on Evasive Data Storage.

6.3 Performance of Evasive Data Storage Algorithms

Now that the Camouflage algorithms have been covered, what remains to be analyzed is the other major notion that is introduced and developed in this thesis: Evasive Data Storage. The most important part of this section is the first one, where the properties of the *Choice Functions* are investigated. There are two reasons that make the investigation of the choice functions one of the most important part of the complete thesis: the first one is the direct connection to the Evasiveness Security Property that describes the behavior what is needed such that an Evasive Data Storage actually proves useful for the protection of data and the second one is that choice functions not only are the crucial building block of all Evasive Data Storage algorithms presented in this work, but also for the generalization of the Fractal Propagation notion. Therefore, can also be assumed to have applications beyond the two techniques of Camouflage and Evasive Data Storage only.

6.3.1 The Choice Functions

An integral part of every Evasive Data Storage algorithm presented in the preceding chapters is the concept of the choice function. Every evasive algorithm running on a hot node $\tilde{s}$ needs to choose a new node where to displace $\tilde{s}$'s data to, which is done in the New Storage Node Choice algorithm. The part in the New Storage Node Choice algorithm that makes the actual decision is the choice function, making it a crucial element guiding where data is allowed to go to and highly influential on security properties of the evasion process. Along with the basic notion of Evasive Data Storage and its simplest implementation, the Simple Evasive Data Storage algorithm, several choice functions have been introduced, but a thorough analysis of their effectiveness was not provided. This section aims at clarifying the properties the choice functions exhibit without a tight context to any of the Evasive Data Storage algorithms.

In order to do so the choice functions are investigated in an idealistic environment, aiming at revealing intrinsic properties dealing without other parameters, like message omission, delays, or failures. Thus the simulation environment that will provide insights into those function will consist of a *rigid grid*, hence making the assumption that nodes are equally distributed in a rectangular environment and, except for the nodes at the boundary of the grid, have a an equal number of vicinity nodes. Every node is assumed to work perfectly over the time span the simulations are carried out, have perfect communication primitives available, without need for retransmission or similar, and constant access to all the nodes in its vicinity. The basic parameters for the simulations carried out are summarized in table 6.1. The virtual network consists of 10000 nodes and in order to obtain interesting information regarding the security properties of the choice functions, for instance probability distributions, simulations are iterated 5×10^6 times. The underlying principle that is used for the argumentation regarding probabilities is *the law of large numbers*, which states that repeating experiments iteratively and collecting the outcomes/probabilities, will converge to the true probability, the larger the number of iterations is. This is also why the simulations are repeated 5×10^6, in order to guarantee that the results are close to the true probabilities or probability distributions. This principle of course is not only applied in this section, but in others as well, but should be mentioned explicitely for this utmost important analysis.

The basic setup assumes that a hot node performing a New Storage Node Choice algorithm is located in the rigid grid at position $(0, 0)$ and each evasion is performed successfully at any attempt. Furthermore, in order to investigate the intrinsic properties of the choice functions, several values for parameters are needed, those are separated with a slash in the table 6.1 and will be mentioned explicitly in the explanations accompanying the respective figures.

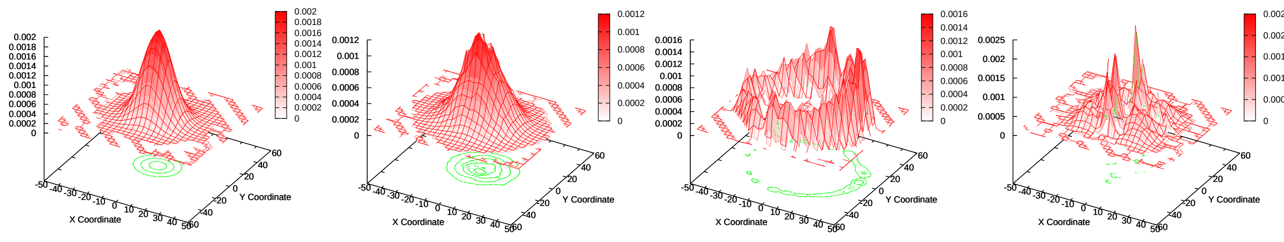
As mentioned in the security sections of the Evasive Data Storage algorithms [5] the major property of a choice function is the exhibited probability density for the nodes that surround the initial hot node to become

[5]Especially in the respective section of Simple Evasive Data Storage.

Type of Network	**Size of Network**	**Number of Sensor Nodes**
Rigid Grid	100×100	10000
Number of Iterations	**Number of Evasive Steps**	**Vicinity Radius**
5×10^6	6 / 17 / 18	7 / 10 / 10
Initial hot node Location	**UFurthest$k$**	**DirectionV**
$(0, 0)$	$k = \frac{1}{6} \cdot \lvert V \rvert$	$p_c = 0.005$ / $p_c = 0.008$ / $p_c = 0.008$

Table 6.1: Choice Function Simulation Parameters

the hot node after e evasive steps. The anticipated distribution would be a uniform distribution in the region consisting of nodes that can be reached after e steps or the complete network, which in our simulations consists of a rectangular grid. Thus the first results of the simulations shall investigate the probability density for finding a hot node in the vicinity of the initial hot node after a certain number of evasive steps. This probability distribution can also be interpreted as the success probability an adversary $\mathcal{A}$ has, when he remembers a hot node (in this case at location $(0, 0)$) and tries to access data in the region surrounding $(0, 0)$ after a certain amount of time or number of evasion steps, which can be considered to be isomorphic. The description just given also stresses the significant connection of the choice function with the Evasiveness Security Property. Meaning that if a uniform distribution can be achieved using one of the choice functions, then the Evasiveness Security Property is fulfilled in the best possible way. In the latter statement the speed at which the uniform distribution is reached also plays a significant role, as a look at the Evasiveness Security Property given in the Data Oriented Techniques Chapter verifies. Hence, the subsequent investigation of choice functions is equivalently to determining the effectiveness of all Evasive Data Storage algorithms of this thesis.

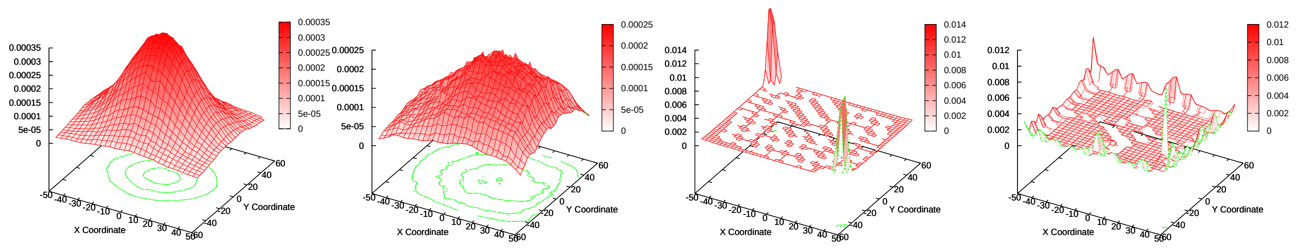


UVicinity UFurthest$k$ GNodeFurthest DirectionV

Figure 6.13: Probability Density for Hot Nodes after 6 Evasion Steps

To start this investigation the fundamental choice functions are looked at first. Figure 6.13 shows the probability distribution for each of the basic choice functions after 6 evasive steps. The assumed vicinity radius, i.e. the radius of the communication primitive that each node disposes of, is set to 7 in this simulation run, which entails that not all nodes in the grid could have been reached by after the 6 evasive steps. The actual region is a circle with center $(0, 0)$ and approximate radius $7 \times 6 = 42$. Looking at the resulting figures, the simple UVicinity choice function, yields a distribution that resembles a three dimensional normal distribution with center around the initial hot node. Hence, using this choice function will lead yield a hot node that is very close to the initial node $(0, 0)$, which of course allows an adversary $\mathcal{A}$ to be quite successful when picking nodes close to the initial one. A very similar, distribution is exhibited by the UFurthest$k$choice function, only that the focus around the location $(0, 0)$ is more diffuse. The GNodeFurthest choice function marks the perimeter of the circular region consisting of all the reachable nodes mentioned above. The goal of this choice function is to move the data as far as possible from the generating node, hence the result is a probability density in form of a ring consisting of the nodes furthest from the location $(0, 0)$ after the 6 evasive steps. The adversary $\mathcal{A}$

thus will not be successful when trying nodes close to the initial node, but rather estimating the region and picking nodes on the perimeter, which however can be considered as a difficult task. The last choice function, DirectionV, reveals an irregular pattern with several peaks, which can be accounted to the used pseudo random number generator. There is no particular pattern recognizable, which makes this choice function the closest one to the desirable uniform distribution.

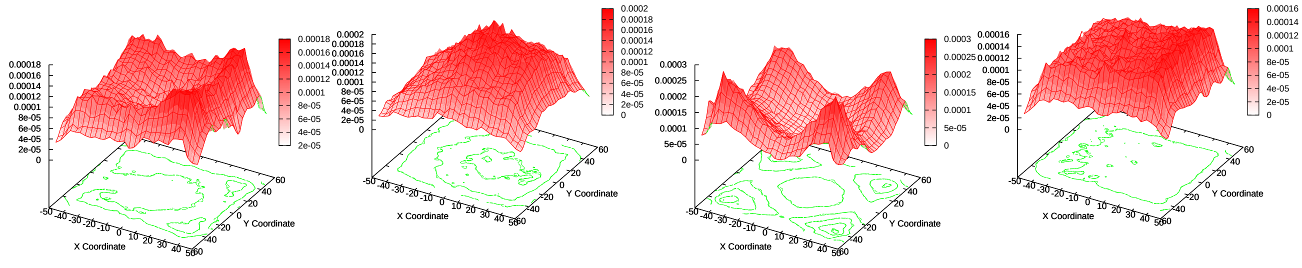


UVicinity UFurthest$k$ GNodeFurthest DirectionV

Figure 6.14: Probability Density for Hot Nodes after 17 Evasion Steps

Of course, in order to give a more accurate estimate of the intrinsic properties each of the choice function exhibits, the development of the probability density after a larger number of steps is needed. Furthermore, the number of steps should be chosen such that all nodes in our grid can be reached. This is equivalent to a large number of evasive steps in a real sized sensor networks, with greater size and mount of nodes as well as to a hot group of similar or smaller size. Figure 6.14 shows the respective probability densities after 17 evasive steps, which more that certainly allows all of the nodes to be reached in the 100×100 grid used. Furthermore, this simulation data reveals some of the rather unpleasant properties of the choice functions. The UVicinity choice function has not changed drastically compared to the density after 6 evasion steps: the initial node is still the center of the peak of the distribution, making the surrounding nodes most probable for harboring the data after 17 evasion steps. However, the steepness of the peak is lower, thus making the region an adversary needs to consider in his effort to find hot data larger than it was after only 6 evasive steps. The prediction for UVicinity based on the two densities is clear: slowly the peak around the initial hot node will erode, hence approached very slowly a uniform distribution. However, this can take too many evasion steps to be considered useful. Looking at the UFurthest$k$function, the 17 evasive steps lead to a density that does not exhibit a high peak around the location $(0, 0)$, but a very shallow hill is recognizable. Thus with UFurthest$k$an adversary will not be able to pick a hot node in the vicinity of the initial location, as the probability density makes a large region around $(0, 0)$ possible home of hot nodes. But, unfortunately, a uniform distribution is still not reached as nodes far away from the initial node still have a significantly less probability for becoming hot that nodes closer to the initial node. As with the UVicinity choice function the UFurthest$k$will slowly approach a desirable and approximate uniform distribution, but still needs too many evasive steps to do so. The last two choice functions reveal a quite unanticipated density after 17 steps: the GNodeFurthest causes a concentration of hot nodes at the location furthest from the initial node, which is unpleasant as an adversary can access data easily in a network with such a distribution. Similarly, the DirectionV causes the boundary of the grid to become the hot spot for data. Therefore, both of those are not suitable choice functions to be used to a large number of evasion steps.

