Spring 4-16-2012

DI-SEC: Distributed Security Framework for Heterogeneous Wireless Sensor Networks

Marco Valero
Georgia State University

DI-SEC: *DI*STRIBUTED *SEC*URITY FRAMEWORK FOR HETEROGENEOUS WIRELESS SENSOR NETWORKS

by

MARCO VALERO

Under the Direction of Dr. Raheem Beyah and Dr. Yingshu Li

ABSTRACT

Wireless Sensor Networks (WSNs) are deployed for monitoring in a range of critical domains (e.g., health care, military, critical infrastructure). Accordingly, these WSNs should be resilient to attacks. The current approach to defending against malicious threats is to develop and deploy a specific defense mechanism for a specific attack. However, the problem with this traditional approach to defending sensor networks is that the solution for one attack (i.e., Jamming attack) does not defend against other attacks (e.g., Sybil and Selective

Forwarding). In reality, one cannot know a priori what type of attack an adversary will launch. This work addresses the challenges with the traditional approach to securing sensor networks and presents a comprehensive framework, *Di-Sec*, that can defend against all known and forthcoming attacks. At the heart of Di-Sec lies the monitoring core (*M-Core*), which is an extensible and lightweight layer that gathers information and statistics relevant for creating defense modules. The M-Core allows for the monitoring of both internal and external threats simultaneously supporting the execution of new or existing detection and defense mechanisms against different threats in parallel. Along with Di-Sec, a new user-friendly domain-specific language was developed, the M-Core Control Language (*MCL*). Using the MCL, a user can implement new defense mechanisms without the overhead of learning the details of the underlying software architecture (i.e., TinyOS, Di-Sec). Hence, the MCL expedites the development of sensor defense mechanisms by significantly simplifying the coding process for developers. The Di-Sec framework has been implemented and tested on real sensors to evaluate its feasibility and performance. Our evaluation of memory, communication, and sensing components shows that Di-Sec is feasible on today's resource-limited sensors and has a nominal overhead. Furthermore, we illustrate the functionality of Di-Sec by implementing and simultaneously executing detection and defense mechanisms for attacks at various layers of the communication stack (i.e., Jamming, Selective Forwarding, Sybil, and Internal attacks). We ran all our experiment on a cluster of real sensors and showed the functionality of the cluster heads running Di-Sec.

INDEX WORDS: Di-Sec, Security framework, Monitoring core, Wireless sensor networks

DI-SEC: *DI*STRIBUTED *SEC*URITY FRAMEWORK FOR HETEROGENEOUS WIRELESS SENSOR NETWORKS

by

MARCO VALERO

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2012

DI-SEC: *DI*STRIBUTED *SEC*URITY FRAMEWORK FOR HETEROGENEOUS WIRELESS SENSOR NETWORKS

by

MARCO VALERO

Committee Chair: Dr. Raheem Beyah

Committee: Dr. Yingshu Li

Dr. WenZhan Song

Dr. Anu Bourgeois

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2012

DEDICATION

This dissertation is dedicated to my parents, Antonio and Sonia Valero.

ACKNOWLEDGEMENTS

This dissertation work would not have been possible without the support of many people. I want to express my gratitude to my advisor Dr. Raheem Beyah who offered me invaluable support and guidance for the past five years. He has been my role model and mentor. I also want to thank Dr. Anu Bourgeois, Dr. Yingshu Li, and Dr. Raj Sunderraman for all the motivation and great advice during my Ph.D.

I want to express my gratitude to the Communication Assurance and Performance Group (CAP) for all the support and ideas. I also extend my gratitude to Dr. WenZhan Song, Shaochieh Ou, Tammie Dudley, Adrienne Martin, Celena Pittman, Venette Rice, and my colleagues Nick, Sangsin, Marwan, Juan Diego, Walid, Alejandra, Dinesh, Adrian, Serghei, Basam, Thamer, Yang, Jing, Shouling, Mingsen, Debraj, Abinashi, Kebina, Mackenzie, and Selcuk.

Finally, I want to thank my family, especially my parents Sonia and Antonio, my sister Maria, Jose, Santiago, and Samuel.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

- WSN - Wireless Sensor Network
- CH - Cluster Head
- BS - Base Station
- M-Core - Monitoring Core
- COMM - Communication Module
- SENSE - Sensing Module
- INTERNAL - Internal Module
- DDM - Defense and Detection Module
- MCL - M-Core Control Language
- IDS - Intrusion Detection System
- MAC - Media Access Control

PART 1

INTRODUCTION

1.1 Problem and Motivation

Wireless Sensor Networks (WSNs) are no longer a nascent technology and today, they are actively deployed as a viable technology in many diverse application domains such as health care, military, and environmental. Moreover, with recent initiatives such as *Cyber-Physical Systems* [1], *Internet of Things* [2], and *Planetary Skin* [3], sensor-based applications have gained a new impetus in the research community. WSNs have been predicted to be one of the ten technologies that will change the world in the next 10 years [4]. More companies offer sensor-based solutions and the usage of billions of networked sensors are envisioned to be deployed on land, sea, air, and space to detect and predict the environmental changes in an effort to build a globally pervasive nervous system composed of these tiny devices [3].

Over the last decade, the WSNs research community has identified many unique security threats. There has been a tremendous effort in building defense mechanisms against these threats and a myriad of security solutions have been introduced in the literature. Nonetheless, the trend with different security schemes so far has been to focus on defending against individual threats/attacks rather than a comprehensive security solution. We observe several legitimate reasons for this trend. First, sensors are limited in terms of energy, memory, and computational resources and this situation poses unique challenges for protocol builders. Second, sensors were initially considered to be deployed for single-task applications; thus, the threat models envision the protection against only single attacks. Third, the sensor research was evolving and the sensor software and hardware platforms were not as rich and mature as today.

However, in reality, the traditional method of defending against only a certain attack does not eliminate the risk of other attacks. For instance, the solution for the Jamming attack

does not defend against other possible attacks (e.g., Sybil, Selective Forwarding). Armed with only one defense mechanism, sensor nodes and WSNs are unable to defend against attacks other than what the current defense mechanism can defend against. Albeit being useful in theory, this is far from the truth because in this traditional approach to securing WSNs one must unrealistically assume that the attacker will only employ the attack for which the network is prepared to defend. In fact, one cannot know a priori what type of attack an adversary will launch. Given the multifaceted threats on today's networks, it is very probable that one or more attackers launch single or multiple attacks simultaneously at different places in the network and employ different techniques to target different nodes and functionalities. Keeping the realistic threat model in mind, WSNs must be prepared to defend against all known attacks at any given time [5]. Additionally, the intuitive idea of combining the existing schemes will not suffice given the resource constraints of current inexpensive sensors. For example, if we consider the memory capacity of a sensor node (e.g., Mica2 mote with 4 KB RAM and 128 KB program memory), it is not possible to store mechanisms that detect and analyze many attacks, and therefore prevent the nodes/network from a security breach. For instance, [6] requires 2.2 KB RAM to defend against DoS attacks, [7] needs 1.5 KB RAM to defend against sinkhole attacks, and [8] requires approximately 1 KB RAM to defend against hello flood attacks. If we consider program memory, then 60KB is required for the operating system (e.g., TinyOS [9]), 45.26 KB to store a code dissemination tool such as [6], and 7.2KB (approx.) to provide link layer security [10], which consumes 88% of the available program memory while still leaving the node vulnerable to many attacks.

1.2 Proposed Solution

In this work, we present a comprehensive security framework, *Di-Sec*, that can defend against all known threats for WSNs. To the best of our knowledge, there is not a solution that can defend against all known attacks in realistic situations. Although the previous security mechanisms are well established for each individual layer of the communication stack

or individual attack, **combining all of the mechanisms and making them work in collaboration is a challenging research problem** [11]. Therefore, in this work, we propose a framework that can provide generic security to WSNs using real sensors. Moreover, motivated by the future applications of sensors and the growing interest [2] to integrate these resource limited devices with more powerful infrastructures, Di-Sec provides an architecture for heterogeneous sensor networks where there is a combination of high-end sensors along with low-end sensors to define a general framework for security. The approach is also beneficial because providing defenses for all known attacks at different layers would not be possible with the low-end sensor nodes memory and other constraints, and using only high-end sensor nodes (base stations and cluster-heads) introduces high deployment costs.

The Di-Sec framework runs on TinyOS. TinyOS is a modular operating system based on components that are wired together through interfaces to create applications with different functionalities. Following this operating system characteristic, we designed the Di-Sec framework with a highly modular architecture where every component is independent, and can be easily added and removed without affecting the rest of the framework.

In order to create a comprehensive security solution we analyze the functionality of WSN devices and the variety and nature of WSN attacks. Three important functions of sensor devices include sensing physical or environmental conditions, processing collected data, and communicating with other sensors. All of these three functions are potential attack vectors and should be continuously monitored. The sensing device itself can be used to trigger attacks on the sensor. For instance, a motion pattern, a change of temperature, or even a flashing light could be used to activate a trojan on the sensor. The collected data should also be monitored since it might be used to inject some malicious code or data that can compromised the sensor. The sensor communication function is the main target of attacks. Given the broadcast nature of the wireless medium used by sensors to communicate, it is very attractive and easy for adversaries to launch attacks against communication channels.

For the design of the Di-Sec framework we also considered the different types of attacks: 1) Active attacks including packet modification, injection and replaying, 2) Passive attacks

or Eavesdropping, 3) External attacks, and 4) Internal attacks which are the more dangerous attacks. All of these attacks are also classified at different layers of the protocol stack, for example Jamming attacks (at the physical layer), Sybil attacks (at the MAC layer), Wormhole attacks, and Selective Forwarding (at the network layer). Therefore, our framework should be able to defend at all the layers of the protocol stack.

The novel architecture of Di-Sec includes three fundamental components: the Monitoring-Core (M-Core), the reporting control (R-Control), and the Detection and Defense Modules (DDMs). Conceptually, the M-Core is the heart of Di-Sec for individual node activities monitoring. It is an extensible and lightweight layer responsible for gathering specific statistics to support the operations of the DDMs. The M-Core is a simple yet effective novel component-based solution for monitoring of both internal and external threats. It supports the execution of new or existing detection and defense mechanisms against different threats in parallel. On the other hand, the R-Control is use for network control and communication with the cluster head.

The DDMs are specific attack detections and/or defense mechanisms, but *can also be used as conduits to provide services to other layers*. Each DDM would include the implementation of the necessary behavior to detect and defend against attacks utilizing the M-Core services. Furthermore, to easily use the Di-Sec framework to access the services provided by M-Core, we have created a new domain specific language named M-Core Control Language (*MCL*). Using MCL, a user can implement new DDMs and M-Core services without the overhead of learning the details of the underlying software architecture (i.e., TinyOS, Di-Sec). Hence, MCL expedites the development of sensor defense mechanisms by significantly simplifying the coding process for developers.

We have implemented the Di-Sec framework and tested it on real sensors to evaluate its feasibility and performance. Our evaluation of memory, communication, and computation shows that Di-Sec is feasible on today's resource-limited sensors and has a nominal overhead. Furthermore, the comprehensive architecture of Di-Sec framework allowed us to implement four detection and defense mechanisms that span different layers of the sensor communication

stack (i.e., Jamming, Sybil, Selective Forwarding, and Internal attacks). To the best of our knowledge, this is also the first implementation of defenses against these four prevalent WSN threats at the same time on sensors.

In this work, we assume a realistic scenario in which the network is divided into clusters. In each cluster, all the sensors (regular nodes) are running the Di-Sec software as an invisible layer, and one gateway node is present and serves as the cluster head. The regular nodes can run any application layer program together with Di-Sec. Usually, they will be collecting data from the physical environment and sending it back to the cluster head in a multi-hop fashion.

The cluster heads running Di-Sec, contain a large database of program images with DDMs to defend against various attacks, which are used to reprogram the regular nodes according to detected attacks and required defenses. This set of DDMs can be updated and upgraded since we assume that in the future more sophisticated attacks like buffer overflow on sensors [12] are possible, and therefore, new detection and defense techniques will be developed.

Providing defense against all known as well as future attacks requires regular nodes to have the defense mechanisms for all these attacks in memory. However, because of memory constraints, regular nodes cannot contain all DDMs, only a subset of these mechanisms can be stored in the local memory of these devices. Based on this subset, the regular nodes will be capable of detecting and defending against some attacks. The cluster head is responsible for monitoring the network, detects and defends against other attacks by updating the sensors with a new program image containing the required DDMs. One way to do this is by using wireless network reprogramming (remotely reprogramming sensor nodes through wireless links after they are deployed). The Di-Sec framework uses the Secure and Link-Quality Cognizant Image Distribution (SIMAGE) [13] tool for code dissemination and sensor reprogramming to update the set of DDMs in the cluster. Note that to add a new DDM, it might be necessary to delete another DDM from the node's subset due to its limited memory capacity. How to decide the DDMs subset for regular nodes in the same cluster is

an optimization problem. After detecting an attack in its own cluster, a cluster head also propagates a warning to other cluster heads. After receiving the warning information, the cluster heads evaluate the likelihood of each attack and choose a new subset of DDMs for regular nodes in their cluster.

The goal of the Di-Sec framework is to realize an architecture that can be leveraged by researchers to expedite the development of sensor defense mechanisms and to allow their parallel execution. We envision a platform that is community driven, similar to open-source network simulators (e.g., NS-3). The Di-Sec framework provides a realistic environment and a complete security solution where the resources of high-end devices (base stations and cluster heads) are available to enhance the functionality of the framework.

The main contributions of this work are the following: (1) Realizing an extensible architecture that can rapidly allow the implementation and execution of multiple attack defense and detection mechanisms simultaneously; (2) presenting a new language to significantly simplify the development of new defense mechanisms; (3) illustrating scenarios for single and multiple simultaneous attacks and how Di-Sec can host multiple defense mechanisms to stop the attacks. The code and more information about the Di-Sec are available at [14].

1.3 System Overview

To better illustrate our proposed solution, in this subsection we explain the general idea of the Di-Sec framework with a collection of figures and define our network and threat model.

Figure 1.1 presents the main components of our system. The Di-Sec framework is an invisible layer that provides a variety of services for node and network control and monitoring, and facilitates the creation of DDMs. As shown in figure 1.2, Di-Sec supports and runs multiple DDMs simultaneously. Figure 1.3 shows a cluster of sensors running Di-Sec and a cluster head, which is in charge of monitoring the cluster.