Since none of the choice functions exhibits the anticipated uniform distribution for nodes to be hot in the network, after an acceptable number of evasion steps, an attempt should be made to combine strengths of the basic choice functions to achieve a distribution close to a uniform distribution in a small number of evasion steps. Thus the notion introduced in the Data Oriented Techniques Chapter of how to combine the basic choice functions to form more complex ones has to be investigated next. As not all combinations of the basic choice functions do make significant leaps towards a uniform distribution only the combinations of the GNodeFurthest and DirectionV with the UFurthest$k$are investigated. The decision to combine those functions



(GNodeFurthest, $\frac{e}{3}$, UFurthest$k$)
(DirectionV, $\frac{e}{3}$, UFurthest$k$)
(GNodeFurthest, $\frac{e}{2}$, UFurthest$k$)
(DirectionV, $\frac{e}{2}$, UFurthest$k$)

Figure 6.15: Probability Density for Hot Nodes after 18 Evasion Steps (combined Choice Functions)

can easily be accounted for: UFurthest$k$shows a trend towards a uniform distribution, but does that too slowly as it is centered around the initial node, thus it makes sense to first use UFurthest$k$at a location further away from the initial hot node. As both choice functions GNodeFurthest and DirectionV provide such a property, their combination seems to combine strengths that are needed to achieve a good approximation to the wanted uniform distribution.

Figure 6.15 shows the resulting probability densities for the combined choice functions after 18 evasion steps, hence making the distributions comparable to the ones found for the basic choice functions. The first one is (GNodeFurthest, $\frac{e}{3}$, UFurthest$k$), which lets the choice function GNodeFurthest make the first 6 steps before UFurthest$k$is used for all remaining ones. The result is a distribution that is too much influenced by the GNodeFurthest choice function, resulting in peaks at the regions furthest from the initial node's location $(0, 0)$. The same kind of combination, but with the DirectionV function making the first 6 choices, results in a distribution that is still centered around the initial hot node and is therefore less desirable. Having seen that those combinations, do not yield the anticipated uniform distribution another attempt is taken, however allowing for more evasive steps to be guided by the first choice function. The (GNodeFurthest, $\frac{e}{2}$, UFurthest$k$) function, which now lets GNodeFurthest make 9 choices shows even more extreme peaks at the furthest locations of the initial node, which is not an anticipated distribution at all. However, the (DirectionV, $\frac{e}{2}$, UFurthest$k$) combination exhibits a density that is very close to a uniform distribution, and is therefore the best choice function in order to avoid adversaries from picking hot nodes easily in any of the Evasive Data Storage algorithms.

To summarize, the results above investigated all basic choice functions and several of their combinations. Each of them lowers the probability for the initial node to remain hot after several evasive steps significantly, but each function has different properties regarding the probabilities for other nodes in the network to become the final hot node after the respective number of evasive steps. Naturally, the anticipated density distribution for this probability is the uniform distribution, making all nodes in the network equally likely to become a hot node. And in the same way protecting hot nodes from being easily found by an adversary and providing an adequate satisfaction of the Evasiveness Security Property. The investigations reveal that only the (DirectionV, $\frac{e}{2}$, UFurthest$k$) combined choice function shows a distribution close to uniform. However, to achieve this several factors especially the topology of the network, the vicinity radius and the expected number of evasion steps play a major role. As most of those can be estimated, especially in the case of hot nodes, the appropriate value for the combined choice function's second component can be calculated, therefore allowing the (DirectionV, $\frac{e}{2}$, UFurthest$k$) combined choice function to provide an almost uniform distribution in most scenarios. This is therefore the choice function that needs to be employed in order to force the success probability of the adversary towards the lower bound of $\frac{h}{n}$ (as anticipated by the Evasiveness Security Property). The densities investigated are of course simplifications, as in real networks several data items are stored evasively, which will therefore lead to densities resulting from an overlapping of the ones presented here. However, a good estimate of the effectiveness of the choice functions even in the more complex context of several hot nodes can be derived from the results obtained here.

Now what still should be evaluated is the probability that an adversary who returns to a hot node, he remembered, will be successful when picking that same node again. This allows for a simple and direct comparison to conventional storage algorithms, where an adversary has a probability of 1, to obtain updated data once he has found a hot node. In order to show the superiority in this simple but relevant scenario, the same simulation environment is assumed. Hence, the hot node is located at location $(0, 0)$ and several evasion steps are taken. What is being evaluated is the probability that the initial node $(0, 0)$ will remain hot after a certain number of evasion steps, which is the same probability for an adversary picking a former hot node to be successful after several evasion steps.

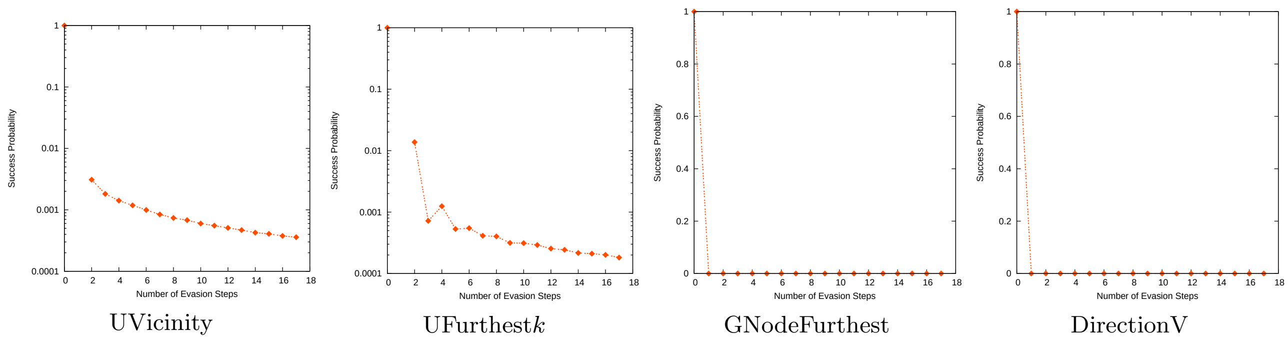


Figure 6.16: Adversaries Success when picking a former Hot Node

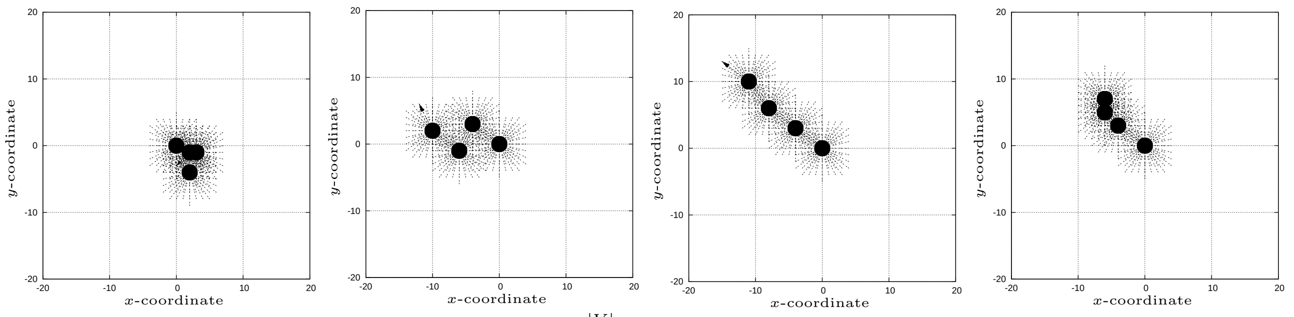
Figure 6.16 shows the results, i.e. the probability that an adversary is successful. Each of the diagrams, one for each choice function, shows the probability for being successful depending on the number of evasion steps performed. Of course, all diagrams show a probability of 1 for the case that zero evasion steps have been performed. Both the results for the UVicinity and the UFurthest$k$choice functions are shown on a logarithmic scale, revealing that the adversary's success probability approaches zero quite fast with the number of evasion steps. Thus to beat such an adversary a low number of evasion steps is sufficient. Similarly, the GNodeFurthest and DirectionV show that after the first evasion step the probabilities for the initial node to be hot after any number of evasion steps is as good as equal to zero. This can also be seen in the probability distributions discussed above, where both the GNodeFurthest and DirectionV, caused nodes further away from the initial node to become hot.

This finishes the investigation of the properties of the different choice functions and reveals that any choice function can significantly increase the failure probability for an adversary looking for hot nodes, but especially the ones exhibiting probability densities close to uniform distributions are desirable. Furthermore, the results obtained here directly show which choice function needs to be used in order to guarantee the Evasiveness Security Property of the Evasive Data Storage notion. Similarly, the probability distributions that show intrinsic properties of the choice functions also allow inferences to be made about the generalization of the Fractal Propagation algorithm and can prove helpful in finding even more suitable choice functions for usage in this family of camouflage algorithms. After showing the intrinsic properties of choice functions, a closer look at the actual properties of the Evasive Data Storage algorithms should be taken.