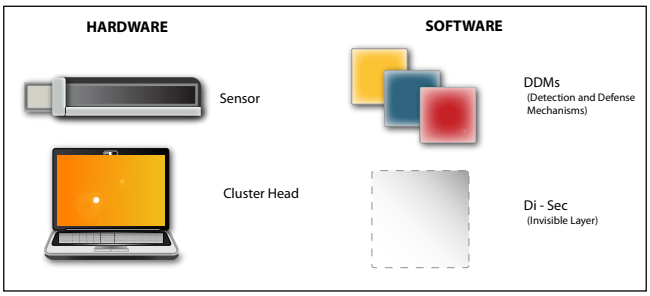

Figure 1.1 Network Components.

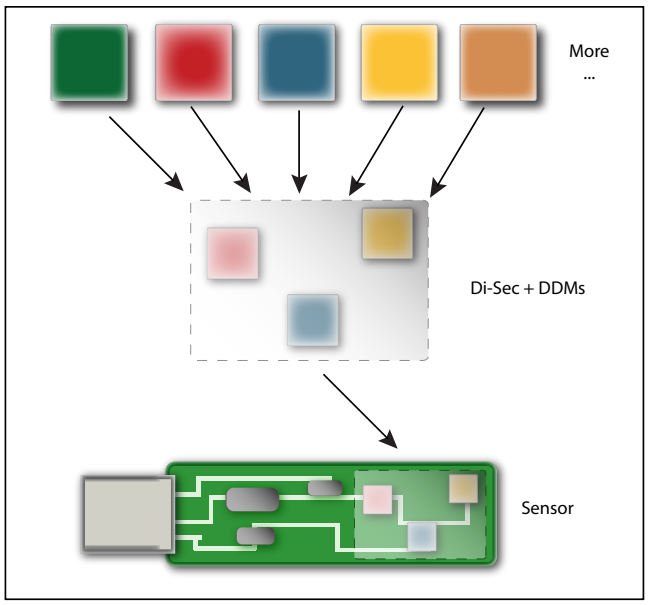

Figure 1.2 Di-Sec + DDMs in a Sensor.

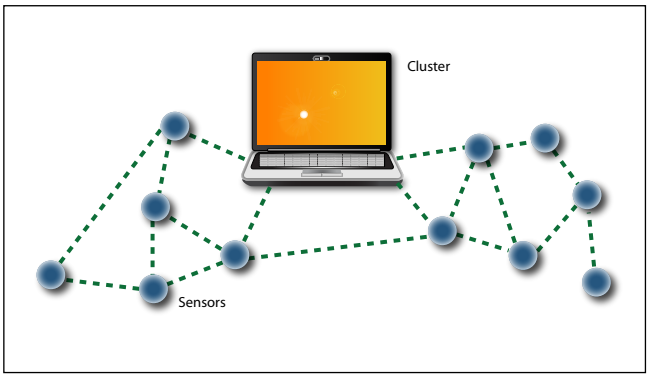

Figure 1.3 Sensor Cluster.

1.3.1 Network Model

We consider heterogeneous WSNs in this work, where there are two kinds of nodes, regular nodes and cluster heads (CHs). Regular nodes have limited energy, poor computation ability, short sensing, and small transmission ranges while CHs have plentiful resources including more energy, a larger memory size, stronger communication ability, and more powerful computation ability. In our model, we have a network represented as an undirected graph $G = (V, E)$, where each edge $(u, v) \in E$ represents a communication link between nodes u and v, and each sensor $v \in V$ collects data from one of its sensing components and forwards the values through one or multiple hops to the cluster head for further processing, analysis, and storage. The CHs are more expensive than the regular nodes. As a result, fewer CHs are deployed in a WSN. Usually, these CHs are responsible for complex computations to increase the cluster processing capabilities and prolong the network lifetime. The network is divided intro cluster and each cluster head is also a gateway node that can communicate with outside networks. A regular sensor joins the nearest cluster and communicates with the cluster head over multiple hops. Figure 1.4 shows a network with multiple cluster, which represents a realistic scenario of WSN deployments.

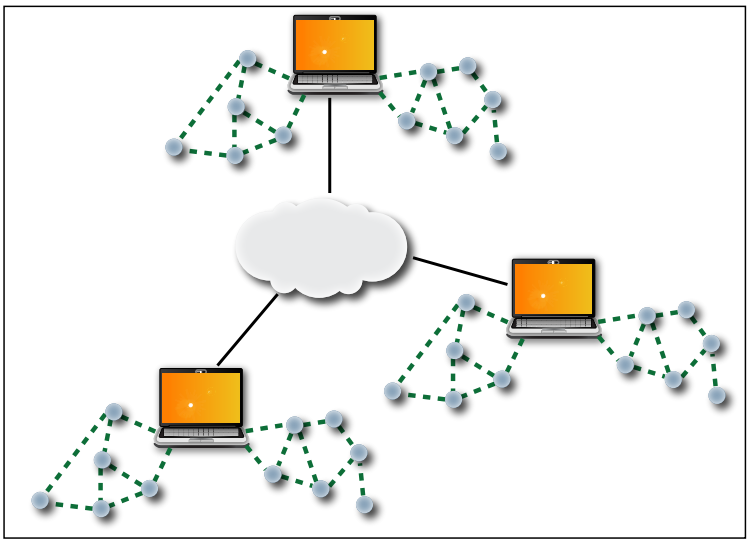

Figure 1.4 Network with Multiple Clusters.

When running the Di-Sec framework, the gateway nodes have a database component in memory which maintains the records of various details regarding the previous threats/attacks as well as possible oncoming attacks and their respective detection and defense schemes (DDMs). Once an attack is detected in the cluster, the cluster head notifies its neighbor clusters so that the other cluster can prepare themselves for possible attacks. Figure 1.5 shows the warning notification process once and attacker is identified, and figure 1.6 shows the clusters after updating their DDM set.

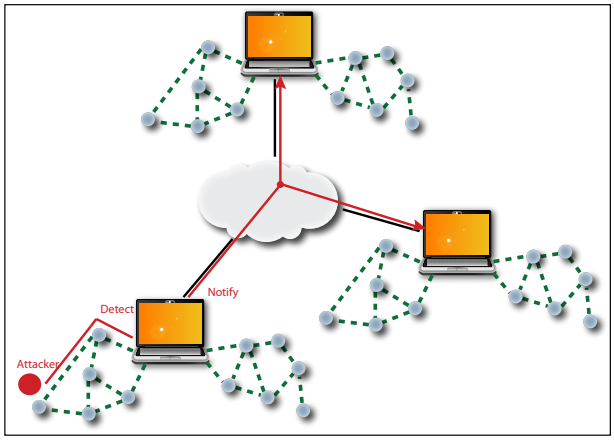

Figure 1.5 One Cluster Under Attack.

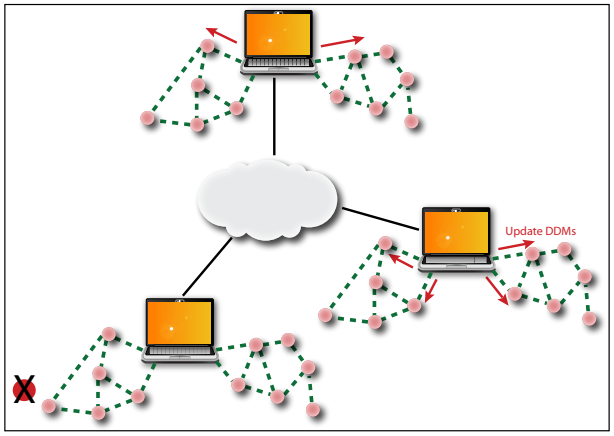

Figure 1.6 All Cluster Updating their DDMs Set.

1.3.2 Threat Model and Assumptions

We assume that is possible to have one or more attackers in the network. The malicious nodes are structurally the same as the regular nodes, and possess hardware capabilities either similar to or higher than that of legitimate nodes. An attacker can launch multiple attacks on the cluster and also may change his position to target other regions of the cluster. An example of an heterogeneous network with multiple attackers is shown in Figure 1.7.

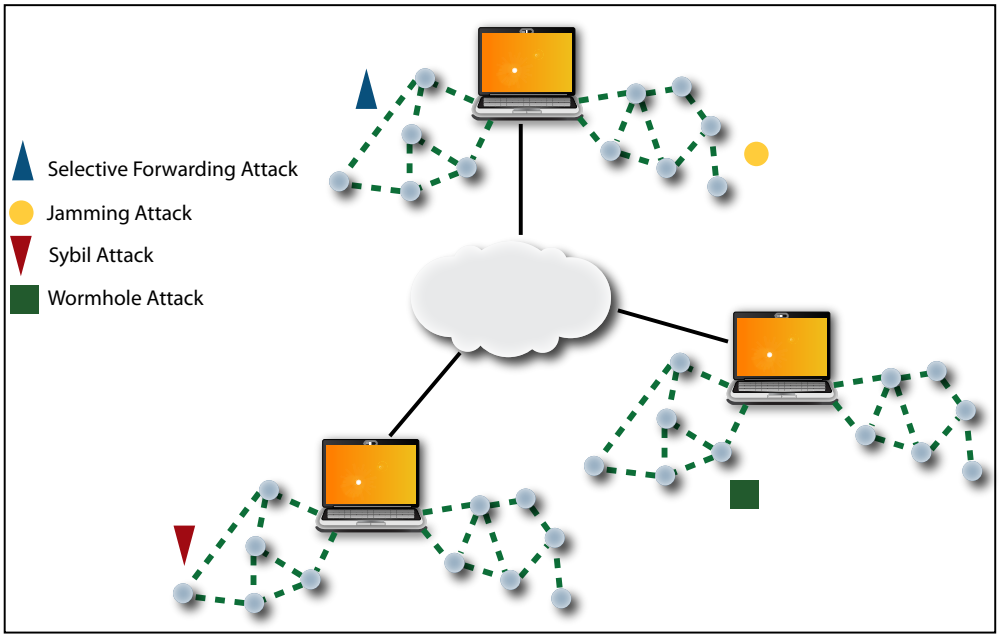

Figure 1.7 Heterogeneous Network with Various Attackers.

PART 2

LITERATURE REVIEW

The issue of providing security to WSNs is a significant and open research problem which has been discussed extensively in earlier studies. When designing a generic security framework for WSNs, it is imperative to sift through the relevant literature; hence, in this chapter, we list several related works from the literature. We classify the relevant literature under two categories. (1) General knowledge that include surveys, security analyses, attacks definitions, and defense techniques, and (2) Existing security solutions.

2.1 General Knowledge

The majority of sensor security related works can be further divided into three groups: (a) general security overviews, (b) works associated with one or more layers of the protocol stack, and (c) cryptographic and key management works.

(a) General security overviews:

In [15], the authors present a WSNs security survey where they summarize defense methods based on the networking protocol layer and present an analysis of the advantages and disadvantages of current secure schemes. In this work, the authors give a holistic overview of the security issues divided into seven categories including cryptography, key management, attacks detection and prevention, secure routing and others. It is relevant to briefly discuss some of those categories:

Cryptography: Cryptography is fundamental to providing security services in WSNs as well as in most ad hoc networks. Many researchers consider that public key cryptography is too expensive for WSNs in terms of computation and energy costs. However, some recent research results show that it is feasible to apply public key cryptography by using lightweight algorithms. The use of symmetric key on the other hand, is a more popular and feasible

solution because of its low cost even though the key management is still a drawback.

Key management: Is an important component of the cryptographic solution. Most proposals use a key-predistribution technique to facilitate the key management. Some of the recommendations for key management are: 1) To use symmetric cryptography, 2) To predistribute the keys before deployment, 3) To use master keys with small number of key-seeds to reduce computation complexity, 4) To use location information or deployment knowledge in key management to improve the performance, and 5) To integrate the node identity in the process of key generation to make the systems more secure.

Attacks detection and prevention: Attack detection techniques are either centralized approaches or distributed cooperative approaches. Centralized mechanisms gather the data from monitoring nodes and compare it with the data from other neighbor nodes. Based on this comparison, the system can make a decision whether a given node is under attack or not. In the distributed cooperative approaches, all neighbors of a given node make collective decisions to detect attacks. The disadvantage of the first approach is that it introduces more routing traffic from the network to the base station while the disadvantage of the second approach is that it introduces more computation requirements for monitoring neighbor nodes and taking decisions.

Secure routing: Another important component for the security of the network is a secure routing. There are many secure routing algorithms for WSNs proposed in the literature. Some of them are reputation based schemes that rely on neighbor nodes cooperation, and other works propose the use of multi-path techniques. Although there exist many secure routing protocols, the design of new algorithms is still open to research.

(b) Security associated with layers of the protocol stack:

Other research in the literature focus on attacks and detection at different layers of the protocol stack, like the work in [16] where the authors presented the Denial-of-service (DoS) attacks and defenses at different layers, which is summarized in table 2.1.

The work in [17] also focusses on the prevention of DoS attacks using watchdog and reputation schemes to identify misbehaving nodes. For instance, a reputation variable of a

Table 2.1 DoS Attacks and Defenses at Different Layers of the Protocol Stack

Layer	Attacks	Deffenses
Physical	Jamming	Spread-spectrum, priority messages, lower duty cycle, region mapping, mode change
	Tampering	Tamper-proofing, hiding
MAC	Collision	Error-correcting code
	Exhaustion	Rate limitation
	Unfairness	Rate limitation
Network	Neglect and greed	Redundancy, probing
	Homing	Encryption
	Misdirection	Egress filtering, authorization, monitoring
	Black holes	Authorization, monitoring, redundancy
Transport	Flooding	Client puzzles
	Desynchronization	Authentication

node A from the point of view of a node B should be constantly updated based on various parameters to decide whether node A is a misbehaving node or not.

Some other works in the literature are more specific and address particular attacks, like [18], where the authors presented a survey with the different Jamming techniques that include: Spot Jamming, Sweep Jamming, Barrage Jamming, and Deceptive Jamming. The authors also provided a thorough lists and comparison of the different anti-jamming approaches.