6.3.2 THE ALGORITHMS

The major properties of Simple Evasive Data Storage as well as Location Bound Evasive Data Storage can be deduced from the results given in the previous section on choice functions. Hence, the purpose of this section is to put these results into the context of those two algorithms, give a more detailed insight into the behavior in a sensor network and evaluate the usage of camouflage for those two algorithms.

The most basic algorithm that has been developed to implement the Evasive Data Storage notion is the Simple Evasive Data Storage algorithm. The main parameter that is used for this algorithm is the choice function and will therefore account for the amount of simulations done. Besides, this influential parameter, another important question that arises is how the traffic patterns resulting from using Simple Evasive Data Storage in a sensor network look like. Figure 6.17 shows the algorithm for varying choice functions and stresses that the results that have been obtained in the previous section are directly applicable to an actual implementation of the algorithm. The diagrams obtained from the simulations show an execution of the algorithm in a sensor network of 3600 nodes in a rigid grid topology, i.e. a network of size 60×60. The vicinity radius used for the nodes is 5, hence allowing nodes to choose among a larger number of nodes to displace their data to. Furthermore, the number of steps is limited to 4, which is considered to be a good value to preserve clarity of the diagrams, but still allow for properties to be identified.



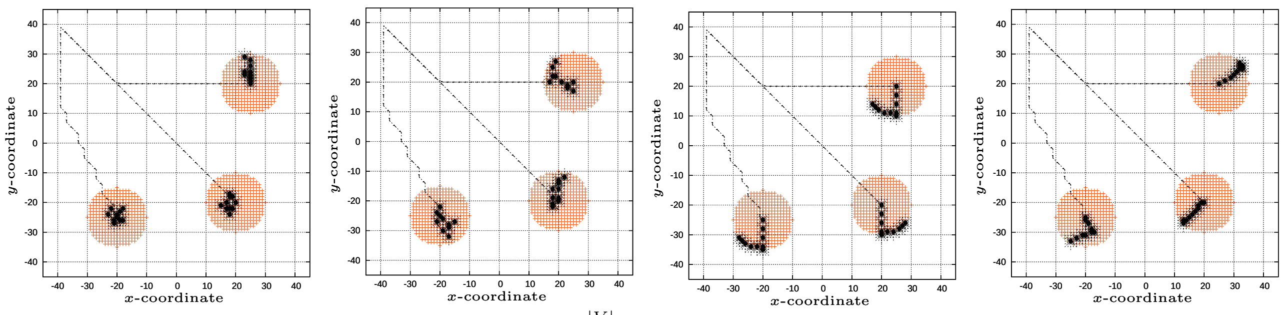
UVicinity   UFurthest$k$ with $k = \frac{|V|}{2}$ GNodeFurthest with $k = 1$ DirectionV with $p_c = 0.5$

Figure 6.17: Simple Evasive Data Storage for varying Choice Functions

The diagrams of figure 6.17 are executions of the Simple Evasive Data Storage algorithm in a network where an initial hot node is located at $(0, 0)$. The diagrams stress the properties that have been observed in scope of the detailed choice function evaluation of the preceding section. The UVicinity and UFurthest$k$ functions tend to keep the data close to the original generating node, which in this case is equivalent to proximity to the $(0, 0)$ coordinate. The GNodeFurthest tends to move data quickly far away from the initial node and the DirectionV function can be seen to perform similar, if the p_c probability is low enough. Using higher values for p_c the choice function resembles the UVicinity behavior, as frequent changes in direction will cause a concentration around the generating nodes, whereas low values will lead to behavior more similar to that of the GNodeFurthest function. All of these properties are nicely supported by the diagrams in figure 6.17. However, the crucial point that has to be mentioned is the connection between which choice function is employed and how well the Evasiveness Security Property is fulfilled by the Simple Evasive Data Storage algorithm. Here, the results of the investigation of the choice functions can already be applied. For details, refer back to the probability distributions given for each of the choice functions and most importantly for their respective combinations. Imagining a network where Simple Evasive Data Storage with UVicinity is used, the obvious distribution that results is one where nodes close to the original hot node are likely to store data, thus making the probability for picking a hot node quite high. Thus in terms of the Evasiveness Security Property an adversary $\mathcal{A}$ that knew the position of the initial hot node at some time t can restrict $R_{eds}(s, t')$ to a small region around the initial hot node. This is the result of the conglomeration tendency that UVicinity exhibits, which is especially strong when t' is close to t, or equivalently when the number of evasion steps is low. Furthermore, the probability distribution obtained for UVicinity also suggests that $\mathcal{A}$ has a high success probability p_{eds} in the small region $R_{eds}(s, t')$, which clearly is not favorable. However, using the choice function that exhibited an almost uniform distribution, the (DirectionV, $\frac{e}{2}$, UFurthest$k$) function, a clearly better satisfaction of the Evasiveness Security Property can be achieved. Similarly, the results obtained in the section concerned with choice functions can be used to infer conclusions about the Simple Evasive Data Storage algorithm using the respective function.

Hence, the main result is that the choice function has a substantial influence on the effectiveness of the Simple Evasive Data Storage algorithm and the Evasive Data Storage notion as a whole.

Now that the first Evasive Data Storage algorithm has been investigated a look has to be taken at the Location Bound Evasive Data Storage algorithm, whose main contribution is to provide increased efficiency regarding query and event costs. As the notion is based on Simple Evasive Data Storage the same parameter plays the most significant role, i.e. the used choice function. However, as the Location Bound Evasive Data Storage algorithm restricts the area that is available for the evasion of data, a bound is imposed on the region that can be used. Hence, the attention is shifted to the behavior of the evasion process within such a region, or more precisely within a hot group. Before examining the Location Bound Evasive Data Storage notion in context of the used choice function, sample executions in a network using this refinement should be looked at. In order to do so, a network of size 80×80 is used and the vicinity radius of each node is restricted to 3. The simulated network is setup with three hot groups, whose members are nodes located within a radius of 10 from the center of the group, which is assumed to be the location of the respective head node. For the executions the number of evasion steps is setup to be limited to 10.

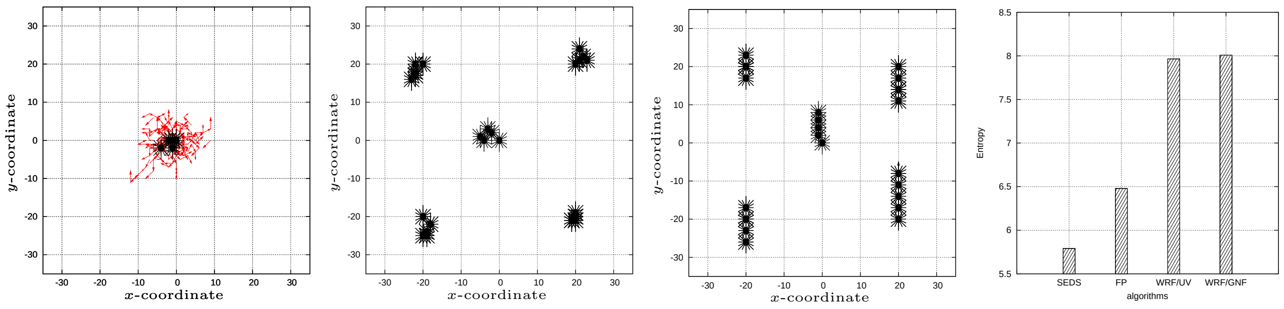


UVicinity UFurthest$k$ with $k = \frac{|V|}{2}$ GNodeFurthest with $k = 1$ DirectionV with $p_c = 0.05$

Figure 6.18: Location Bound Evasive Data Storage with different Choice Functions

The results of the simulation are presented in figure 6.18, where the hot groups are made visible through little red marks and stand for nodes that share the same $HGID$. Data is generated in a node in the upper left corner and depending on the type specifier send to the three available hot groups. The head nodes, which are located at the center, receive the respective data items and start to store the data item using the Simple Evasive Data Storage algorithm restricted to members of the hot group. The figures indicate that the behavior of the hot groups reflects that of a sensor network, just at a smaller scale. Thus all the results that have been obtained for the Simple Evasive Data Storage case and therefore also from the investigation of the choice functions can be adapted for the Location Bound Evasive Data Storage algorithm as well. The main difference however is that the scale has changed, i.e. the region R_{eds} is bounded by the number d or more precisely by the region R_d induced by the number d and the setup procedure. It is clear that in case of Simple Evasive Data Storage the region R_{eds} is bounded by the overall network, in case of Location Bound Evasive Data Storage $R_{eds} \leq R_d$ holds true. However, within this region the same behavior and same results regarding the choice functions apply. This can be seen in figure 6.18 as well, where the UVicinity tends to conglomerate around the head node and GNodeFurthest quickly moves data to the boundaries of the hot group. Thus the goal to achieve is to have a uniform distribution in the hot group regarding the probability of becoming a hot node. Hence, the most suitable choice function is (DirectionV, $\frac{e}{2}$, UFurthest$k$), as shown in the section on choice functions. However, it is important to stress that the combination parameter $\frac{e}{2}$ of that choice function has to be chosen accordingly to the size of the hot group. Summarizing, the properties of choice functions allow Location Bound Evasive Data Storage to satisfy the Evasiveness Security Property given that the right choice function is employed.

Now that the algorithms and the security properties have been investigated, another issue remains that has to be addressed: the usefulness of Camouflage in context of Evasive Data Storage. The communication



Fractal Propagation — WRF and UVicinity — WRF and GNodeFurthest — Entropy

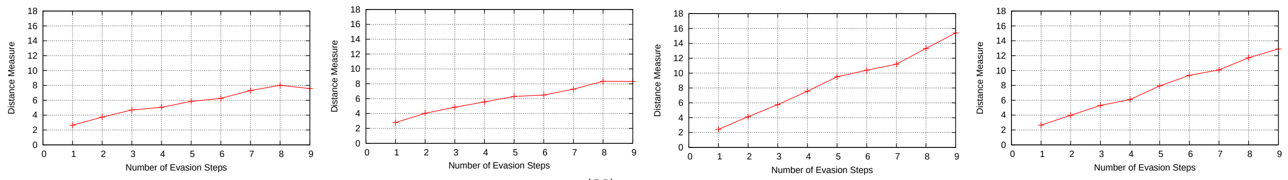
Figure 6.19: Evasive Data Storage with Camouflage Techniques

patterns that emerge when storing data in a sensor network using the Simple Evasive Data Storage have a close resemblance with the ones seen in the Wandering Repeated Firework figures, which is easily seen when comparing figures 6.17 and 6.12. This is no coincidence, because the basic notion of Repeated Firework and Wandering Repeated Firework was conceived with algorithms in mind that exhibit precisely such patterns. It is obvious that global camouflage, like Fractal Propagation, does not help much with algorithms similar to Simple Evasive Data Storage, which concentrate their majority of communication on a very local region, as the fake messages that are supposed to provide the actual camouflage have almost no possibility to kick in and would rather lead an adversary $\mathcal{A}$ with traffic analysis to hot nodes using Evasive Data Storage as storage method. However, with the Wandering Repeated Firework algorithm a perfect match for the Simple Evasive Data Storage notion is available. The same is true for the Location Bound Evasive Data Storage notion, as it can be seen as a refinement that does not change the basic patterns exhibited by Simple Evasive Data Storage. Thus the advantages of using Wandering Repeated Firework will be described in the following concentrating on Simple Evasive Data Storage only and assuming that the same results apply to Location Bound Evasive Data Storage as well.