The work in [19] is very similar to [16] and also analyzes the attacks on the network communication stack. This work provides a taxonomy of attacks on sensor network and outline possible solutions for each attack. For instance, at the physical layer, some defenses against Jamming attack include frequency hopping and code spreading, and at the link layer the attacks can be detected by using collision detection techniques, reducing the rate of packet requests, or using smaller frames for each packet.

Some analyses on MAC layer attacks are introduced in [20], where the authors present a good overview of Sybil attacks and divide the attacks into the following six categories:

Direct Communication: A Sybil nodes communicate directly with legitimate nodes.

Indirect Communication: No legitimate nodes are able to communicate directly with

the Sybil nodes. Therefore, one malicious node claims to be able to reach the Sybil nodes.

Fabricated Identities: The Sybil nodes create arbitrary new identities.

Stolen Identities: The attacker uses a legitimate ID as his own.

Simultaneous: The attacker use multiple Sybil identities at the same time.

Non-Simultaneous: The attacker alternately use a large number of identities over a period of time.

Some attacks at the network layer are defined in [21] including: replayed routing information, Wormhole, acknowledgement spoofing, and Selective Forwarding. In [21] the authors also introduced two new attacks: Sinkhole attacks and Hello Floods. Since all of these attacks can target the majority of the routing protocols in sensor networks, the paper also discusses the different defense solutions.

(c) Cryptographic and key management

Some other research works related to cryptography and key management include [22], [17] and [23]. According to [22], sensor networks pose unique challenges, and traditional security techniques used in traditional networks cannot be applied directly because of sensor devices limited energy, computation, and communication capabilities. Also, sensor nodes are generally deployed in accessible areas increasing the risk of physical attack. In this work the authors cover several security challenges, including key establishment, secrecy, authentication, and privacy. The proposed solution to the challenges is the use of cryptography, and the simplest solution proposed for key establishment is a network wide shared key.

In [17], part of the work focusses on key management, which the authors say is an unsolved problem in ad hoc sensor network due to sensors limited resources. Their conclusion is that key management should adopt a "local-updated and global distributed algorithm and should be combined with topology management". Finally, the work in [23] focusses on designing sensor networks that are at the same time connected (with high probability) and provably secure. The authors introduced a mathematical analysis to show connectivity via secure links and resilience against malicious attacks by using random pre-distributed keys.

2.2 Existing Security Solutions

2.2.1 802.15.4

The 802.15.4 standard [24] provides link layer security using three different modes of operation: unsecured mode, access control list (ACL) mode, and secured mode. In unsecured mode, no security is provided. In ACL mode all frames are sent without security but the frame originator can be checked against an ACL table. Therefore, it is easy to spoof the source address to bypass this security service. The secured mode allows a choice of security suites to be applied to all frames. These suites include: 1) AES-CTR: where all the data is encrypted using a defined 128-bit key and the AES algorithm. 2) AES-CBC-MAC: where a Message Authenticity Code (MAC) is attached to the end of the data payload. 3) AES-CCM: which is the mixture of the previous two methods.

One of the main problems of the 802.15.4 security solution is that not all the features are actually implemented by the chip manufacturers or the developers in case of software implementation. Therefore, not all the devices using the 802.15.4 have full access to all the security functionalities including access control lists.

Although Di-Sec runs on TinyOS and uses the 802.15.4 as its communication standard, we are not using any of the security solutions provided by the standard. The matching characteristic of Di-Sec and the 802.15.4 security is that Di-Sec also provides encryption and ACLs.

2.2.2 TinySec

TinySec [10] is another link layer security architecture for WSNs, easy to use and transparent to applications. Similar to Di-Sec it is implemented for the TinyOS operating system. It uses a single shared global cryptographic key to provide link layer encryption and integrity protection. Its cryptography is based on block cipher and provides two modes of operation: authenticated encryption (TinySec-AE) and authentication only (TinySec-Auth). Its principal security features are access control, integrity, and confidentiality.

In contrast to our work, TinySec is not prepare to defend against other attacks like Jamming attacks or Internal attacks. Its packet overhead is 5 bytes, which is one byte smaller than the Di-Sec packet overhead.

2.2.3 MiniSec

MiniSec [25] is a secure sensor network communication protocol that provides data secrecy, authentication, replay protection and freshness with lower energy consumption and better security than TinySec. It uses offset codebook (OCB) mode as its block cipher mode of operation, which allows the ciphertext to be the same length as the plaintext. Another strong characteristic of MiniSec is the replay protection without the transmission overhead of sending a large counter with each packet. MiniSec has two modes of operation: MiniSec-U for unicast packets and MiniSec-B for broadcast packets as explained in [10].

Similar to TinySec, MiniSec does not provide security against other attacks like DoS attacks.

2.2.4 SecureSense

SecureSense [26] is another link-layer security solution designed to provide energy efficient communication in WSNs. It proposes the use of a security broker that enables a sensor node to dynamically modify its security controls and optimally allocate the resources required for the security services (CPU cycles, memory consumption and RF messages) depending on the observed external variables (e.g., environment), internal constraints and application requirements.

This work also provides secure communications but in the same way that the previous solutions, it does not defend the network from other attacks.

2.2.5 SPINS

A set of secure protocols for sensor networks, SPINS, is given in [27]. It describes two important secure building blocks: secure network encryption protocol (SNEP) and micro,

timed, streaming, loss-tolerant authentication protocol (μTESLA), where SNEP gives the baseline security primitive (i.e., data confidentiality, two-party data authentication, and data freshness) whereas μTESLA is a protocol which provides authenticated broadcast. Although these security protocols detect and correct some classes of abnormal node behavior, they do not consider all scenarios of malicious activity that a node is susceptible to. For example, DoS attack.

2.2.6 AMSecure

AMSecure [28] is a link layer security suite which provides message confidentiality, authentication, integrity, replay protection and semantic security [29]. AMSecure was designed to interact with the CC2420 radio chip and implemented in TinyOS. It uses the security features of the Texas Instruments CC2420 radio chip in order to provide all of its security services. An interface is provided to allow security aware applications to manage the keys being used [29]. For the secure communication of Di-Sec, we use the *SecAMSenderC* component provided with TinyOS-2.x. This component is similar to the AMSecure and also uses the security features of the CC2420 chip.

As we can see, some studies provide classifications and address the relevant issues from a general perspective [15], [16], [17], [19], [21], [22], [23], [29]. Some others focus only on a particular layer of protocols [18], [20], [21], [30] identifying various common attacks like Jamming (physical layer), Sybil (MAC layer), Selective Forwarding (network layer). The common drawback with earlier security schemes is the fact they were designed to defend against only individual threats/attacks rather than a comprehensive security solution. However, these are very useful studies and in fact, many of our design choices in Di-Sec stem from them.

2.2.7 Intrusion Detection Systems

Although Di-Sec is not solely an intrusion detection system (IDS) per se, it is a pertinent area to Di-Sec because using the facilities provided in our framework, an IDS could be

implemented. To have a complete security coverage in our literature review, we also analyze some intrusion detection systems. In [31], the authors propose a hierarchical framework for intrusion detection (ID). However the focus of this work is on providing solutions to only a specific subset of sensors called industrial sensors rather than providing a generic solution. Although this study claims to support several attacks using real sensors and report the performance of intrusion detection via real experiments, there is no explicit evaluation of the performance of each defense mechanism on sensors. For instance, the implementation details and the overhead and cost associated with the design were not analyzed. In the neighbor-based IDS scheme [32], the authors implemented an IDS on TinyOS and evaluated accuracy of the neighbor-based technique in detection of Selective Forwarding, Jamming and Hello Flood attacks. However, similar to [31] the focus is on the performance of the successful detection rate of the IDS rather than a generic security framework. In [33], a framework of a machine learning based IDS for WSNs was presented without any evaluation of the scheme; only the rules for the proposed IDS was listed without any results and real experiment on sensors. In [34], embedded sensor networks were utilized to supplement wireless intrusion detection systems (WIDSs) on physical site surveillance and security tasks. However, the main aim of this work is to aid current WIDSs in physical security via deployed sensors rather than designing an IDS framework for WSNs. On the other hand, the IDS works in [35, 36] only treat the matter via simulations without real experiments.

Di-Sec is fundamentally different from previous approaches in several ways. First, Di-Sec is neither an IDS nor a solution to a specific attack. It is a generic modular security framework for heterogeneous WSNs that can be easily extended and enhanced, used as a solution for innumerous attacks. Given the facilities provided by Di-Sec, an IDS can also be implemented in our framework. By default, the Di-Sec framework supports solutions on real sensors to several attacks at different layers of the communication stack including Jamming (physical layer), Sybil (MAC layer), Selective Forwarding (network layer). It also supports internal threats detection with the M-Core. The Di-Sec architecture was designed with modularity and flexibility in mind to ensure compatibility with future applications.

2.3 Network Reprogramming Protocols

When using Di-Sec, it is expected that cluster heads update the DDMs set in their cluster nodes. This action can be accomplished by using network reprogramming protocols and tools. This subsection presents our review of existing protocols including Deluge, the Secure Network Programming, Seluge, Dynamic TinyOS, and the Secure and Link-Quality Cognizant Image Distribution (SIMAGE) [13].

Deluge [37] is a reliable code dissemination protocol for propagating large binary images to all sensor nodes in multihop wireless sensor networks. It can push about 90 bytes per second (11% of the maximum transmission speed of the radio supported by TinyOS). Moreover, each node can maintain multiple code images, and quickly switch between different programs.

Deluge is a widely accepted and used method for code dissemination and it comes along with the distribution of TinyOS. Deluge splits the program image into pages and each page is split into fixed size packets. It generates an advertisement packet to propagate the information about the new program image version and the image is transmitted based on the Selective Negative Acknowledgement (SNACK) requests from the nodes. In Deluge, the propagation will be in terms of pages, that is, nodes will receive all the packets in a particular page before advertising it to the neighboring nodes. It uses an epidemic protocol and a page by page propagation method which uses spatial multiplexing for an efficient dissemination.

Since code dissemination is a critical step for the network reprogramming, securing this step is a requirement. There were many proposed ideas for the secure implementation of Deluge to provide encryption along with integrity. The Secure Network programming [38] proposed a public key - private key encryption method to sign the advertisement and a SHA1 hashing method for computing the hash of each packet. In this method, the computed hash of one packet is embedded into payload of previous packet in sequence. This allows the receiver node to verify the integrity of the received packet immediately and save or discard the packet based on the result. But in the case of out of order delivery scenario, the receiver

needs to cache the packet and wait for the previous packet in sequence to verify the cached packets hash. This can be used by the attacker to inject bogus packets and deplete the cache memory of receiver nodes.

Seluge [6] was also introduced as the secure extension of Deluge. It proposed a new propagation method which utilizes the efficient page by page propagation method of Deluge. A hash of each packet in a page is computed using SHA1 and this value is embedded in the previous page packets. So once all the packets in a page are received, the receiver has all the hash values for the packets in the next page. So it can immediately authenticate the out of delivery packet. Seluge inherits efficiency, robustness, and reliability from Deluge, and also provides protection for code dissemination. Seluge not only promises the integrity of code images but also resistance to various DoS attacks.

Another secure extension of Deluge is the Secure and Link-Quality Cognizant Image Distribution (SIMAGE). SIMAGE uses a dynamic link quality adaptive packet-sizing technique to reduce the retransmission bytes by 93% and the image transmission time by 35% when compared to other existing code dissemination protocols. It also provides confidentiality and integrity to the code dissemination process by utilizing energy-efficient encryption and authentication mechanisms with RC4 and CBC-MAC.

The last code dissemination tool that we analyzed to use with Di-Sec was the Dynamic TinyOS [39], which enables the dynamic exchange of software components and thus incrementally update the operating system and its applications. The benefit of using Dynamic TinyOS is that it allows replacements of program modules instead of the complete program image, which is faster and more efficient.

We evaluated all the code dissemination tools and tested Deluge and SIMAGE. *We chose SIMAGE as Di-Sec's network reprogramming tool* because it provides a better security and performance.

PART 3

DI-SEC FRAMEWORK

3.1 Di-Sec Framework Architecture

In this section, we discuss the architecture of the Di-Sec framework in detail. It consists of seven main components that have unique and important roles in the framework: the Monitoring-core (*M-Core*), the Communication Module (*COMM*), the Sensing Module (*Sense*), the Detection and Defense Modules (*DDMs*), the Reporting Control Module (*R-Control*), the Network/Transport Module (*NET*), and the Cluster Head Application (*CH*). The complete framework was implemented in TinyOS-2.x and tested using Tmote Sky and MicaZ sensors. The general Di-Sec architecture is shown in Figure 3.1. Along with the framework, we implemented four default DDMs using Di-Sec to defend against Jamming, Sybil, Selective Forwarding, and Internal attacks.

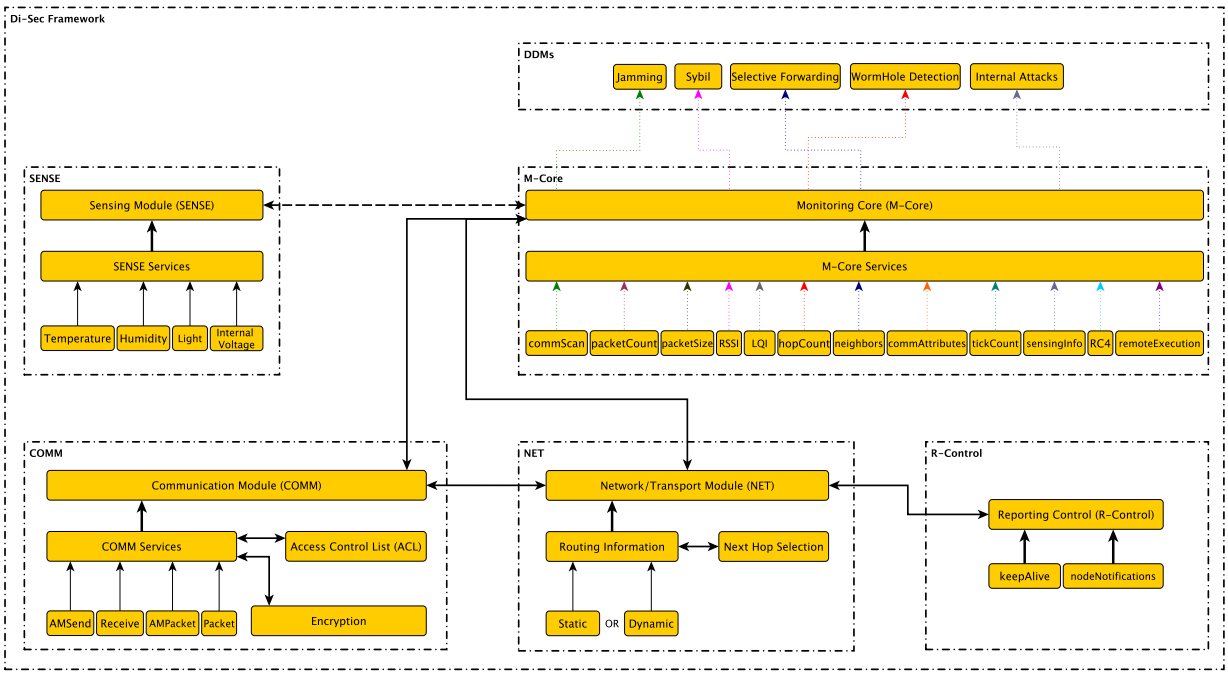

Figure 3.1 Di-Sec Architecture.