Figure 6.19 shows the combination of Camouflage and the Simple Evasive Data Storage algorithm. The simulation network used for those figures consists of 3600 nodes in a rigid grid topology. Sensor nodes are assumed to have a vicinity radius of 3 and 5 evasion steps of the algorithm are looked at. The leftmost diagram shows the addition of Fractal Propagation to the algorithm, which causes fake messages to be sent very close to the communication pattern of the storage algorithm. Such a behavior rather increases the traffic close to the hot node and can cause the attention of an adversary to be shifted towards hot nodes rather than doing the desired opposite.[6] Looking at the addition of Wandering Repeated Firework Camouflage to the network, a different picture emerges. Now besides to original Simple Evasive Data Storage pattern, identical fake patterns can be observed, which successfully lure an adversary away from the actual hot node. To achieve this, the Wandering Repeated Firework algorithms parameters have to be chosen accordingly, i.e. the number of rounds $r = 3$, the spreading is set to $spread = 1$ in order to match the Simple Evasive Data Storage algorithm. Of course, the depth of the Wandering Repeated Firework can either be chosen to be infinity, then the algorithm causes a complete faking of the wandering of a data item for the life time of the network, or a finite value can be ascribed to the depth, which is more energy efficient and useful for temporal distraction away from the actual data item. The latter should be triggered by overhearing the communication messages used by the Simple Evasive Data Storage algorithm, and thus causing the camouflage algorithm to be employed where it is needed. The observations for the superiority of the Wandering Repeated Firework algorithm over the Fractal Propagation in case of Evasive Data Storage, is stressed even more by the entropy values yielded. The rightmost diagram of figure 6.19 clearly shows that Wandering Repeated Firework increases the entropy more effectively than the Fractal Propagation and therefore is more likely to protect data successfully according to the Camouflage

[6]The Fractal Propagation parameters are the same as those used in figure 6.2.

Security Property.



UVicinity UFurthest$k$ with $k = \frac{|V|}{2}$ GNodeFurthest with $k = 1$ DirectionV with $p_c = 0.05$

Figure 6.20: D_{hot} values for varying Choice Functions for Evasive Data Storage

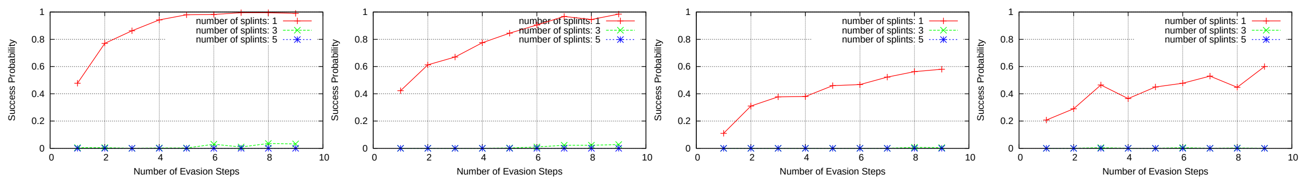
Before the discussion of the Simple Evasive Data Storage and Location Bound Evasive Data Storage is finished a look has to be taken at the values of the distance measure D_{hot} obtained for varying choice functions. Figure 6.20 shows how the measure stresses the different properties of choice functions seen in the previous section once again. Here the simulation is carried out in the same network used for the Camouflage enhancement evaluation and uses 25 data items stored evasively to calculate the D_{hot} depending on the number of evasion steps. The diagrams show that both the UVicinity and UFurthest$k$ functions tend to position the data items rather close to each other even after many evasion steps or equivalently after longer periods of time. However, the GNodeFurthest and DirectionV show the more desired property to disperse hot nodes far away from each other. Those results are naturally in correspondence to the figures given in the choice function section. This finishes the investigation of the Evasive Data Storage algorithms and shows that a significant increase in the security of data in a sensor network can be achieved, given that the right choice function is employed.

6.3.3 Evasive Data Storage with Data Splitting

To evaluate the Data Splitting notion a quite simple performance measure is used. In the simulation environment several malicious nodes of an assumed adversary $\mathcal{A}$ are placed. Those nodes will wait for data items that are stored evasively in the network to be evaded into their palms, thus allowing $\mathcal{A}$ to access data of the sensor network without actually searching for hot nodes. This is done with both ordinary Evasive Data Storage and with the addition of Data Splitting to the process. The results then give a fair indication of the effectiveness of the Data Splitting notion, furthermore its effectiveness is also put into context of the lifetime of the network towards the end of this section. The evaluation also includes the different choice functions that can be used, however as the results do not differ significantly they can be seen as being provided for the sake of completeness.

For the simulation and the involved evaluation of Evasive Data Storage with Data Splitting several simplifications are employed. The first is that the case of using the Data Splitting notion in the Location Bound Evasive Data Storage case can be omitted as the same results will be achieved as with Simple Evasive Data Storage. The difference is that a hot group is formed with a smaller number of nodes, but otherwise resembles the a complete network. Hence, focus will be on a complete (virtual) sensor network using the Simple Evasive Data Storage algorithm. As mentioned above, to evaluate the advantage of the Data Splitting approach several malicious nodes are assumed to be in the network, which wait for data to be evaded into them. The simulations intend to compare the success probability, i.e. the amount of complete data items that the malicious nodes were able to obtain after a certain number of evasion steps relative to all data items sent through the network. The main weight is laid on the comparison of Simple Evasive Data Storage and the modification Simple Evasive Data Storage with Data Splitting for varying parameters.

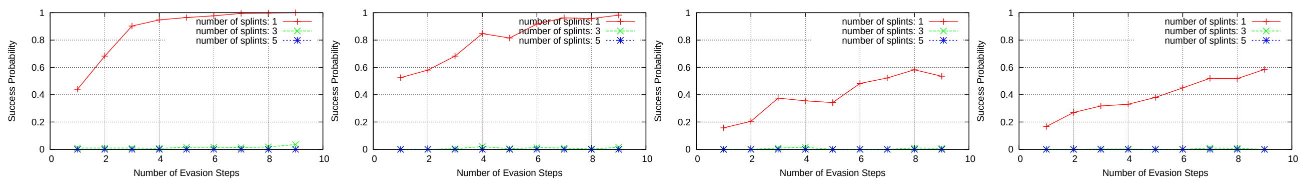
The simulations are interpreted from two fundamentally different perspectives: *non-cooperative malicious nodes* and *cooperative malicious nodes*. The former assumes that a certain number of malicious nodes is located in the network (distributed equally densely), but that in order for a data item to be captured each of those nodes needs to collect all created splints separately. The latter on the other hand, allows for cooperation among the equally distributed malicious nodes, meaning that as long as a group of malicious nodes collects all splints of a data item, it is counted as having been captured. Of course, the cooperative malicious nodes assumption is more severe than the other one and can also be considered to be more realistic, as in most cases the adversary can query all its nodes to collect the caught splints. However, the cooperative case can only be considered valid for very small amounts of malicious nodes, as infiltrating the network on a large scale is just not realizable. On the other hand, the non-cooperative malicious node setup gives a better estimate for the case only a few malicious nodes are to be found in the network and therefore cooperation between the nodes becomes less severe.



50 percent malicious nodes   30 percent malicious nodes   10 percent malicious nodes   5 percent malicious nodes

Figure 6.21: Data Splitting with UVicinity and non-cooperative Malicious Nodes

The discussion of the Data Splitting this section will provide starts with the case of non-cooperative malicious nodes. For the simulation environment a network is setup that is similar to the ones used mostly in this chapter, namely with 3600 nodes in a rigid grid topology and with a vicinity radius of 3. For the evaluation of the success probability, a data item is assumed to be generated at location $(0, 0)$ and the probability indicates the success an adversary has in collecting this data item through his malicious nodes infiltrating the network. In order to obtained this probability the law of large numbers is used as in most of the simulation results presented in this chapter. Figure 6.21 shows the results obtained for the case of using the UVicinity function. The leftmost figure shows the results for a large amount of malicious nodes, namely 50 percent of the nodes in the network are malicious. Hence, the ordinary case, where no Data Splitting is used, which is the same as using Data Splitting with only a single splint, allows non-cooperative nodes to quickly collect the data item even after a few evasion steps. Looking at the graph of the 3 splints and 5 splints cases the success probability remains closely to zero. The first slow rise of the success probability is seen for 3 splints, after several evasion steps, namely 8. Whereas 5 splints cause the probability to stay close to zero for all 9 evasion steps looked at. Thus even with such a dense distribution of malicious nodes in the network, which is assumed to take form of an uniform distribution, the usage of merely 3 splints increases security drastically. Of course, when the number of malicious nodes is reduced even further, to 30 percent, 10 percent and 5 percent, as shown in figure 6.21, the success probability for the adversary decreases, even in the 1 splint or ordinary Simple Evasive Data Storage case. However, it is important to keep in mind that the figures only look at 9 evasion steps and therefore can given a good comparison of the properties of the Data Splitting for varying l parameter. And for the Simple Evasive Data Storage to give away data items with a high probability after only 9 evasion steps and 5 percent, as indicated by the right most figure, can be considered to be an unpleasant property.