3.1.1 The Monitoring Core (M-Core)

The M-Core is designed to be the heart of a sensor node. M-Core proposes a novel way to provide information and support for the defense against both internal and external threats. The way M-Core collects its data is from the COMM and SENSE components. This data is analyzed and transformed into useful information at the M-Core. To reduce the complexity of the implementation and increase the flexibility and modularity of our design, M-Core is divided into sub-components (services), each of which provides some specific services to the defense and detection modules (DDMs). It is important to note that any of these sub-components can be removed or replaced, and more sub-components can be added to enhance the M-Core functionality. The current architecture of the M-Core is shown in figure 3.2.

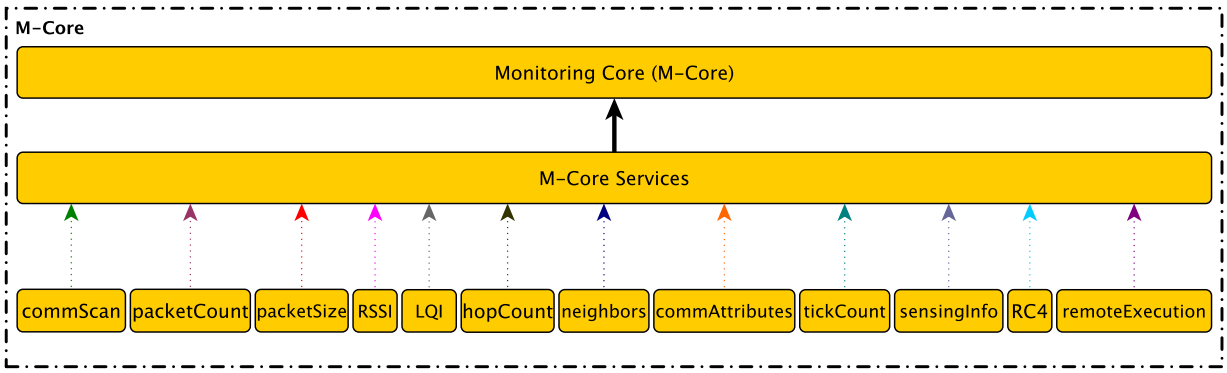

Figure 3.2 M-Core Architecture.

For each outgoing packet, the COMM module notifies the M-Core whether the transmission was successful or not. For all incoming packets, the COMM module passes a copy of the message to the M-Core even though the packet is not addressed to that specific sensor node. With the SENSE's module help, the M-Core is also able to intercept, monitor, and record all internal sensing measurement values and requests.

The interaction between the M-Core and the DDMs is shown in figure 3.3, where we illustrate four M-Core sub-modules (A, B, C, and D), each of which provides a service. Table 3.1 summarizes some of our implemented M-Core sub-modules and services. The M-

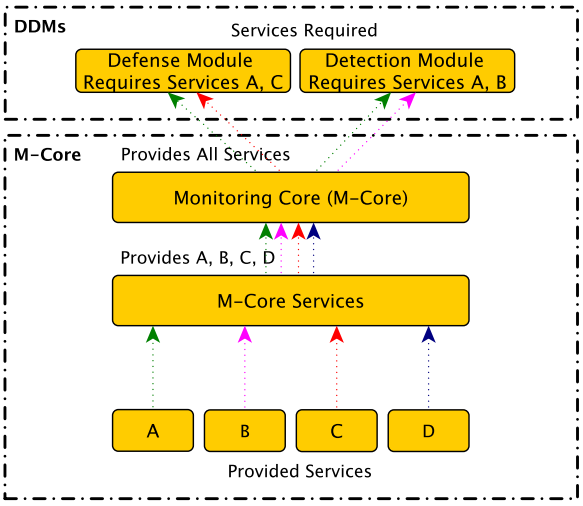

Figure 3.3 M-Core Interaction with DDMs.

Core services module advertises all the services provided by the sub-modules to the M-Core module, and the M-Core module allows the DDMs to access those services. In our implementation we included four security modules for the detection of: Jamming, Sybil, Selective Forwarding, and Internal attacks. The security modules are specific attack detections and/or defenses mechanisms against threats. Each security module includes the implementation of the necessary behavior utilizing the M-Core services. For instance, to implement the defense mechanism against sybil attacks, our sybil module uses the *rssivalue* interface provided by the received signal strength indication *RSSI* sub-module of the M-Core.

The benefits of using the M-Core when developing detection and defense mechanism for WSNs include: Built-in modular and flexible software architecture that provides an easy means to add, remove, and replace services. It is a lightweight monitoring and control layer invisible to upper layers (e.g., application layer). It is easy to activate and use. The provided services can be enhanced and expanded.

In TinyOS jargon, *interfaces* are used to interconnect *components*. Each interface define *commands* and *events* that can be used by developers of DDMs. Our implemented M-Core sub-components (services) are presented in table 3.1 and their descriptions are as follows:

Table 3.1 M-Core Sub-Components

sub-component	Interface	Commands/Events	Action
commScan	channelinfo	getConsecutiveSuccess	Returns the number of consecutive sent packets
		getpps	Returns the number of received packets per second
		setThreshold	Sets the threshould for acceptable consecutive sent packets
packetCount	packetcount	getPacketCount	Returns the total received packets
		lostPacket	Returns the number of lost packets by node
packetSize	packetsize	getPacketSize	Calculates and returns the size of a packets
		getReceivedPacketSize	Returns the average size of all the received packets
RSSI	rssivalue	getRssiTable	Returns neighbors RSSI table
		initRssiTable	Initializes the neighbors RSSI table
LQI	lqivalue	getLqiTable	Returns neighbors LQI table
		initLqiTable	Initializes the neighbors LQI table
hopCount	hopcount	hopcount	Initiates the hop count process
		hopcountDone	Notifies the nodes when the hop-count value is ready
neighbors	neighbors	request	Triggers a neighbor discovery message
	neighborsinfo	getNeighbors	Returns the number of current neighbors
		initNeighbors	Initializes current neighbors table
commAttributes	commAttributes	setCommChannel	Changes the communication channel
		setTransPower	Adjusts the transmission power a the specified value
tickCount	tickcount	getTicks	Returns the number of CPU ticks to transmit a packet
sensingInfo	sensingstat	getAvgSenseValue	Returns the average sensed value aggregated at the mcore
RC4	encryptI	encrypt	Encrypts/Decrypts a message based on a secret key
remoteExecution	remoteexec	execute	Executes command provided by another M-Core component

1) commScan The *commScan* sub-component can be used to detect the status of the communication channel based on two parameters: the number of consecutive successfully transmitted packets, and the average number of received packets per second. Both parameters can be obtained by the commands *getConsecutiveSuccess()* and *getpps()* respectively. To collect information about the successfully transmitted packets, we configured the COMM module to notify the M-Core whether a packet is successfully sent or not by signaling two different events, *packetsuccess()* and *packetfail()* respectively. For the received packets per second calculation, we count all the packets received by the node and average them every second. The COMM module passes a copy of all messages to the M-Core even when the packets are not addressed to the sensor.

This sub-component also provides a command *setThreshold()* that allows users to define a threshold for a minimum acceptable successfully transmitted packet rate. If this threshold is not reached, the commScan sub-component will notify the upper layer by signaling an event. Some other possible uses of this sub-component include the detection of correct (or incorrect) functioning of the radio transceiver, the amount of traffic in the channel, and congestion in the network.

2) packetCount The *packetCount* sub-component provides the interface *packetcount* with one command (*getPacketCount()*) to return the total number of packets received by the sensor, and one event *lostPacket()*, which is signaled every time a packet is lost by one of the node's neighbors. For the lostPacket event the sensor keeps track of all its neighbor's transmissions and maintains a four-tuple table to simplify the detection of a neighbor losing or dropping packets. The four-tuple contains the ID of the node who created the packet ($NodeS$), the ID of the node forwarding a packet ($NodeF$), the sequence number for the combination ($NodeS, NodeF$), and the total lost packets for the same combination. Hence, one can detect if any of the forwarding nodes F is dropping packets generated at node S. This sub-component can be used to detect unreliable communication links as well as malicious activities in the network.

3) packetSize The *packetSize* sub-component is used to get information about the size of the packets. The *packetsize()* interface provides two commands: *getPacketSize()* which was designed for the users to directly ask the M-Core for payload size of a specific packet, and *getReceivedPacketSize()* which returns the average payload size of all received packets. Average payload size information can be used in conjunction with the received packets per second information (provided by the *getpps()* command) to calculate network throughput.

4) RSSI and LQI The receiver signal strength indication ($RSSI$) and link quality index (LQI) are independent sub-components but the way they collect data and operate is very similar. A copy of every packet received by the COMM module is passed to each of these sub-components where they parse the message to extract the RSSI and LQI values respectively. The sub-components maintain a neighbors' table and all the extracted values are averaged with the corresponding values from the tables. Our current implementation supports $CC2420$, and $RF230$ radios, which are used by many of the sensors currently available in the market. However, support for other radios can be easily added. The information provided by these two sub-components can be used for defining reliable links in routing protocols, and also for device fingerprinting for authentication.

5) neighbors The *neighbors* sub-component provides two interfaces, *neighbors* and *neighborsinfo*, to gather neighbor information. Using the *request()* command from the *neighbors* interface, the sensor initializes a neighbor discovery process where it broadcasts a discovery message that is acknowledged by all the neighbors within its transmission range. Once the acknowledgement is received, the sensor refreshes its neighbor table with updated information. Accordingly, the *getNeighbors()* command provided by the *neighborsinfo* interface returns the neighbors' table, which can be initialized or reset with the *initNeighbors()* command. The information provided by this sub-component can be used to dynamically create communication routes for mobile WSN deployments.

6) commAttributes The *commAttributes* sub-component is used to adjust communication attributes such as the communication channel and the transmit power using the *setCommChannel()* and *setTransPower()* commands respectively. These attributes can also be modified at run time, which is convenient for users that need to tune their application while running.

7) tickCount The *tickCount* can be used to verify the CPU load. The command *getTicks()* provided by the *tickcount* interface initiates the transmission of an arbitrary packet to calculate the number of CPU ticks elapsed from the message creation until the confirmation of the transmission. This value varies according to the CPU load, and can be used to establish a threshold for minimum or maximum CPU utilization. This sub-component can also be used for security to observe abnormal activities onboard.

8) sensingInfo The *sensingInfo* sub-component collects information about the sensor readings and keeps an average of the measured sensing values. Sensor information is important to monitor since it can be used to verify the correct functioning of the sensor itself. For instance, an overloaded sensor might report a higher temperature readings due to overheating.

9) RC4 An instance of the *RC4* [40] encryption algorithm is also provided as a service in the M-Core. The *encrypt* command provided by the *encryptI* interface is used to *encrypt* and *decrypt* a message that is passed as a parameter along with the encryption key of a certain size.

10) hopCount The *hopCount* is used to enable all the nodes to determine their hop-count values with respect to the cluster head. It provides the interface HopCount with a command *hopcount()* to initiate the hop-count service and an event *hopcountDone()* which is signaled once the nodes have determined their hop-count values. This service can be used periodically or on-demand.

11) remoteExecution The *remoteExecution* is a sub-component used to remotely call commands from other other M-Core sub-components to initialize variables as well as configure sensors attributes. This component can be used to initialize neighbors, RSSI, and LQI tables, to remotely change the channel and transmit power, and to adjust any parameter available in the M-Core.

3.1.2 The Communication Module (COMM)

The current architecture of the communication module is shown in figure 3.4.

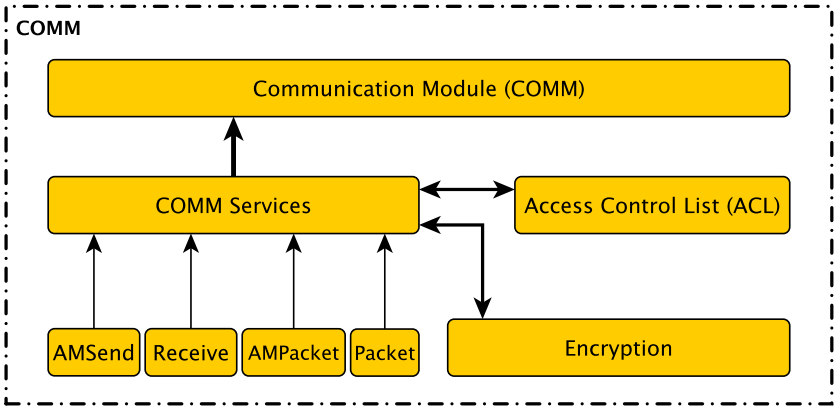

Figure 3.4 COMM Architecture.

The communication module provides the main communication interfaces: AMSend, Receive, Packet, and AMPacket. AMSend is the active message sending interface, Receive provides a message reception interface, Packet facilitates access to the message_t data type, and AMPacket for active message accessors for the message_t data type. When using the Di-Sec framework, all the packets will pass through the communication module. For each outgoing packet, the COMM module notifies the M-Core whether the transmission was successful or not (Figure 3.5). For all incoming packets, the COMM module passes a copy to the M-Core even though the packet is not addressed to that sensor node (Figure 3.6). The COMM module is also in charge of adding Di-Sec headers to all outgoing packets before transmitting them and analyzing the headers when packets arrive (Figure 3.7).

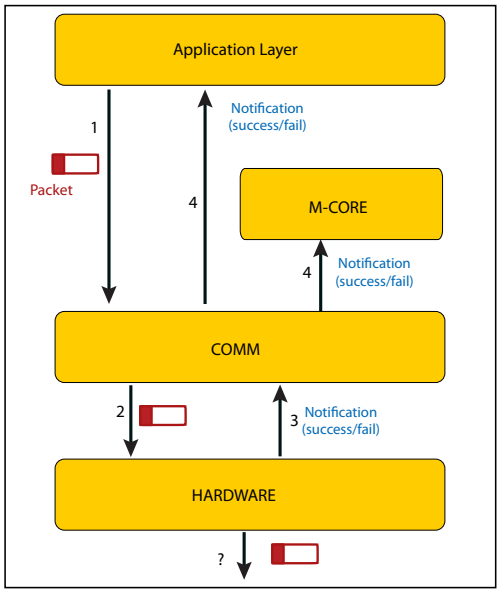

Figure 3.5 COMM when Sending Packets.

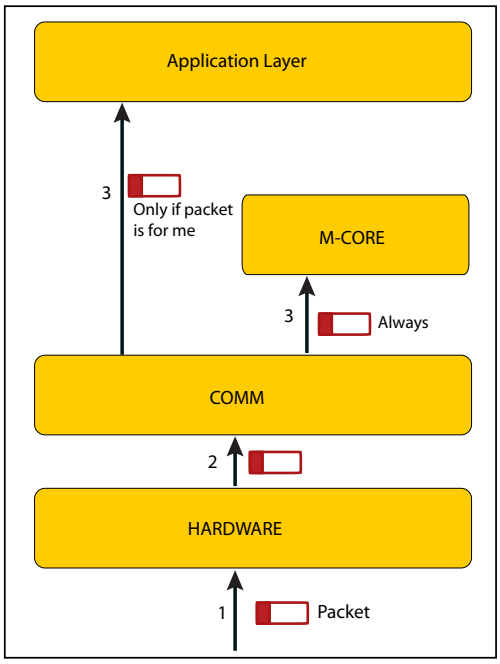

Figure 3.6 COMM when Receiving Packets.

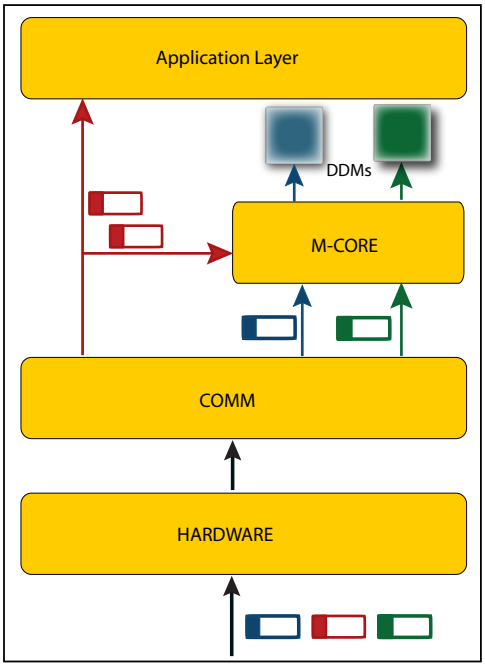

Figure 3.7 COMM Headers and Multiplexing.

The purpose of Di-Sec headers is to facilitate control of the communications and also for the multiplexing of the messages. From figure 3.7 we notice that DDMs and other modules inside the M-Core can also communicate securely with the same DDM or M-Core module in other sensors through the COMM module. Figure 3.8 shows other important features of the COMM module: encryption and access control lists (ACL). When encryption is enabled, all the packets sent by the COMM module are securely encrypted using the embedded AES-128 encryption provided by the CC2420 radio transceiver. The ACL can be configure using the *aclmanager* interface provided by the *coreaclC* M-Core service. The *aclmanager* interface provides the commands *addNode* to the ACL, *deleteNode* from the ACL, and *findNode* in the ACL.

To increase the simplicity and cleanliness of activating and utilizing the Di-Sec framework, we assigned the COMM module to be the framework's activation component. Users can easily enable the Di-Sec framework by adding the COMM module and wiring it to their applications as shown in Listing 3.1.

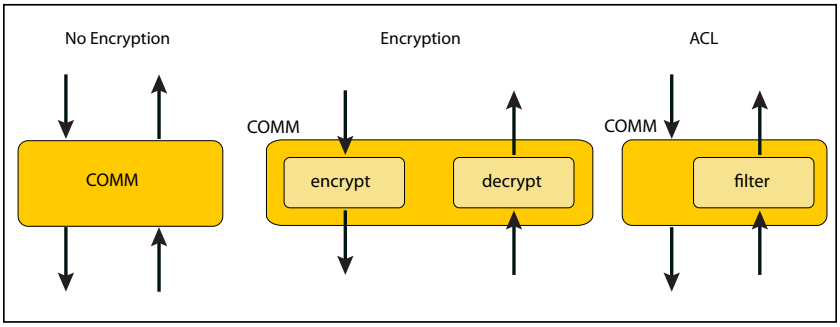

Figure 3.8 COMM Operation Modes (Encryption and ACL).

Listing 3.1: How to Enable Di-Sec.

```
components new Comm(AM_MSG);
MainC.SoftwareInit->Comm.Init;
App.Packet -> Comm;
App.AMSend -> Comm[AM_MSG];
App.Receive -> Comm[AM_MSG];
```

3.1.3 The Sensing Module (SENSE)

Similar to the COMM module, we added to our framework the capability to intercept, monitor, and record all internal sensing measurement values and requests. We implemented a sensing component to facilitate upper layers to get information such as the temperature, humidity, total solar radiation, Photosynthetically active radiation, and internal voltage by calling simple commands like *sensing.getTemperature()* or *sensing.getHumidity()*. The current architecture of the sensing module is shown in figure 3.9.

The main functionality of this module in our framework is the capability to monitor internal sensing activities. In this way, the sensing component is used to supplement the detection and defense modules security mechanisms. For example, the Sensing module can be used to leverage the temperature sensing component to detect the temperature of the sensor itself. A significant spike in temperature could indicate that the CPU has been excessively utilized, possibly as a result of malware. Another use of the Sensing module would be to

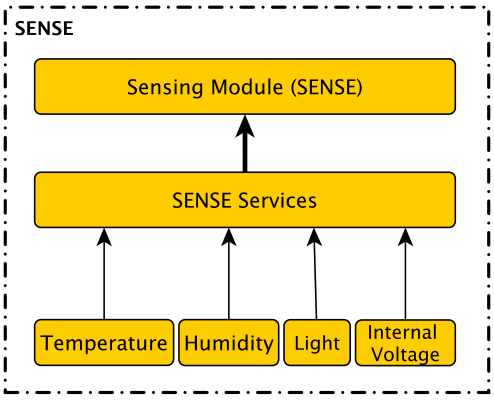

Figure 3.9 SENSE Architecture.

detect various types of proximity-based attacks, where the sensing component is used to trigger malware (i.e., a certain light pattern observed by a light sensing component used to activate a Trojan on the sensor). Figure 3.10 shows the interaction of the SENSE component with the M-Core.

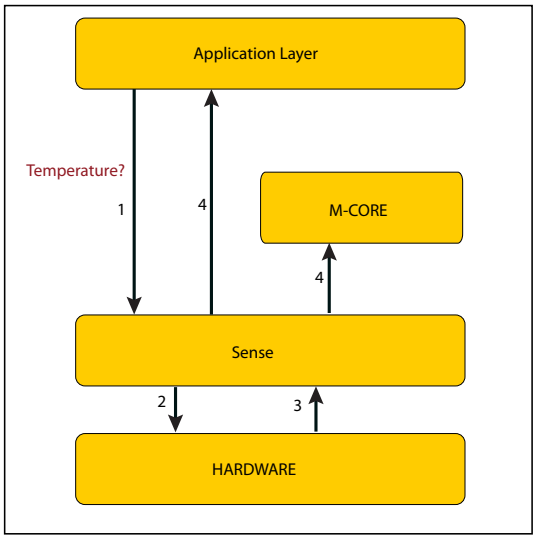

Figure 3.10 SENSE Interaction with M-Core.

3.1.4 The Detection and Defense Modules (DDMs)

DDMs are specific attack detections and/or defenses behaviors mechanisms against threats, but can also be used as conduits to provide services to other layers. Each DDM would include the implementation of the necessary behavior utilizing the M-Core services. Like the M-Core sub-components, the DDMs have a modular architecture too and can be added, removed, and replaced without affecting the rest of the framework. In order to not restrict the Di-Sec framework to only detection and defense mechanisms implemented for our architecture, **we allow the DDMs to communicate and collaborate with external security mechanisms as well**. This feature enhances the main functionality of the DDMs. For instance, a network layer that implements an implementation of a Sinkhole attack defense mechanism does not have to be ported into our framework, but it can use the services provided by the M-Core through the easy implementation of a DDM that will actually act as an information conduit. We implemented four different detection and defense mechanisms against Jamming (DDM1), Sybil (DDM2), Selective Forwarding (DDM3), and Internal attacks (DDM4) which are distributed with the framework as the default DDMs. Figure 3.11 shows the interaction of the DDMs component with the M-Core. Note that the details for the behavior and implementation of each individual attack are discussed in the performance evaluation section along with the results.

3.1.5 The Reporting Control Module (R-Control)

In the same way the M-Core provides local services to monitor and control sensor nodes individually, the R-Control is intended to manage *network services* to monitor and control the cluster. The R-Control also multiplexes incoming and outgoing messages to deliver the packets to the corresponding modules. These modules implement the algorithms and required behavior to facilitate the control of the cluster and communication with cluster heads. For instance, the keepAlive module sends keep alive messages from all the nodes to the cluster head every 30 seconds, and the nodeNotifications module allows the cluster head to send messages and notifications to individual nodes.

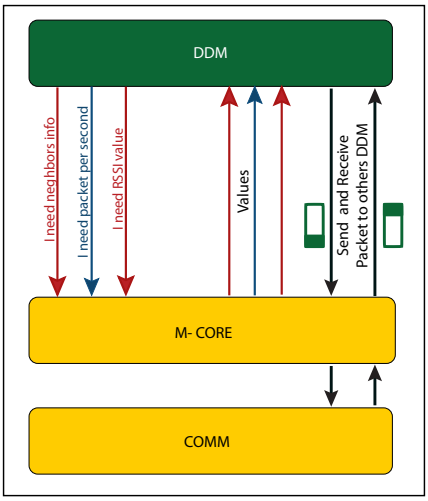

Figure 3.11 DDM Component.

In order to detect malicious activities or unexpected behaviors in the network, each regular node is required to periodically send its current state to the cluster head. If the cluster head does not receive the report on time from a specific node, it assumes that it has been compromised, is dead, or a link failure has occurred. Thus, the cluster head can request more information from the neighbors of the unresponsive node. Cluster heads also have the ability to communicate with DDMs in the sensors and send requests to activate defense actions. The current architecture of the reporting control module is shown in figure 3.12. Note that more network services modules can be easily added to the R-Control in a similar way that we add services to the M-Core.

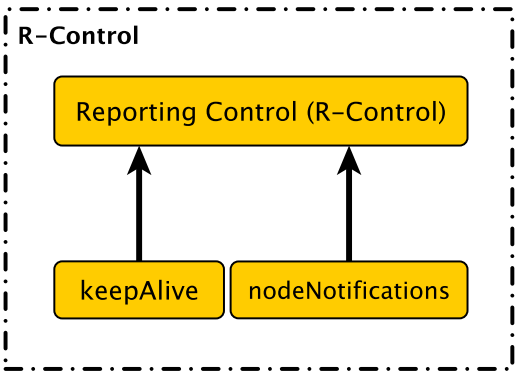

Figure 3.12 R-Control Architecture.

The interaction of the network services modules with the R-Control is presented in figure 3.13. The R-Control does not intend to provide any routing or network layer protocol. From figure 3.13 we can see that the R-Control relies on Di-Sec's Network/Transport module to take care of message forwarding and delivery.

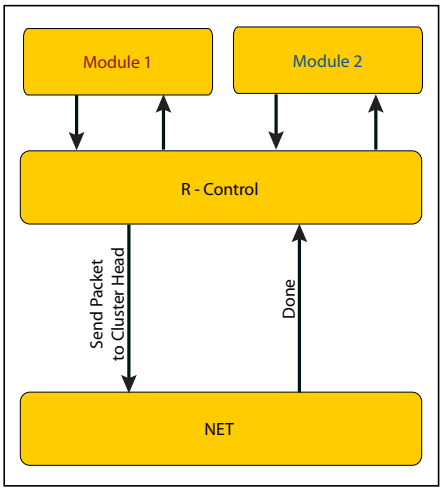

Figure 3.13 Interaction of Network Services Modules and R-Control.

The main difference between M-Core and R-Control is that M-Core provides services that are used to implement DDMs, while the R-Control manages network services modules that implement a complete behavior.

3.1.6 The Network/Transport Module (NET)

The network/transport module (NET) allows the Di-Sec framework to have communication independence. This means that Di-Sec can define its own routing algorithms and forwarding paths. This is important since the next hop selection depends on the status of the network. If the cluster is under attack, the NET module can update the routing information to make sure the messages reach the cluster head. The NET module also makes sure we have a reliable message forwarding and delivery in our internal communication. This module provides a similar functionality as the network and transport layer of the TCP/IP protocol stack and its current architecture is presented in figure 3.14.

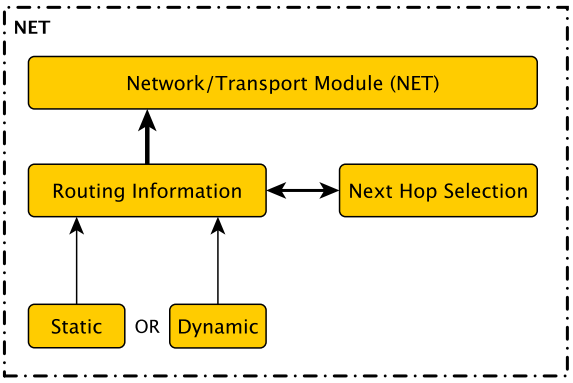

Figure 3.14 NET Architecture.