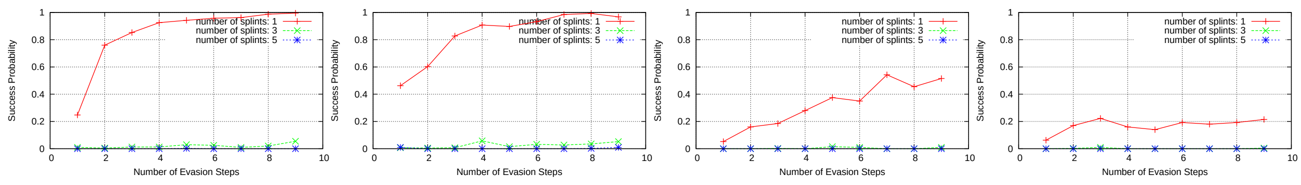


50 percent malicious nodes   30 percent malicious nodes   10 percent malicious nodes   5 percent malicious nodes

Figure 6.22: Data Splitting with UFurthest$k$ ($k = \frac{|V|}{2}$) and non-cooperative Malicious Nodes

The next simulation results are given for the sake of completeness, as already stated. Figure 6.22 shows the behavior for the case the UFurthest$k$ choice function is employed for guiding the Simple Evasive Data

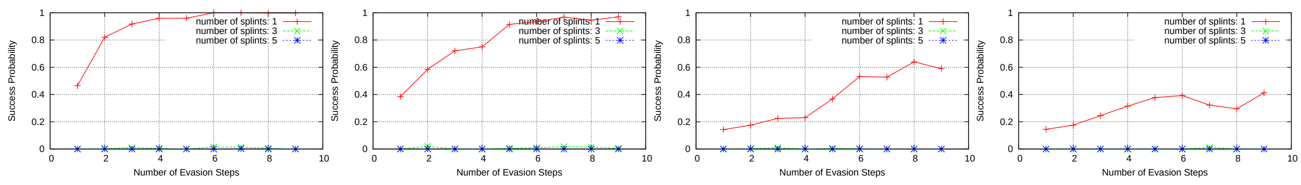
Storage algorithm. The figures show a close resemblance to the UVicinity case, which is obvious as both choice functions UVicinity and UFurthest$k$ have already been proved to have a similar distribution for low values of performed evasion steps.



50 percent malicious nodes 30 percent malicious nodes 10 percent malicious nodes 5 percent malicious nodes

Figure 6.23: Data Splitting with GNodeFurthest ($k = 1$) and non-cooperative Malicious Nodes

The more interesting case is the usage of the GNodeFurthest choice function, whose performance is indicated by the graphs shown in figure 6.23. Here the graphs seem to have similar properties to the previous ones, however the less malicious nodes are assumed to be located in the network the less the graph shows coherence with the graphs for UVicinity and UFurthest$k$. Especially, in the 5 percent case, the deviation cannot be simply explained with insufficient amount of simulations in order to make the law of large numbers do its magic. Rather the explanation is that less nodes are considered for evasion, thence making it also less probable to encounter malicious nodes. This is a side effect of the way the choice function works, which naturally has its advantages, but also disadvantages as seen in the section devoted to the analysis of choice functions.



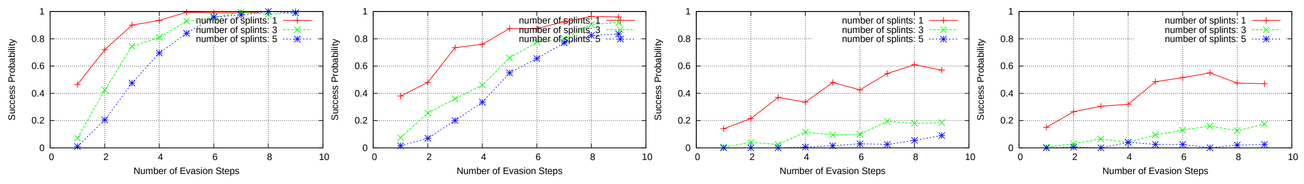
50 percent malicious nodes 30 percent malicious nodes 10 percent malicious nodes 5 percent malicious nodes

Figure 6.24: Data Splitting with DirectionV ($p_c = 0.01$) and non-cooperative Malicious Nodes

The last case to consider is that of using the DirectionV choice function. Its graphs are an intermediary between UVicinity and GNodeFurthest, where the degree of coherence to one of the two choice functions is determined by the used p_c, the graphs in 6.24 are obtained by setting $p_c = 0.01$ and therefore yielding neither complete resemblance with UVicinity nor with GNodeFurthest. This has been already observed before, that increasing the p_c probability tends to make DirectionV behave like UVicinity and a low p_c is causing a greater similarity to GNodeFurthest. Invariably, all figures however show that using only 3 splints does increase the security of the Evasive Data Storage notion noticeable.

Now that a thorough examination of the Data Splitting algorithm for the non-cooperative case of malicious nodes was given, where the emphasis was on the probability that a single malicious node can collect all the splints in the network. The next part of this section will allow for cooperation among the malicious nodes, which naturally leads to a more hostile environment for splints and the Data Splitting notion.

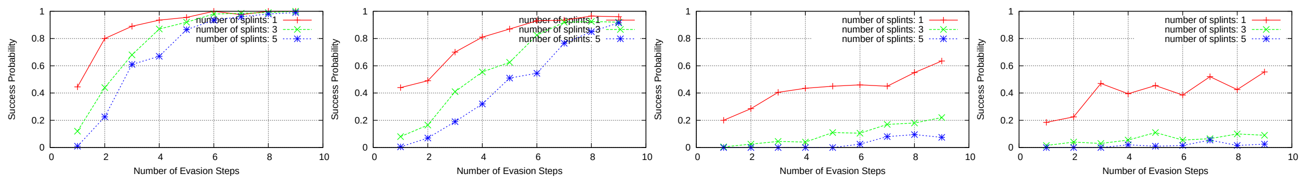
As with the non-cooperative case, the evaluation is made for all four fundamental choice functions that were looked at. None of the combinations are looked that, because it can be argued that those elementary results offer sufficient information to infer about the behavior of the more complex choice functions that can be obtained by using the combination method given in the scope of this work. To start with the discussion of the cooperative case, the UVicinity choice function is discussed first. Figure 6.25 shows the graphs obtained when malicious nodes do not needs to collect all splints individually, but can rely on the success of their malicious peers. The left most figure, where every second node in the network is malicious, shows that due



50 percent malicious nodes 30 percent malicious nodes 10 percent malicious nodes 5 percent malicious nodes

Figure 6.25: Data Splitting with UVicinity and cooperative Malicious Nodes

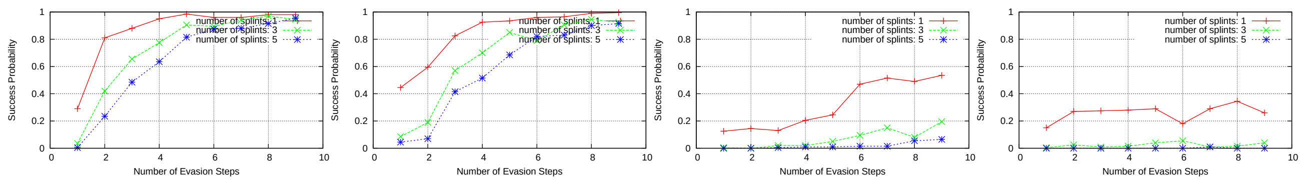
to the cooperation splints, even in case five splints are used, do not offer a lot of protection, but show a steep increase of the success probability for the adversary. However, the probability graph is less severe compared to the ordinary case, where no splints are used. The difference becomes more obvious when the number of malicious nodes is decreased. In case of more realistic 5 percent malicious nodes, 5 splints once again bestow an adversary with a success probability very close to zero. This can be considered to be quite effective as 5 percent of malicious nodes in the setup sensor network is equivalent to 180 malicious nodes, a rather large effort on the side of the adversary and with little effect when Data Splitting is used with 5 splints.



50 percent malicious nodes 30 percent malicious nodes 10 percent malicious nodes 5 percent malicious nodes

Figure 6.26: Data Splitting with UFurthest$k$ ($k = \frac{|V|}{2}$) and cooperative Malicious Nodes

Continuing the discussion a look at the graphs yielded by other choice functions reveals comparable behavior to the non-cooperative case. Figure 6.26 showing the success probability graphs for the UFurthest$k$ is almost identical to the UVicinity case.

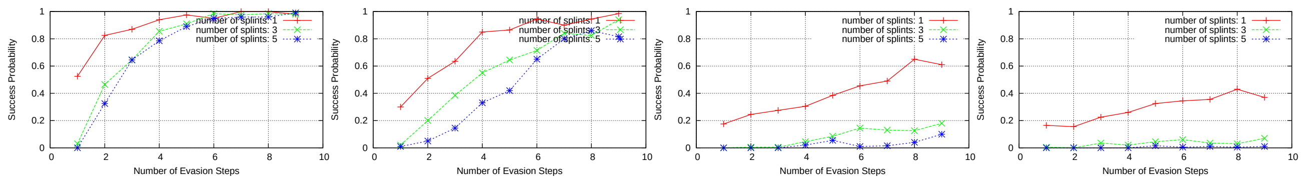


50 percent malicious nodes 30 percent malicious nodes 10 percent malicious nodes 5 percent malicious nodes

Figure 6.27: Data Splitting with GNodeFurthest ($k = 1$) and cooperative Malicious Nodes

However, looking the the GNodeFurthest choice function in figure 6.27 shows that cooperation does not change the advantage obtained by using this choice function. Again the probability graph reaches lower values than the ones obtained in the UVicinity or UFurthest$k$, but the difference is not as obvious as it has been in the case of non-cooperating malicious nodes.

Finally, the graphs induced by th DirectionV show that this choice function can exhibit similar behavior to the GNodeFurthest choice function, which again depends on the used p_c. Looking at all the graphs obtained for varying choice functions, it is clear that cooperating malicious nodes do increase the success probability and that for very high densities the advantage of using splints does not pay off, when taking the increased message

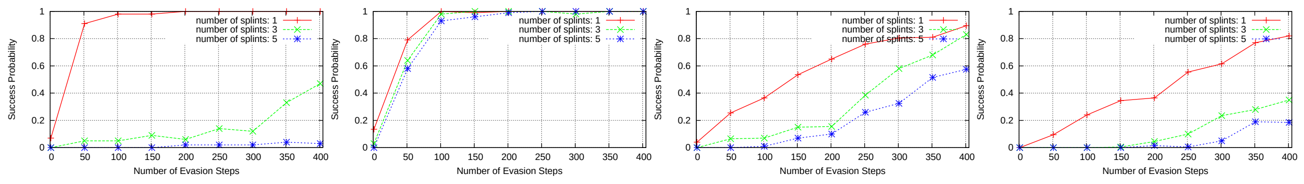


50 percent malicious nodes   30 percent malicious nodes   10 percent malicious nodes   5 percent malicious nodes

Figure 6.28: Data Splitting with DirectionV ($p_c = 0.01$) and cooperative Malicious Nodes

overhead into account that results from using this method. However, when the number of malicious nodes approaches more realistic values, which can be considered to have a generous maximum at 5 percent, the Data Splitting notion again shows a superior protection against malicious nodes, even in case those are cooperating. The main drawback, however, is that the simulation results do only look at a small number of evasion steps. Thus the rest of this section needs to address the long term behavior of the Data Splitting notion.