The modularity of the NET layer allows us to use any available routing protocol. In our implementation and experiments we used a static routing protocol (routing tables), but any dynamic protocol can be use. The benefit of using a dynamic routing protocol is that every node constructs a map of the network and independently calculates the best path to every possible destination based on its collected information. All the messages passed to the NET module only contain the final destination node ID and the module handles the routing and next hop selection (figure 3.15).

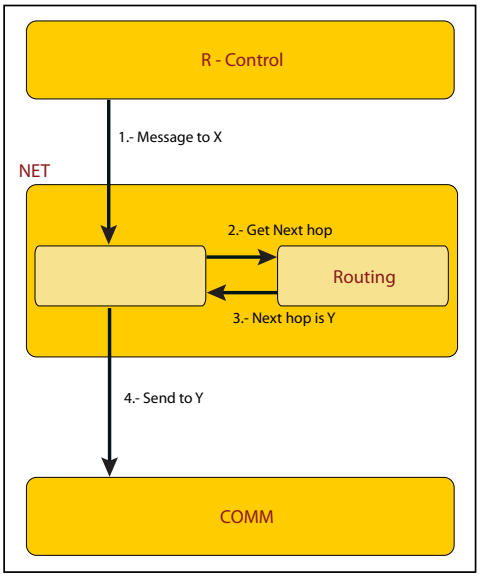

Figure 3.15 NET when Sending Packets.

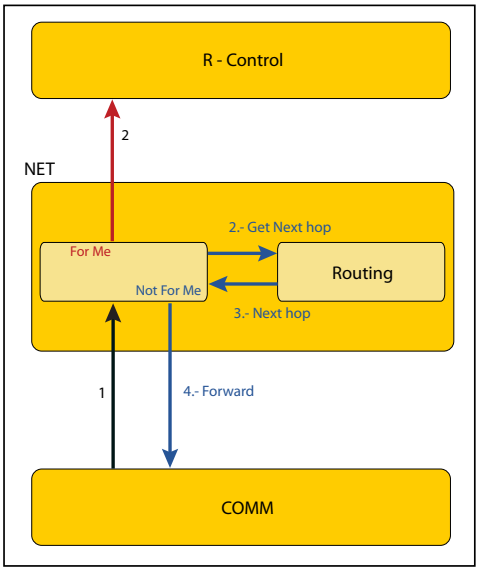

Figure 3.16 NET when Receiving Packets.

Figure 3.16 shows the behavior of NET when receiving a packet: if the message is intended for that node it passes the packet to the R-Control, otherwise it finds the next hop and forwards the message.

3.1.7 Cluster Head Module (CH)

Although each cluster is autonomous, the cluster heads collaborate with each other by exchanging information and warnings to provide a more robust security solution. The use of cluster heads facilitate the monitoring and control of the network, and highly increase the security of the cluster. Di-Sec's cluster head module (CH) consist of two applications App 1 (*Monitoring*) and App 2 (*SmartDDM*). App 1 is for monitoring and communication with the R-Control modules running in the regular nodes. App 2 is for DDM selection and DDM redistribution. Our cluster heads are 10" Dell Netbooks with Ubuntu Netbook Edition 10.04 connected to 2 MicaZ sensors as shown in figure 3.17. Sensor 1, which interacts with the App 1, is running the Di-Sec framework to receive information from the sensors and communicate with all the devices in the cluster. Sensor 2 is a passive sensor used for network reprogramming and activated by App 2.

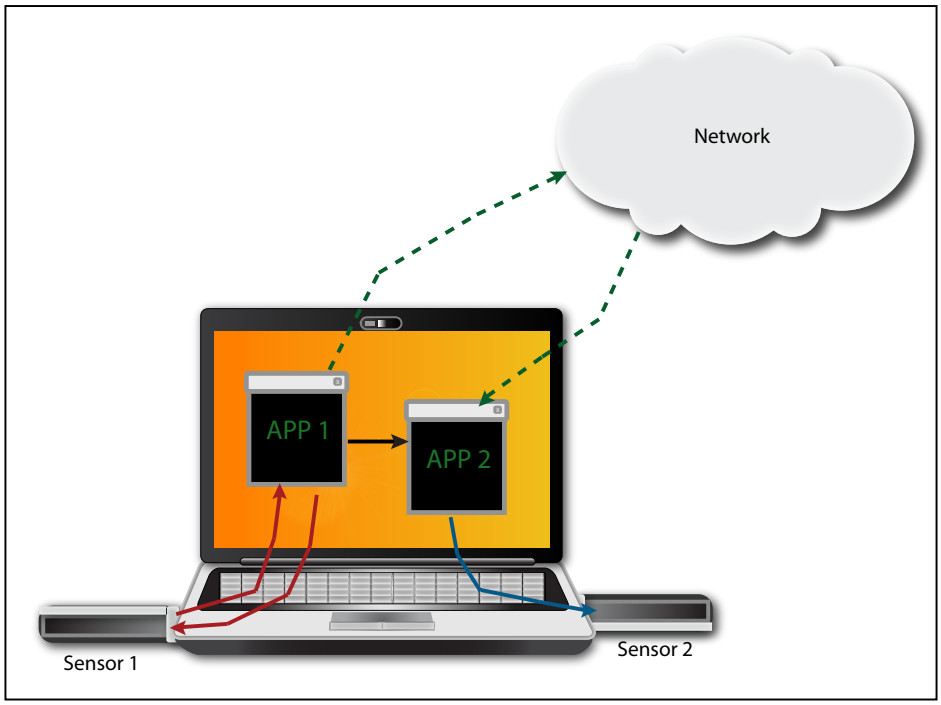

Figure 3.17 Cluster Head Architecture.

App 1 (*Monitoring*) is in charge of monitoring the cluster and detecting attacks and misbehavior in the network. A basic application of the App 1 has been implemented and tested to show that it successfully receives notifications, detects attacks, and communicates with other cluster heads. App 1 is a Java application connected to the TinyOS *SerialForwarder* to receive and send packets. Once an attack is detected by this program a new TCP socket connection is established with other cluster heads and a warning message is sent. Figure 3.18 presents this cluster heads communication and shows that the App1 communicates with the App 2 in other clusters.

App 2 (*SmartDDM*) is an intelligent Java application consisting of two main components: DDMs Selection and DDMs Redistribution. A basic application of the App 2 has been implemented and tested to show that it successfully receives warnings from other cluster heads and activate the network reprogramming protocol (SIMAGE in our case). Even though the App 2 was not fully implemented for our experiments (no sophisticated decisions), it provides the base schema for future enhancements.

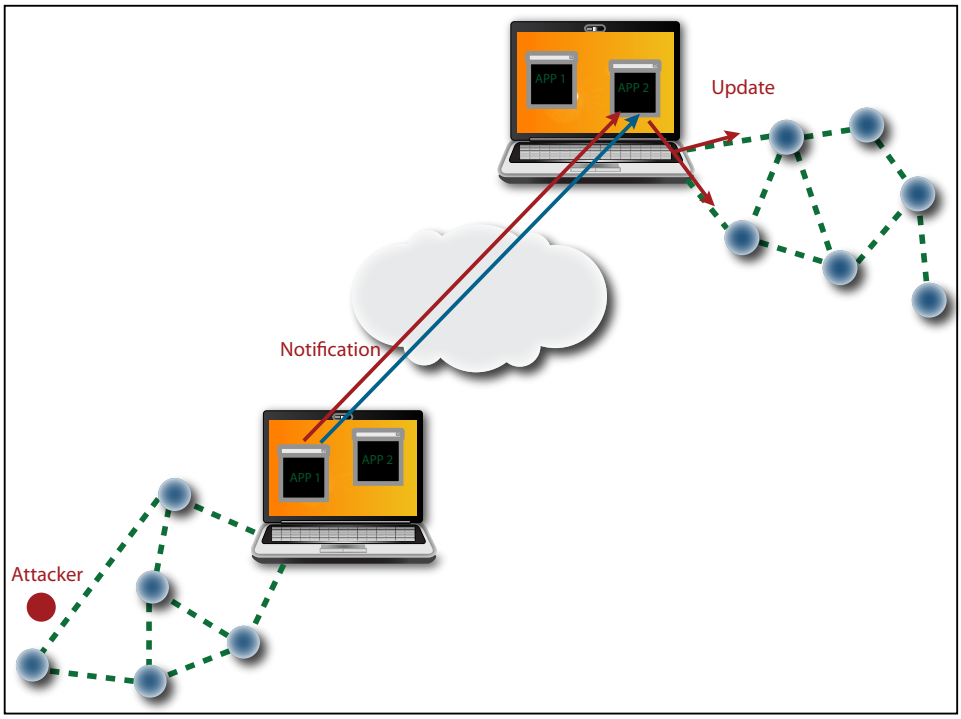

Figure 3.18 Cluster Heads Communication.

The following are some details to be considered to enhance the *SmartDDM* application:

DDMs Selection:

The main problem is to find a solution to select the optimal subset S of $DDMs$ to be installed in the sensors. Although we would like to include as many defense schemes as possible in S, the cluster head will be responsible for determining the optimal subset S according to different parameters including the likelihood of being attacked, more frequent attacks, severity of the attacks, detectability and attacks on other clusters. Since an attacker can repeat the same attack or launch a new attack at any time in the network, the likelihood of occurrence of each attack at a given time will be measured at each cluster head. Based on this likelihood and the other parameters, the cluster heads recalculate the subset S for the sensors in their respective clusters. As a result, an attacker can launch an attack and compromise a node or several nodes because S does not contain the detection and defense mechanism for that attack. However, the attacker cannot continue to compromise more nodes once the cluster head detects it and propagates defense schemes to the regular nodes.

Given that memory capacity is not a concern with cluster heads, we assume that the cluster heads have all DDM modules available in their databases for all the known attacks. Whether the detection and defense scheme for an attack is available in the subset S of the regular nodes or not, the base station would be able to identify an undergoing attack by detecting the unexpected behaviors in the network.

DDMs Redistribution:

Given the multiplicity of detection and defense mechanism available for different attacks, a network reprogramming tool is a necessity. Even though there exists a variety of tools to deploy new software updates and upgrades into systems, TinyOS is still limited to full image replacement as nodes only execute a statically-linked system-image generated at compilation time. As stated in Section 2.3, there were different options that we considered for the Di-Sec program redistribution including SIMAGE, Deluge, the Secure Network Programming, Seluge, and Dynamic TinyOS. For our experiments we chose SIMAGE since it has better performance and also provides a secured network re-programming.

3.2 Di-Sec Architecture Overview

Now that we have introduced and described all the components of the Di-Sec framework, we present the complete implemented architecture in the following figure:

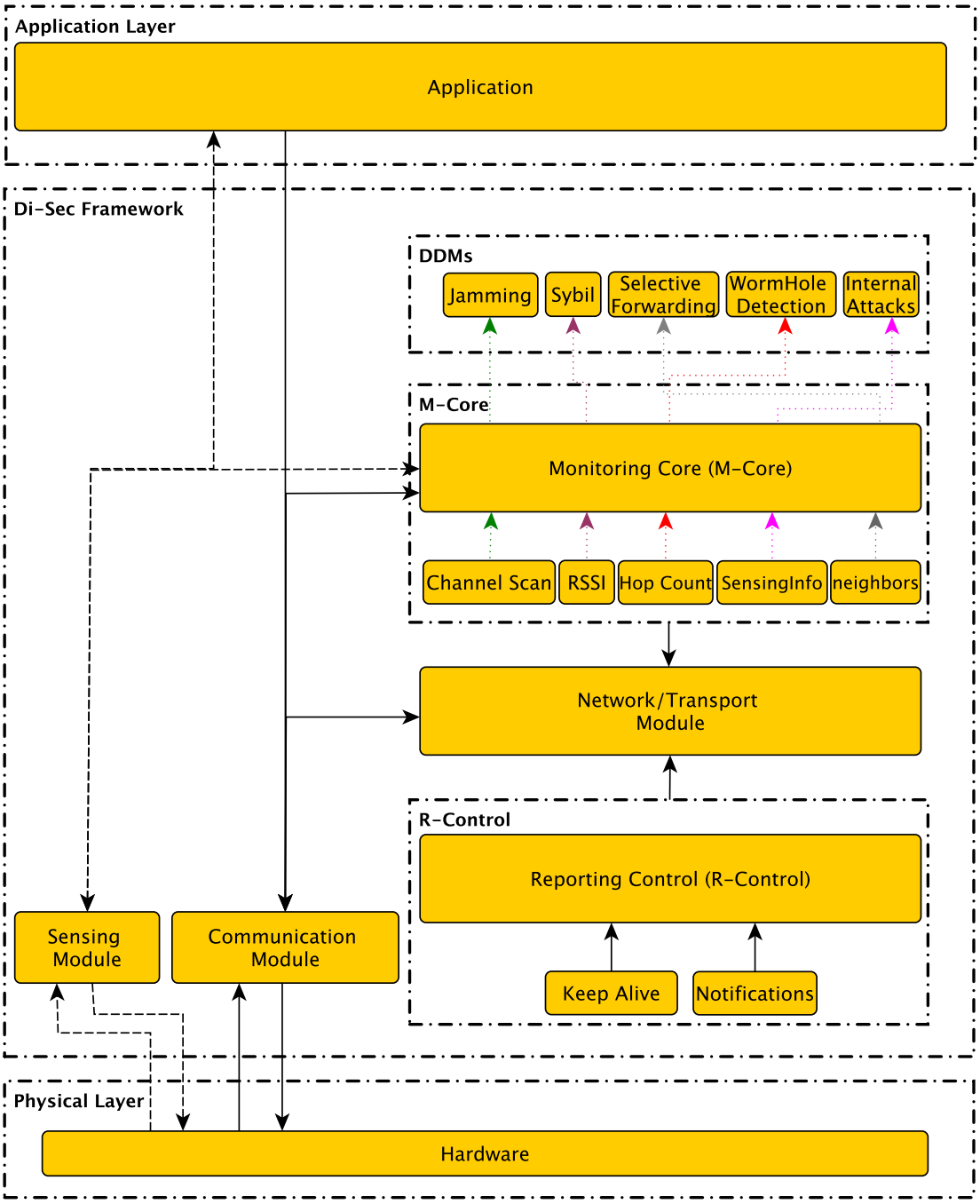

Figure 3.19 Di-Sec Complete Architecture.