The question is: is Data Splitting useful over the lifetime of a sensor network. As the lifetime cannot be simulated directly, the other parameter which can be thought of being equivalent is used: the number of evasion steps. In order to investigate, the long term behavior, a larger number of evasion steps has to be looked at. Thus the simulations to follow that aim at answering the question over the long term behavior and advantages of Data Splitting are carried out for several hundred evasion steps. This is considered sufficiently, when considering actual numbers. Assuming that a data item D will rest at a node for only five minutes, the time that needs to pass until the D counts 400 evasion steps amounts to 33 hours. Which can be considered to be quite a large time period in sensor network terms. Furthermore, for the evaluation of the long term behavior more realistic settings are considered. Meaning that realistic numbers of malicious nodes that can cooperate with each other are the best indicator for the advantages that the Data Splitting notion has to offer.



5 percent non-cooperative malicious nodes   5 percent cooperative malicious nodes   0.5 percent cooperative malicious nodes   0.25 percent cooperative malicious nodes

Figure 6.29: Data Splitting Long Term Behavior

Figure 6.29 shows the graphs obtained from performing a large number of evasion steps and therefore allowing to look at the success probability of an adversary that has a long lasting commitment to extract data from a sensor network. The two leftmost graphs show five percent cases, for non-cooperative nodes as well as cooperative nodes. The five percent case of course is still not a truly realistic scenario as 180 malicious nodes cannot be considered to be achievable by any adversary. However, the figures make clear that non-cooperative nodes do not have a great chance to obtain data even after 400 evasion steps, especially when five splints or more are involved. The cooperative case however, causes the graph even for three and five splints to skyrocket towards one, just as the ordinary notion does. Hence, the question is if Data Splitting manages to overcome this obvious inefficiency in truly realistic settings of 0.5 percent (18 malicious nodes) and 0.25 (9 malicious nodes). The according figures do answer this question positively. The two rightmost figures show that Data Splitting does decrease the success probability of an adversary for an adequate number of cooperative malicious nodes. Especially, when five or more splints are used, the success probability becomes sufficiently low to justify the message overhead that is incurred by Data Splitting. Hence, the simulations have shown that Data Splitting does offer an increase of security compared to using solely Simple Evasive Data Storage and Location Bound Evasive Data Storage respectively.

6.3.4 Evasive Data Storage versus Camouflage

Now that both notions and their respective implementations can supposed to be understood to a sufficient degree, questions arise as to which of those two is actually needed, can one of them be substituted for the other or is there a preference to one notion when looking at efficiency of communication overhead. There are surely many questions that can be posed regarding those two notions and this section intends to shed light onto those by arguing qualitatively and using the simulation results of the sections already seen.

Looking from the perspective of an adversary $\mathcal{A}$ such that $\mathcal{A} \succeq \mathcal{A}(local, \star)$, i.e. one that has some traffic analysis skills and no other information is assumed to be available to him, it becomes clear that camouflage is needed in order to prevent $\mathcal{A}$ from obtaining data. This is grounded in the fact that $\mathcal{A}$ can identify communication patterns easily, thus making it possible to follow communication patterns to hot nodes and in presence of an Evasive Data Storage algorithm to wait for the respective pattern to pass through his vicinity. Hence, Evasive Data Storage without any Camouflage algorithms does not prevent a skillful traffic analysis adversary from obtaining data, but it can take some time for a local bounded $\mathcal{A}$ until data is evaded into his vicinity. Of course, adding the fact that adversary $\mathcal{A}$ knows or remembers a hot node clearly is even worse, as $\mathcal{A}$ does not even need to wait for Evasive Data Storage patterns to be observed. Hence, Evasive Data Storage without any camouflage in case of a strong traffic analysis adversary is not sufficiently powerful to provide protection against illegitimate data retrieval.

Adding Camouflage to the picture, the adversary $\mathcal{A}$ defined above can be shifted towards the $\mathcal{A}(blind, \star)$ level. Thus disabling the adversary to successfully identify patterns of message communication between two parties (see Fractal Propagation figures 6.6 and 6.7) or the patterns of an Evasive Data Storage algorithm (see Wandering Repeated Firework figure 6.12 and figure 6.19). If the Camouflage is so effective that $\mathcal{A}$ is disabled to identify anything at all, the question is if Evasive Data Storage can be omitted from the picture and still sufficient data security can be guaranteed? Here, the answer should be no. Of course, with such a strong Camouflage it is improbable that $\mathcal{A}$ will find a hot node even when he subscribes temporarily to a network, he will not be able to recognize the location of a hot node by mere traffic analysis. However, the main point is that he can still choose nodes at random and be successful with a probability of $\frac{h}{n}$, and thereafter access a hot node, if he was successful initially, over and over again with probability 1, if no Evasive Data Storage is employed. In such a case Camouflage just does not provide any protection at all and is rather a waste of precious resources.

Summarizing, the arguments above clearly identify that Camouflage is more powerful in preventing adversary $\mathcal{A}$ from accessing hot nodes, but still insufficient in several important cases. Thus in real sensor network systems the objective should be to employ both Camouflage as well as Evasive Data Storage, but with more emphasis or more devotion of resources to the Camouflage algorithms. The borderline which should be chosen between the two notions, should be mostly dependant on the application and the behavior it yields in terms of data generation and adversary assumed. Furthermore, to achieve the best result in terms of message overhead, the application and its properties should dictate how much Camouflage and Evasive Data Storage needs to be employed to guarantee appropriate levels of security. This tradeoff should be discussed in more detail in the following section.

6.4 Fundamental Tradeoffs

The preceding sections investigated properties of the Camouflage algorithms and the Evasive Data Storage algorithm, where the main objective has clearly been on the security aspects that those principles want to achieve. The other significant issue to consider is argue where the fundamental tradeoffs lie in terms of security and message complexity, which are the two parameters that should be of most concern in sensor networks.

Of course, as showed previously *both* notions, the Camouflage and the Evasive Data Storage notion, are needed to provide an adequate level of security in the majority of malicious behavior possible in sensor networks and thus not a single notion can be used to take over the other one's duties in all cases. Furthermore, excessive usage of Camouflage can cause a huge increase in the number of messages sent through the network. In the same sense, the Evasive Data Storage can account for a large number of messages if the data items in the network are displaced too frequently without any actual benefit. Thus the objective is to find just the right amount of Camouflage and just the right amount of Evasive Data Storage to secure the network with a minimal amount of additional messages.

This objective, however, is not as easy to achieve, as a significant parameter is the application that the sensor network itself is used for. In circumstances where little data is generated, the Camouflage algorithms will need to be used extensively as little communication for storing data will occur. Thus in order to prevent an adversary from pursuing messages or patterns of the storage algorithm, a lot of camouflage is needed. In the opposing case, where the network is extremely active itself, due to the application it addresses, Camouflage might be needed less, because the overall entropy of the network is already high. Similar applies to Evasive Data Storage, where in case it can be assumed that an adversary will enter the network infrequently less evasion steps are needed in a finite time interval compared to the situation, where the adversary visits frequently in order to obtain updated data. Other restrictions on how to use Evasive Data Storage heavily depend on the assumptions that can be made about the adversary, which is also the case several Evasive Data Storage algorithms have been introduced in this thesis with varying properties. In the end, each sensor network has to be investigated individually in order to obtain an accurate estimation of the need for Camouflage and Evasive Data Storage respectively.

6.5 Summary

The thesis introduced several interesting algorithms based on the two fundamental notions of Evasive Data Storage and Camouflage. Naturally, to complete the picture of those techniques a reference implementation has to be provided as well. Hence, this chapter attempted to give background information about the particular implementation and most importantly discussed simulation results regarding the performance of the Evasive Data Storage and Camouflage algorithms. The chapter provided insights into the advantages camouflage can provide when used in sensor networks, mostly by analyzing the Fractal Propagation algorithm and the local camouflage algorithms. Furthermore, the chapter also introduced the natural generalization of the Fractal Propagation algorithm and showed by using simulations and according performance measures, that were defined as well, that the generalization does provide more effective camouflage than the original one. Besides the investigation of the camouflage algorithms, the chapter also showed the importance the choice functions has in order to allow Evasive Data Storage algorithms to satisfy their respective security property. Due to the usage of the choice function is scope of Evasive Data Storage as well as Camouflage, the results obtained on the properties of choice functions, can surely find applicability in sensor networks beyond those two techniques. Furthermore, a large number of simulations have been devoted to show that the Data Splitting addition to Evasive Data Storage offers another increase in the defense against adversaries in sensor networks. Thus the chapter used simulations and the implementation successfully in showing that the Evasive Data Storage does provide a security enhancement.

Chapter 7
Outlook

This last chapter of the thesis is intended not only as an overall summary, but also as a short outlook as to what the next steps would be in further developing and analyzing the notions of Evasive Data Storage and Camouflage in the context of sensor networks. Of course, this chapter will not explain actual possibilities for more complex extensions to the Evasive Data Storage notion, as this has already been investigated in detail in the chapter of Advanced Data Oriented Techniques. The focus will be more on practical steps that are considered helpful in gaining a deeper knowledge regarding this interesting notion. Furthermore, the end of this chapter does provide a final conclusion regarding the objectives achieved by this work.