3.3 M-Core Control Language (MCL)

In order to easily use the Di-Sec framework, we have created a new domain specific language: the M-Core Control Language (MCL). In this section, we introduce the MCL, present the formal grammar of the language, and show how it can be used to activate, deactivate or create new detection and defense modules with an example.

3.3.1 Rationale for MCL & Formal Definition

Di-Sec was designed to provide a comprehensive security framework to programmers when implementing DDMs. However, a programmer who would like to use the framework would still need to do some additional implementation (e.g., wiring in TinyOS) to take advantage of the existing defense and detection modules or create new ones. Moreover, this situation may be exacerbated given the sophistication needed to implement programs on sensors for a novice programmer. The MCL has been designed to address this issue. It utilizes the sub-modules defined in the M-Core and simplifies the programmer's work to easily activate, deactivate or create their own new defense mechanisms by automatically generating important programming components needed for the underlying Di-Sec architecture (e.g., configuration files, module files and wiring). Primarily, the MCL is a language consisting of a small set of keywords. The formal definition of the grammar of the MCL using the Extended Backus-Naur Form (EBNF) is given in Listing 3.2.

Also, the list of all the keywords in the MCL and their simple descriptions are tabulated in Table 3.2. A program written with the MCL starts and ends with the keywords, *START* and *END*. Between these, one can use the other keywords *ACTIVATE*, *STOP*, or *NEWDDM* to activate, deactivate or create a DDM respectively. A programmer can even define its own variables using the *SET* keyword.

Table 3.2 The Keywords of MCL.

Keywords	Descriptions
START	Starts the program
END	Ends the program
ACTIVATE(*module name*, *time*)	Activates an existing *module name* at specific *time* (ms)
STOP(*module name*)	Deactivates an existing *module name*
SET(*variable name*, *attribute*, *value*)	Creates a new *variable* with a *value*
ASSOCIATE(*module name*, *interface name* ...)	Associates a *module name* with one or more *interface name*
DISSOCIATE(*module name*, *interface name* ...)	Dissociates a *module name* with one or more *interface name*
COMMUNICATE(*module name*, *packet field* ...)	Adds communication capabilities to *module name* using one or more *packet fields*
NEWDDM(*module name*, *interface name* ...)	Creates a new detection and defense module
NEWSERVICE(*module name*, *interf. name* ...)	Creates a new M-Core service

Listing 3.2: Formal definition of MCL with EBNF.

```
MCL ::= 'START', SPACES,
        { KEYWORDS, '(', EXPRESSIONS, ')', SPACES }, 'END' ;
KEYWORDS ::= 'ACTIVATE' | 'STOP' | 'SET' | 'ASSOCIATE' |
             'DISSOCIATE' | 'COMMUNICATE' | 'NEWDDM' | 'NEWSERVICE' ;

EXPRESSIONS ::= PARAMETERS, { [ ',', SPACES,
            PARAMETERS ] }, [ ',', SPACES, VALUE ] ;

PARAMETERS ::= [a-zA-Z]\w* ;

VALUE ::= \d* ;

SPACES ::= ' '* ;
```

3.3.2 Sample Usage

In this sub-section, we show a sample usage of MCL. In our realistic scenario, the user implements a secure WSN program using MCL to protect against several attacks. The MCL written by the user is given in Figure 3.20 (code snippet in the middle). Specifically, the user instructs the Di-Sec to activate and deactivate the existing defense and detection modules D1, D2, and D3. The user also adds a new module, D4, into Di-Sec and sets the specific activation time and specifies that it use the *cpucycles* sub-component of the M-Core. In the example, *ACTIVATE* enables the existing D1 and specifies the D1 starting time. *ASSOCIATE* is used to connect the D1 to the sub-modules of M-Core in Di-Sec. Also, *STOP* simply disables the existing D2 and D3 that might not be used at run time and disconnects them from the sub-modules. Moreover, *NEWDDM* adds the new D4 module configurations into Di-Sec and generates a new template file for the D4 module implementation. With this one keyword (*NEWDDM*), the users can start writing their own detailed defense mechanisms in the template file without worrying about the underlying details of the Di-Sec and TinyOS. As seen in the figure, a user would be able to handle existing detection and defense modules and create new ones with simple keywords. Most importantly, the conversion from MCL to the necessary underlying components (i.e., side files in Figure 3.20) of the Di-Sec framework and the integration with Di-Sec are automatically handled by MCL.

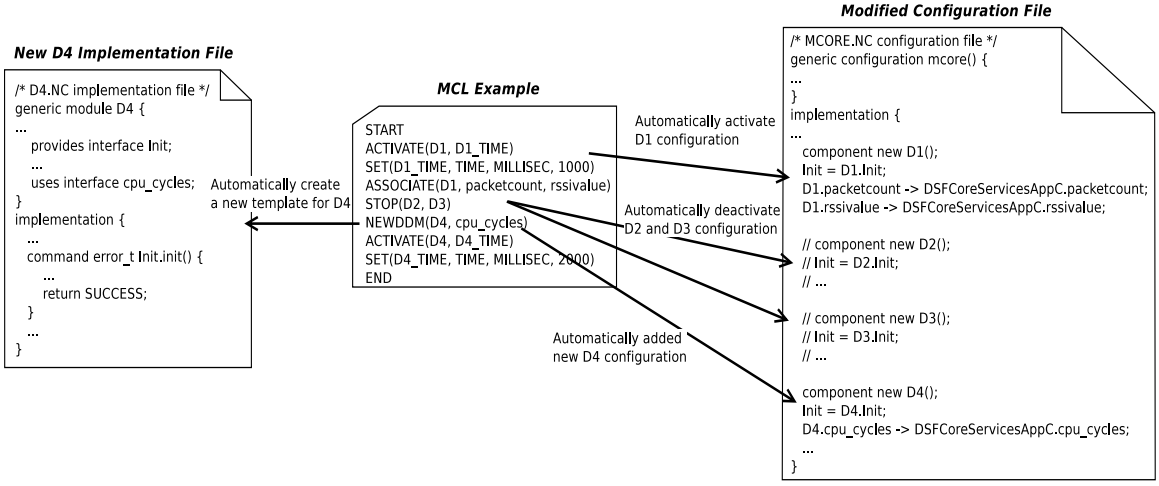

Figure 3.20 A realistic example usage of MCL.

3.4 Performance Evaluation

Given that different sensor platforms have different characteristics, the analysis of the performance of the framework depends on the platform. For example, Let us consider Telosb and Iris motes. Telosb sensors use the CC2420 radio transceiver and MSP430 micro controller, while the Iris motes use the RF230 radio and ATmega1281 micro controller. Each chip has different characteristic, performance, and energy consumption. Therefore, the average energy consumption in one platform it is not necessary same in other platform. For our performance evaluation we only focus on two platforms that we are currently supporting with Di-Sec: Micaz and Telosb.

In this section, we evaluate the performance of the Di-Sec framework on real sensors in four dimensions: (1) we evaluate the different components' storage costs (RAM and ROM), (2) we show the CPU overhead, (3) we evaluate Di-Sec's communication overhead, and (4) we present the energy consumption evaluation based on simulations.

3.4.1 Storage Costs

In our evaluation, we present 11 configurations with different components to analyze the cost of each of them. We have 1 plain upper layer configuration and 10 Di-Sec configurations. In this experiment the upper layer is an application layer provided with the default installation of TinyOS: *RadioCountToLeds*. The configurations are the following:

- (A) Plain Upper Layer.
- (B) M-Core Full.
- (C) M-Core + Encryption.
- (D) M-Core + Encryption + ACL.
- (E) M-Core + Encryption + ACLSENSE.
- (F) M-Core + Jamming DDM.

Table 3.3 Di-Sec ROM and RAM Footprint (Bytes)

Platform	**MicaZ**				**Telosb**			
CONFIG	**ROM**	**ΔROM**	**RAM**	**ΔRAM**	**ROM**	**ΔROM**	**RAM**	**ΔRAM**
A	18496	18496	1655	1655	20610	20610	1824	1824
B	20650	2154	2491	836	22630	2020	2572	748
C	23140	2490	2607	116	25484	2854	2710	138
D	23542	402	2748	141	26018	534	2730	20
E	23542	0	2748	0	29218	3200	2816	86
F	24570	1028	3029	281	30030	812	2988	172
G	24882	312	3158	129	30244	214	3084	96
H	25260	378	3182	24	30522	278	3112	28
I	25260	0	3182	0	30564	42	3128	16
J	43194	17934	3921	739	42678	12114	3936	808
K	43842	648	4146	225	43010	332	4136	200

- (G) M-Core + Jamming DDM + Selective Forwarding DDM.
- (H) M-Core + Jamming DDM + Selective Forwarding DDM + Sybil DDM.
- (I) M-Core + Jamming DDM + Selective Forwarding DDM + Sybil DDM + Internal DDM.
- (J) M-Core + Jamming DDM + Selective Forwarding DDM + Sybil DDM + Internal DDM + SIMAGE.
- (K) M-Core + Jamming DDM + Selective Forwarding DDM + Sybil DDM + Internal DDM + SIMAGE + R-Control.

The costs of the different components in terms of storage are presented in Table 3.3. For ROM, we observe that the network reprogramming tool (SIMAGE), the sensing component, and the encryption components have the largest storage costs. For RAM, M-Core has the largest cost, which is expected because the M-Core includes all the submodules previously discussed. Table 3.4 shows a summary of the cost for each of the Di-Sec's Components.

Table 3.4 Di-Sec ROM and RAM Footprint (Bytes)

Platform	**MicaZ**		**Telosb**	
COMPONENT	**ROM**	**RAM**	**ROM**	**RAM**
COMM	2892	257	3388	158
SENSE	0	0	3200	86
M-Core	2154	836	2020	748
DDMs	1718	434	1346	312
R-Control + NET	648	225	332	200
SIMAGE	17934	739	12114	808

3.4.2 CPU Costs

For the CPU footprint we captured the CPU cycles introduced when using Di-Sec. To show the CPU overhead we recorded the CPU information from a sensor running a basic application with and without Di-Sec. We used two techniques to collect this information. The first was adding monitors into our code to keep track of the CPU cycles, and the second was using the Avrora [41] simulation tools, which shows the total CPU cycles at the end of the simulation. Table 3.5 shows the CPU overhead when sending and receiving packets, and collecting values from the sensor. We compare CPU ticks of the plain application configuration and the full M-Core configuration. The results show that M-Core adds an overhead for the transmission scenario which is expected since our framework adds and verifies Di-Sec headers before transmitting the packets. For the receiving scenario we do not see any overhead since the COMM module passes the incoming packet directly to the upper layer as soon as it is received. For the sensing component there is a minimal overhead since the request has to pass through Di-Sec's Sense component.

Table 3.5 Di-Sec CPU Ticks

	M-Core	**Plain**	**Diff**
TX	352	239	113
RX	1600	1600	0
Sensing	550	546	4

Table 3.6 Di-Sec CPU Cycles

	30 Sec		**60 sec**		**120 sec**	
Config.	CPU Cycles	% Increase	Cycles	% Increase	Cycles	% Increase
A	8432964	0	8850145	0	9686985	0
B	8610382	2.104	9196315	3.911	10370376	7.055
C	8692504	3.078	9376776	5.951	10744125	10.913
D	8740946	3.652	9472368	7.031	10933816	12.871

We also used Avrora with four different configurations and simulation times to evaluate Di-Sec's CPU cycles. The configurations are the following:

- (A) Plain Upper Layer.
- (B) M-Core Full.
- (C) M-Core + Jamming DDM + Sybil DDM.
- (D) M-Core + Jamming DDM + Selective Forwarding DDM + Sybil DDM + Internal DDM

From table 3.6 we see that Di-Sec adds up to 12% more CPU cycles when running the full framework including the four default DDMs for 120 simulated seconds.

3.4.3 Communication Costs

The Di-Sec header is 6-bytes long and includes 2-byte source node ID and 1-byte Di-Sec sequence number managed individually at each sensor. The header also includes a 2-byte DDMtype variable used to multiplex the message to Di-Sec modules and a 1-byte command variable used for internal communication. Figure 3.21 is a packet sniffer capture showing the difference between the packets sent with the plain application layer, and M-Core with and without encryption. The Di-Sec communication overhead is only 6 header-bytes added to the message plus the overhead of the encryption and decryption of the payload.

Time (us)	Length	Frame control field							Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	LQI	FCS
		Type	Sec	Pnd	Ack req	Intra PAN									
+0													3F 06 00		
=0	17	DATA	0	0	0	1			0x00	0x0022	0xFFFF	0x0001	01 00 01	176	OK
Time (us)	Length	Frame control field							Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	LQI	FCS
		Type	Sec	Pnd	Ack req	Intra PAN									
+17189672													3F 06 01 00 01 00		
=17189672	23	DATA	0	0	0	1			0x00	0x0022	0xFFFF	0x0001	00 00 00 01 00 01	216	OK
Time (us)	Length	Frame control field							Sequence number	Dest. PAN	Dest. Address	Source Address	Encrypted MAC payload		LQI
		Type	Sec	Pnd	Ack req	Intra PAN									
+12595802													E8 00 00 00 01 01 3F 06 4A D1 E9 78 45 5F A5 7A C0		
=29785474	45	DATA	1	0	0	1			0x00	0x0022	0xFFFF	0x0001	17 4A 34 9D 80 EE 92 3C B5 A2 ED 88 BF 56 BA D8 2E		204

Figure 3.21 Same Information to the Application Layer on Different Packet Payloads.

We also used Avrora to confirm that the simulation produces the same results as our experiments. Figure 3.22 shows the simulation interaction between sensors running the plain *RadioCountToLeds* application. This is an example where each node transmits only 2 packets. Figure 3.23 shows the result for the same experiment with the sensors running Di-Sec. The simulation packets are the same that we capture with the packet sniffer and we can see that for 2 packets the plain application transmitted 42 bytes while the application running Di-Sec transmitted 54 bytes (12 extra Di-Sec header bytes for 2 packets).