7.1 Future Work

It has been shown that several interesting techniques are available that allow to protect data in sensor networks. However, in order to increase the efficiency of the network's energy usage a more tight cooperation between the notions of Camouflage and Evasive Data Storage, that were presented in scope of this work, should be achieved. Such a close combination ought to offer all the security advantages each method inherently provides, but additionally make better usage of the communication primitives. However achieving such a combined method is extremely difficult, for once because such an efficient combination has to adapt to the application the sensor network at hand is used for. Thus making attempts to provide general solutions hard to obtain. In addition to the closer cooperation, the suggested improvement of available Camouflage Techniques, especially in form of adaptive algorithms, like the Self Adapting Pattern Imitation notion presented in the Camouflage Chapter, should offer interesting features that current approaches do not offer. Of course, an evaluation of those can be carried out, as it has been done in this work with simulative means, however a much nicer and more meaningful investigation is to use real sensor networks. Unfortunately, real sensor nodes are not yet feasibly available to be used in scope of a work like this. However, in future work the development of those kind of networks should have sufficiently progressed such that the algorithms developed in this work and future extensions can prove their security enhancement properties in real sensor networks applications too. Of course, this would also allow the testing of the more advanced concepts presented in the Advanced Data Oriented Techniques Chapter of this thesis and also their evaluation regarding effectiveness. Thus, the notion of Camouflage and especially Evasive Data Storage do offer sufficiently many options for further work in this area and can be investigated in more meaningful circumstances once sensor networks become more popular than they are today.

7.2 Conclusion

Looking back at the work, it is obvious that several interesting concepts were developed and investigated allowing for improvements of the security properties of data stored in sensor networks. The Data Security and Sensor Networks Chapter provided the necessary knowledge to work with sensor networks and be aware of the problems that those networks exhibit, especially in case of adversaries that can resided in hostile environments and the needed protection of data. Thus the chapter also contains a detailed set of adversary models that allow for a nice classification of adversaries and the tailoring of algorithms to address a specific model or class of models. The Camouflage Chapter showed currently available techniques and added new algorithms that were dubbed as Local Camouflage algorithms, such as Repeated Firework and Wandering Repeated Firework.

Furthermore, the Chapter also defined a security property that all Camouflage Algorithms need to fulfill in order to provide increased protection against adversaries in sensor networks. The subsequent Data Oriented Techniques Chapter finally presents the Evasive Data Storage notion, along with an adequate security property and several algorithms that are given in form of refinements starting with the Simple Evasive Data Storage over to the Location Bound Evasive Data Storage and ending in the Time Constraint Evasive Data Storage algorithm. This Chapter also introduces the integral part to every Evasive Data Storage algorithm, the choice function. Choice Functions are given in form of four fundamental ones, together with a combination technique that allows for an arbitrary number of choice functions to be generated from the four ones given. The Chapter on Advanced Data Oriented Techniques investigates several complex and exotic approaches to the Evasive Data Storage notion, but also provides the Data Splitting concept that proves to be an useful addition to basic Evasive Data Storage. Finally, the Simulation Chapter investigates the security properties using implementations done in scope of this work of most of the algorithms developed to show their effectiveness. Naturally, the emphasis in this chapter is on the fulfillment of the security properties. However, in scope of the evaluation of the Camouflage algorithms, a natural generalization of the Fractal Propagation algorithm is defined and proved to provide more effective camouflage algorithms compared to the original notion. The complete description of the generalization, however, is given in the Appendix. Furthermore, the main emphasis of the Simulation Chapter is placed on the evaluation of the different choice functions and the Data Splitting enhancement of the Evasive Data Storage notion. The former is identified as the most significant variable in determining the effectiveness of fulfilling the Evasiveness Security Property and moreover as an elementary concept that affects Camouflage and most likely can also be used in other contexts within the field of sensor networks. The latter, the Data Splitting enhancement, is also proved to be effective even in case long term behavior is taken into account.

Hence, the main contributions this work offers are a natural generalization of the Fractal Propagation notion, the introduction of the Evasive Data Storage notion along with different algorithms that adhere to that notion and the Data Splitting concept as a significant improvement of the Evasive Data Storage algorithms. Of course, these three major contributions are proved to increase the security of data storage when sensor networks are exposed to adversaries in hostile environments. In addition to providing those concepts several minor algorithms are developed, as the Local Camouflage Algorithms for instance, which are shown to be an adequate camouflage technique to be used with Evasive Data Storage as opposed to the Fractal Propagation notion. In the course of the thesis, another element emerged to be of fundamental importance: the Choice Function. Introduced as an integral part of the Evasive Data Storage notion, choice functions have shown to be applicable to other areas like Camouflage as well. Thus in addition to giving rise to increased security through the three results mentioned above, the contribution and investigation of different choice functions can be seen as another interesting result that is likely to extend beyond Evasive Data Storage and Camouflage.

Bibliography

[1] Jing Deng, Richard Han and Shivakant Mishra
Countermeasures Against Traffic Analysis Attacks in Wireless Sensor Networks
13th Usenix Security Symposium (2004)

[2] Jing Deng, Richard Han and Shivakant Mishra
Intrusion Tolerance and Anti-Traffic Analysis Strategies For Wireless Sensor Networks
2004

[3] D. Liu, P. Ning
Establishing pairwise keys in distributed sensor networks
In CCS'03, Washington D.C., USA, October 2003

[4] Jing Deng, Richard Han and Shivakant Mishra
INSENS: Intrusion-Tolerant Routing in Wireless Sensor Networks
2003

[5] Sylvia Ratnasamy, Brad Karp, Yin Li, Fang Yu, Deborah Estrin, Ramesh Govindan and Scott Shenker
GHT: A Geographic Hast Table for Data-Centric Storage
2002

[6] Sylvia Ratnasamy, Brad Karp, Yin Li, Fang Yu, Deborah Estrin, Ramesh Govindan and Scott Shenker
Data-Centric Storage in Sensornets
2002

[7] Zinaida Benenson, Felix Gärtner, Dogan Kesdogan
An Algorithmic Framework for Robust Access Control in Wireless Sensor Networks
In Informatik 2004, Workshop on Sensor Networks, September 2004

[8] Zinaida Benenson, Felix Gärtner, Dogan Kesdogan
User Authentication in Sensor Networks (Extended Abstract)

[9] Adrian Perrig, John Stankovic and David Wagner
Security in Wireless Sensor Networks
Communications of the ACM, June 2004/Vol. 47, No.6

[10] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler and J.D. Tygar
SPINS: Security Protocols for Sensor Networks
2001

[11] H. Ozgur Sanli, Suat Ozdemir and Hasan Cam
SRDA: Secure Reference-Based Data Aggregation Protocol for Wireless Sensor Networks

[12] Jerry Zhao, Ramesh Govindan and Deborah Estrin
Computing Aggregates for Monitoring Wireless Sensor Networks

[13] Bartosz Przydatek, Dawn Song and Adrian Perrig
SIA: Secure Information Aggregation in Sensor Networks
SenSys '03, November 2003, Los Angeles, California, USA.

[14] Andrew S. Tanenbaum, Maarten van Steen
Distributed Systems - Principles and Paradigms
2002, Prentice-Hall

[15] Rachid Guerraoui, Luis Rodrigues
Introduction to Distributed Algorithms
EPFL Technical Report, March 2004

[16] Felix Freiling
Lecture on Dependable Distributed Systems
`http://www-i4.informatik.rwth-aachen.de/lufg/teaching/ss2004/dds/index.en.html`

[17] Deva Seetharam, Sokwoo Rhee
An Efficient Pseudo Random Number Generator for Low-Power Sensor Networks
29th Annual IEEE International Conference on Local Computer Networks (LCN'04), Tampa, Florida, USA

[18] Saurabh Ganeriwal, Ram Kumar, Mani B. Srivastava
Timing-Sync Protocol for Sensor Networks
SenSys '03, November 5-7, 2003, Los Angeles, California, USA.

[19] B. Karp, H. Kung
Greedy Perimeter Stateless Routing
Proceedings of the Sixth Annual ACM/IEEE International Conference, Division of Engineering and Applied Sciences, Harvard University, 2000.

[20] Chris Karlof and David Wagner
Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures

[21] Stefan Pleisch, André Schiper
Approaches to fault-tolerant and transactional mobile agent execution - An algorithmic view
ACM Computing Surveys (CSUR), Volume 36, Issue 3, (September 2004)

[22] Murat Demirbas, Anish Arora, Mohamed Gouda
A Pursuer-Evader Game for Sensor Networks

[23] SimJava website
`http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/`

Appendix A

Additional Algorithms

In the course of this thesis several algorithms where left out, as they are not considered to particularly contribute to the understanding of the security issues and would rather clutter the respective chapter. This is especially the case of the Location Bound Evasive Data Storage Maintenance Procedures, which therefore will be given here in pseudo-code and with short descriptions, but without any formal analysis whatsoever. Furthermore, the generalization of the Fractal Propagation algorithm is presented in this appendix, as any other location would lead to an awkward sequence of the presented notions or would force the reader to jump back and forth in the work.

A.1 Location Bound Evasive Data Storage - Maintenance Procedures

The Location Bound Evasive Data Storage algorithm described in the Data Oriented Techniques Chapter was developed with the goal of improving the query and event costs, that were identified as a disadvantage of the Simple Evasive Data Storage approach. In order to do so, a restriction on the nodes that are allowed to participate in the evasion process was forced upon the Simple Evasive Data Storage algorithm. As the respective section mentioned several issues need to be addressed in case such a notion is realized, which includes accretion of members and varying maintenance features. Naturally, those can be mostly dropped, if a presetup is assumed that can handle such issues. However, in order to show that those features can be implemented, this section intends to provide according algorithms for the sake of completeness of the Location Bound Evasive Data Storage notion. As this is not directly connected to the security issues the pseudo-code algorithms given here are not evaluated throughly as the other parts of the notion were.

Algorithm A.1 shows how to realize the accretion process, i.e. the recruiting of members for the hot group. This algorithm is executed by the $\hbar$ of the hot group, which is either chosen by the base stations at startup of the sensor network or during the predeployment phase of the network. The algorithm is simple, in the sense that each node that is commanded to recruit members just picks several from its vicinity by sending the appropriate $HGID$ to its neighbors and waiting for their replies. If not sufficiently many are recruited the process is delegated with an updated number of needed members to some other node. Of course, the algorithm has to be refined in order to guarantee recruitment of d many nodes, when unreliability of channels or even malicious nodes are included into the overall picture. However, this would unnecessary increase the complexity of the algorithm and, furthermore, would be beyond the intentions of this appendix to simple provide a starting point for such features of the Location Bound Evasive Data Storage notion.

The next issue that is of importance is that each hot group needs to provide some form of group internal encryption. This is important to avoid sending a message that only members of the hot group are supposed to receive individually encrypted to each of the nodes, which would be necessary without a group wide encryption key. Thus algorithm A.2 shows a simple addition to the accretion process that is used to setup group members and incorporates the distribution of a hot group wide encryption key K_{HGID}. Such a key can be generated using the underlying encryption primitives, which are assumed to be available. The distribution starts with the head node $\hbar$ and is assumed to be continued by other nodes that are ordered to accreted members. Thus all nodes in the hot group will also dispose of such a group wide key for secure communication among group members.

Participants:	
$\hbar$	- the head node that performs the accretion process for hot group H
a	- arbitrary node in the vicinity of $\hbar$

Parameters:	
d	- the size of the designated hot group
$time\_out$	- timeout until all responses must arrive

```
ħ:   upon event recruit $d$ members for hot group $H$ do
     -     set $N = \emptyset$
     -     set $C = \emptyset$
     -     local broadcast $RECRUIT\_REQUEST(HGID(\hbar))$
     -     start timer $time\_out$
ħ:   upon event receive $CANNOT\_PARTICIPATE$ from a do
     -     $N = N \cup \{a\}$
ħ:   upon event receive $CAN\_PARTICIPATE$ from a do
     -     $C = C \cup \{a\}$
ħ:   upon event $C \cup N = V(\hbar)$ or $time\_out$ is reached do
     -     if $C > d$ then
     -        randomly choose $C' \subset C$ with $|C'| = d$
     -        local broadcast $RECRUIT\_CONFIRM(HGID(\hbar))$ to all $v \in C'$
     -        local broadcast $RECRUIT\_WITHDRAWAL(HGID(\hbar))$ to all $v \in C \setminus C'$
     -     if $d - |C'| > 0$ then
     -        randomly choose $v \in V(\hbar)$
     -        local broadcast recruit $d - |C'|$ members for hot group $H$ to $v$
a:   upon event receive $RECRUIT\_REQUEST(HGID(a'))$ do
     -     if already member of hot group with identifier $HGID$ then
     -        local broadcast $CANNOT\_PARTICIPATE$ to $a'$
     -     else
     -        store received $HGID(a'))$
     -        local broadcast $CAN\_PARTICIPATE$ to $a'$
a:   upon event receive $RECRUIT\_WITHDRAWAL(HGID(a'))$ do
     -     delete received $HGID(a'))$
```

Algorithm A.1: Location Bound Evasive Data Storage - Accretion Process

The next feature that is necessary to guarantee the working of the Location Bound Evasive Data Storage algorithm has to address one of the inherent properties that come with sensor networks, namely the frequent failure of nodes in the network. Thus means have to be provided that keep the number of hot group members constant over the lifetime of the sensor network. Algorithm A.3 is given to exactly address the failure deficiency of sensor networks and to maintain a steady membership. The algorithm's idea is simple and bases itself on the exchange of heartbeat messages HM and respective replies HMC. The head node $\hbar$ of the hot group is supposed to send heartbeat messages to all members of the group H for each time interval $time\_int$. Of course, the process of sending the heartbeat message to each of the hot group's members can be made more efficiently through a series of local broadcasts, but for the purpose of this section the described method is sufficient. Receiving such a heartbeat each hot group member is supposed to answer to $\hbar$ with a respective HMC reply. The head node waits for all replies to arrive, or, in case a node does not respond to the heartbeat, waits until the timeout $time\_out$ is reached. This timeout of course needs to satisfy $time\_out < time\_int$, in order to guarantee proper functioning of the algorithm. The next step of the head node, in case not sufficiently many nodes answer, is to recruit the missing amount of node among non-member nodes surrounding the hot group, which is done by using the mechanism provided by the accretion algorithm A.1. Such a maintenance procedure hence, does allow

```
Participants:
    $\hbar$          - the head node of a hot group
---
    $\hbar$:         upon event initialization do
                     - generate hot group key $K_{HGID}$ for group with identifier $HGID(\hbar)$
    $\hbar$:         upon event receiving $CAN\_PARTICIPATE$ from $a$
                     -     local broadcast $K_{HGID}$ to $a$
```

Algorithm A.2: Location Bound Evasive Data Storage - Hot Group Key Generation

```
Participants:
    $\hbar$          - head node of a hot group $H$
    $a$              - arbitrary node that is a member of $H$
Parameters:
    $time\_int$      - time interval for heartbeat messages generation
    $time\_out$      - time to wait for heartbeat reply
---
    $\hbar$:         foreach $time\_int$ do
                     -     set $C = \emptyset$
                     -     encrypted point-to-point $HM(CurrentTime)$ to all $a \in H$
                     -     start timer $time\_out$
    $\hbar$:         upon event receive $HMC$ from $a$ do
                     -     $C = C \cup \{a\}$
    $\hbar$:         upon event $C = H$ do
                     -     stop timer $time\_out$
    $\hbar$:         upon event $time\_out$ is reached do
                     -     set $F = H \setminus C$
                     -     randomly choose $a \in H$
                     -     encrypted point-to-point recruit $|F|$ members for hot group $H$ to $a$
---
    $a$:             upon event receive $HM(t)$ from $\hbar$ do
                     -     encrypted point-to-point $HMC$ to $\hbar$
```

Algorithm A.3: Location Bound Evasive Data Storage - Member Maintenance

for certain growth of the hot group and needs to be employed carefully in order to avoid cancer like spreading of a hot group in the network. Especially, the timeouts have to be chosen such that the time delays of the network are obeyed and possible transmission errors are also accounted for. This is however left to implementations that have specific knowledge about the communication properties of real sensor networks.

The last feature that needs to be provided for the Location Bound Evasive Data Storage algorithm in order to work effectively can be thought of an enhancement feature that addresses a crucial drawback of the notion that has not been discussed yet: energy consumption. Due to the fact, that the Simple Evasive Data Storage is used in a rather small group of nodes, i.e. the members of a hot group, those nodes will also need to participate in communication more intensively than nodes not located in hot groups. Such a scenario automatically entails that nodes in the hot groups will run out of energy faster than nodes that do not participate in any hot groups. Thus the main drawback is that hot group members are subject to greater energy consumption than other nodes. To solve this problem a simple relocation of the complete hot group can be introduced into the picture. Hence, instead of making all hot groups take static positions in the sensor network, the hot groups are allowed to wander in the network. This can be nicely imagined as a swarm of nodes that slowly progresses through the network. An algorithm that can provide such a rather optional feature is an adaptation of the accreation process, which needs to dismiss hot group members and recruit new ones. Of course, this process needs to be controlled by some adequate time interval. Furthermore, other issues like the regeneration of the hot group key and its distribution as well as the update of the global information concerning the head node, which will wander along with the hot group, needs to be provided. The detailed development of such an algorithm, however is left as future work and will not be given here in pseudo-code, but should be kept in mind as an important feature for Location Bound Evasive Data Storage in order to elongate the overall lifetime and effectiveness of the sensor network.

A.2 Generalization of the Fractal Propagation Global Camouflage

Here the generalization of the Fractal Propagation algorithm that was provided in pseudo-code in algorithm 3.5 is presented. This generalization has not been included in the Camouflage chapter itself for two reasons: the first one is that the chapter intended to introduce the concept as it was introduced in the original work [1] and the second reason is that the generalization is obtained by usage of the concept of a *Choice Function*, which is originally developed in the scope of the Evasive Data Storage notion. Thus in order to provide a meaningful order of the presented concepts, the pseudo-code is introduced here in the appendix, but of course was referred to and evaluated in the preceding Simulation Chapter. That chapter already pointed out that the generalization is achieved by introducing the concept of a choice function into the workings of the Fractal Propagation algorithm and showed that choosing the right function can yield better properties regarding the control of the fake packets paths in the network, which naturally can be of advantage.

Participants:

s	- current node
v	- arbitrary node in the vicinity of s
v'	- arbitrary node in the vicinity of s or v

```
s:    upon event overhearing a message $msg$ forwarded by some $v \in V(s)$ do
  -       with probability $p_c$ do
  -           randomly choose a node $v \in V(s)$
  -           randomly generate bits of length $fake\_size$ and store as $fake\_package$
  -           point-to-point message $(fake\_package, K)$ to node $v$
v:    upon event receiving a fake message $(fake\_package, depth)$ do
  -       if $depth > 0$ and $depth < K$ then
  -           with probability $p_f$ do
  -               set $v'' = Choice(V(v))$
  -               decrement $depth$
  -               point-to-point message $(fake\_package, depth)$ to $v''$
v':   upon event overhearing a fake message $(fake\_package, depth)$ do
  -       if $depth > 0$ and $depth < K$ then
  -           with probability $p_t$ do
  -               randomly choose a node $v''' \in V(v')$
  -               decrement $depth$
  -               point-to-point message $(fake\_package, depth)$ to $v'''$
```

Algorithm A.4: Fractal Propagation Generalization

The pseudo-code for the generalization of the Fractal Propagation algorithm is given in algorithm A.4 and yields a family of Fractal Propagation that are parameterized by the actual choice function *Choice* used in A.4. The parameters, besides the Choice Function parameter, are identical to the ones of the original Fractal Propagation notion and are therefore not repeated. The main different between the original algorithm and the generalization is that during the forwarding of the fake packets, the choice where to forward the fake packet to is not done arbitrary, as in the original algorithm, but is guided by the respective choice function. This simple modification of a single line, incurs significantly different behavior and allows the results that were obtained for choice functions and their properties to be used for this generalization as well. For the effectiveness of this generalization refer to the Simulation Chapter where improved behavior has been proven for the case of the GNodeFurthest choice function by using a distance measure. Such a generalization of the Fractal Propagation notion that was originally conceived in [1] and restated in algorithm 3.5 makes the original algorithm become a mere special case of the general algorithm A.4 given here, namely for the case that the used choice function is **UVicinity**. Naturally, the usable choice functions are not restricted to the ones presented in scope of the Evasive Data Storage notion nor to their combinations, but can also be designed specifically to enhance Camouflage even further. However, finding even more suitable choice functions for this generalization is left as future work and the obvious advantage obtained by the GNodeFurthest function is suffiecient for the purpose of this work to show superiority of this generalization over the general algorithm.

Appendix B

Source Code for Simulations

The source code for the implementation of the Camouflage and Evasive Data Storage algorithms used in the Simulation Chapter can be found on the DVD medium that is included in this thesis. The discrete event simulator can be downloaded from the website mentioned in the references, and is therefore not included on the medium. The autonomous implementation used for the simulations of the choice functions can be run on all plattforms that support the Microsoft .NET Framework or The Mono Project respectively. Both of those environments can be downloaded from the internet and are thence not included on the provided medium. The DVD contains further information as to how the implementation can be compiled and started in the respective environment. Thus for more detail refer to the documentation on the medium.