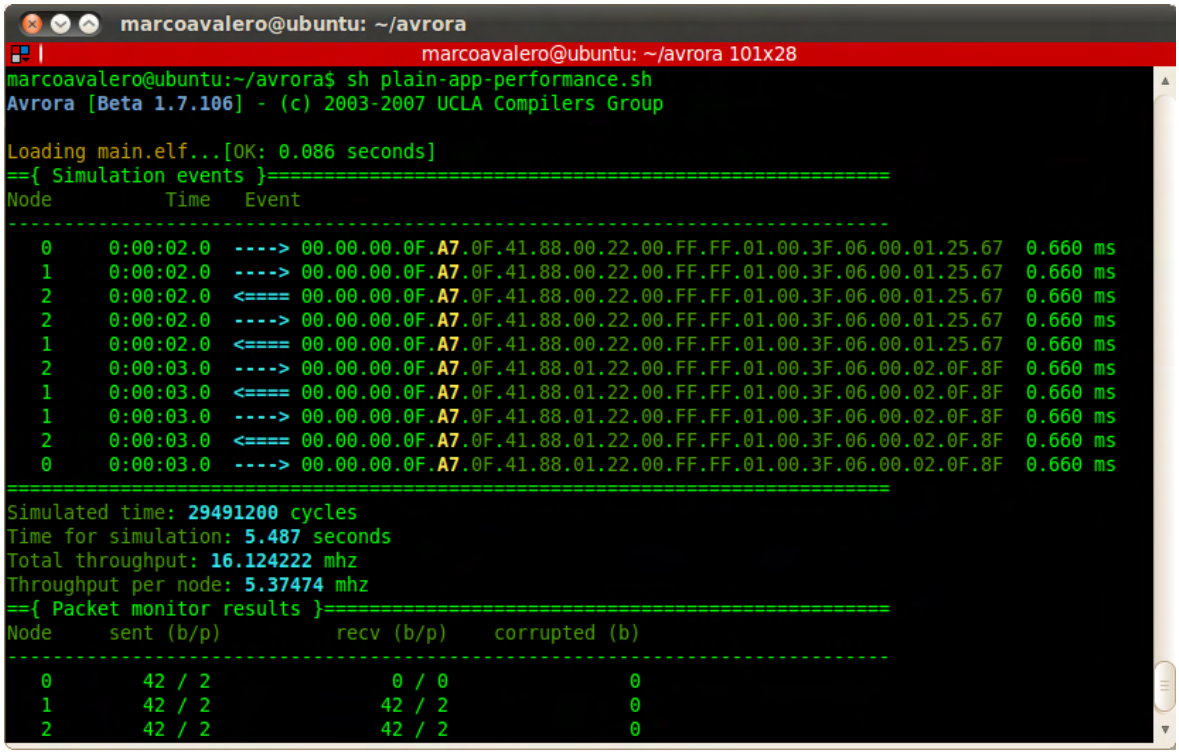

Figure 3.22 Avrora Running Plain Application Layer.

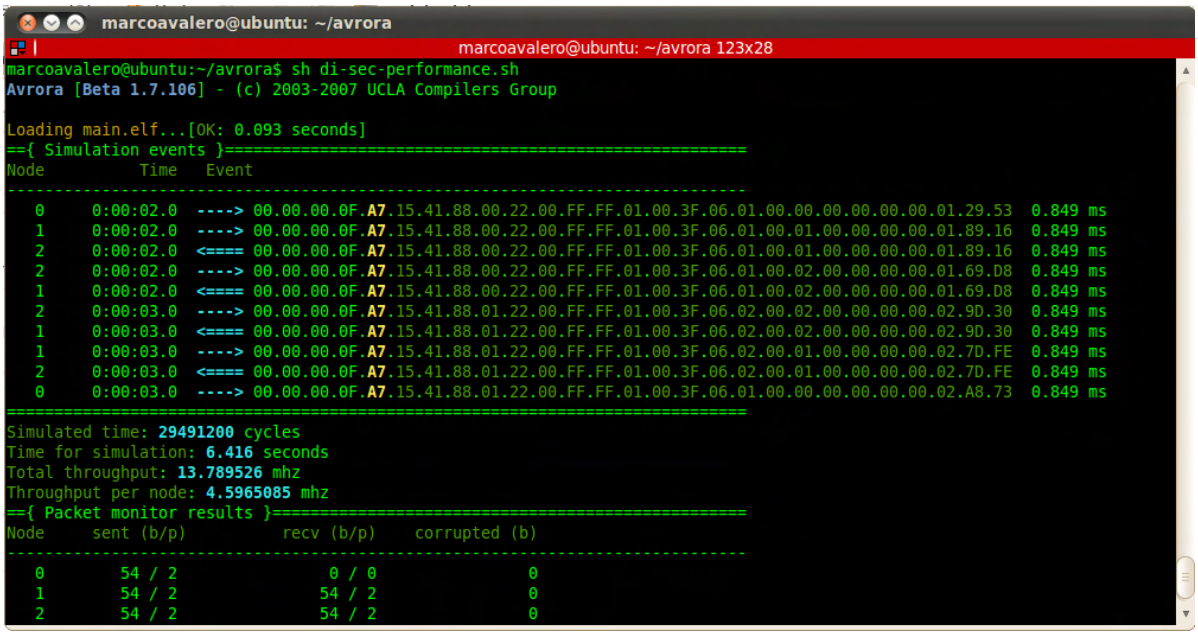

Figure 3.23 Avrora Running Di-Sec.

3.4.4 Energy Consumption Costs

Since there is no a generic model to externally calculate energy consumption in sensor nodes, we use the Avrora simulation tools to provide such calculation. Avrora has several useful monitors to allow the monitoring of the simulation as it progresses as well as the summary of some parameters at the end of the simulation such as energy consumption. The overall energy consumption is calculated by printing the energy consumed by each of the following components CPU, LEDs, External Flash, and Radio independently. Figure 3.24 shows an example of Avrora displaying the energy consumption of a sensor running Di-Sec.

To calculate energy consumption we ran simulations with the following configurations:

- (A) Plain Upper Layer.
- (B) M-Core Full.
- (C) M-Core + Jamming DDM + Sybil DDM.
- (D) M-Core + Jamming DDM + Selective Forwarding DDM + Sybil DDM + Internal DDM

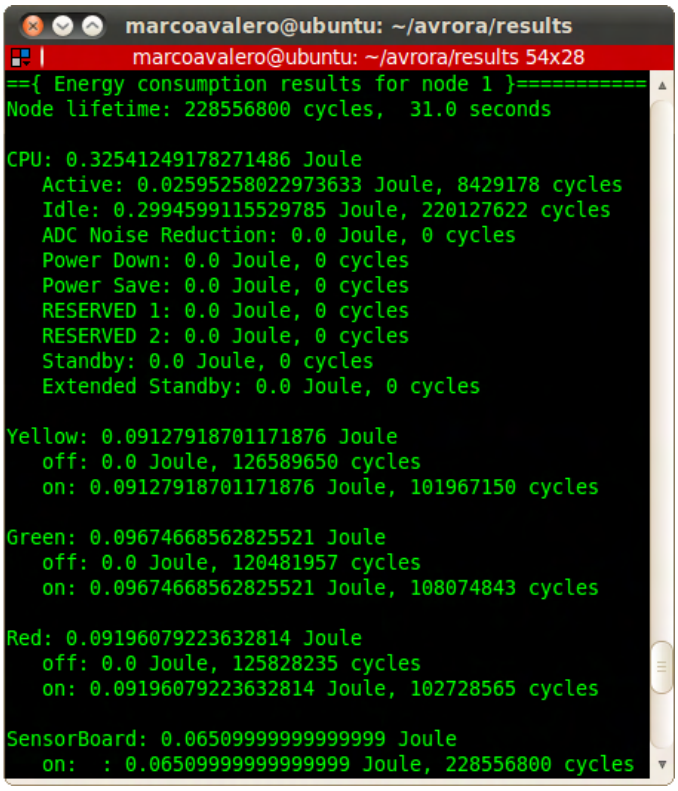

Figure 3.24 Avrora for Energy Consumption Calculation.

Table 3.23 shows the energy consumption for different configurations and simulation times. From this figure we note that Di-Sec has a minimum impact on the energy consumption: an average of 0.09% increase when using the full M-Core (with all the services), and 0.17% increase when including all the default DDMs.

Table 3.7 Di-Sec Energy Consumption (EC) in Joules

	30 Sec		**60 sec**		**120 sec**	
Configuration	EC	% Increase	EC	% Increase	EC	% Increase
A	0.325419	0	0.627033	0	1.230265	0
B	0.325724	0.093605	0.627628	0.094785	1.231439	0.095369
C	0.325865	0.136932	0.627938	0.144197	1.232082	0.1475266
D	0.325948	0.16249	0.628102	0.170371	1.232408	0.1739985

3.5 Experimental Evaluation

To test the Di-Sec framework, we created an experimental cluster scenario where we deployed 6 Tmote Sky sensors with unique IDs from 1 to 6 throughout the second floor of the Klaus Advanced Computing Building (KACB) at the Georgia Institute of Technology. The topology is shown in Figure 3.25. Node 1 is the cluster head and base station (BS) in charge of collecting all the data and the rest of the sensors communicate with the BS through multiple hops. All the nodes collect and average light measurements and transmit packets at the same rate of 1 packet every 9 seconds. It is expected that nodes 2 and 3 will have higher traffic compared with the others since they are the gateways to the base station. The overall traffic behavior and packet loss after an attack was recorded at the base station and presented in this section. Using the this topology we launched Jamming, Sybil, Selective Forwarding, and Internal attacks against the nodes in the cluster and monitor and capture the traffic to show how the cluster defends and recovers from the attacks. Each of the attacks scenarios will be explained along with the results.

Figure 3.25 Experiment Setup at the KACB.

3.5.1 Jamming Scenario

Jamming attacks can be easily accomplished by transmitting radio signals that do not follow an underlying MAC protocol. This will create collisions and interfere with the normal communication of the wireless network [42]. For our experiment, the complete jamming detection and defense mechanism was implemented inside the Di-Sec framework. Figure 3.26 shows the Jamming scenario for our experiments.

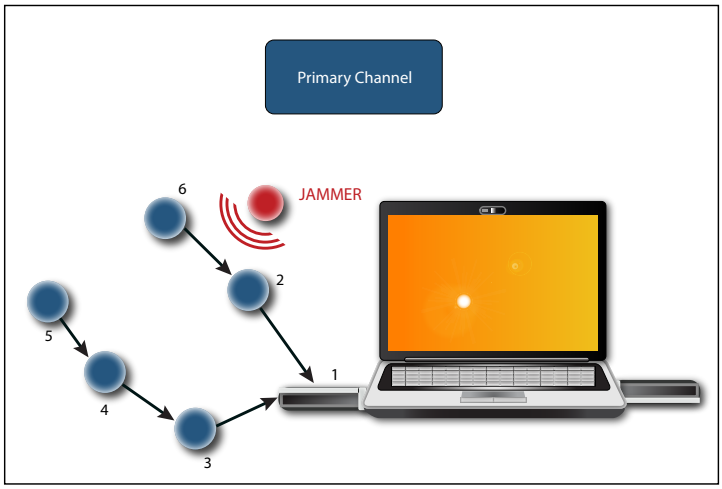

Figure 3.26 Jamming Scenario.

The M-Core services used for the detection of the jamming were the average number of received packets per second and the number of consecutive successfully transmitted packets. We defined a threshold for the maximum number of received packets per second and a minimum acceptable transmission rate (e.g., successfully transmitted packets). If the number of received packets per second is larger than our threshold or the number of consecutive successfully transmitted packets is smaller than our acceptable rate, we assume there is a jamming attack. On the other hand, our implemented defense technique consists of two parts: the monitoring period and the active jamming period. During the monitoring period, all sensors are listening to a default channel and every 5 seconds they will change and listen to the secondary channel for 400 milliseconds watching for any jamming notification. There are two ways to trigger the active jamming period. First, if the sensor detects the jamming

it will permanently change its frequency to the secondary channel and starts sending a *JAMMING_DETECTED* message every 100 milliseconds until it receives a *JAMMING_ACK* message from all of its neighbors. And second, if a sensor, not in the transmission range of the jammer, receives the *JAMMING_DETECTED* message from one of it neighbors, it locks himself in the secondary channel and starts sending the jamming notification until it propagates through the entire network. The message propagation stops when all the neighbors acknowledge the notification message with an *JAMMING_ACK* packet. After recovering from the attack, all nodes are locked in the secondary channel and resume normal operations (Figure 3.27). Note that this is only a simple solution and a different approach can be used for a better defense technique.

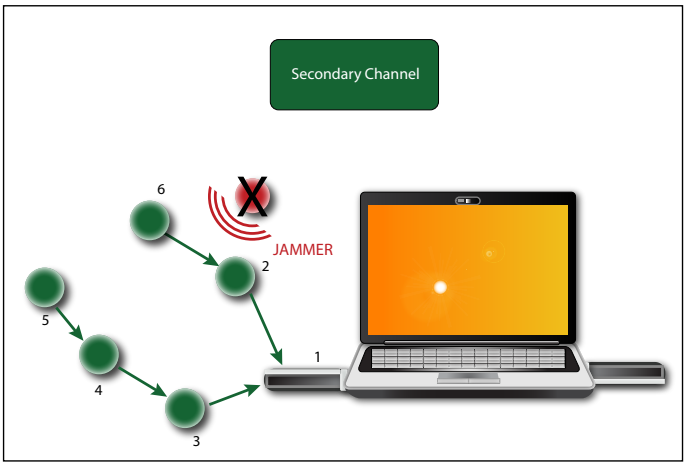

Figure 3.27 Jamming Scenario.

The flowchart of our Jamming DDM implementation is presented in figure 3.28. Figure 3.29 shows the aggregated packet count and arrivals at the base station. We see that node 3 has higher traffic than 2 which is expected since 3 is forwarding packets coming from 4 and 5 and node 2 is only forwarding packets generated at 6. Since all the nodes generate traffic at the same rate, we can perceive that the jamming attack was launched after approximately 32 packet transmissions. From the aggregated traffic received from node 2 we detected that there were approximately 28 lost packets and from node's 3 aggregated traffic we detected 38 lost packets out of a total of 420 packets transmitted during the experiment. The results show

that our implementation of jamming detection and defense using the M-Core services actually protects from jamming attacks and the packet lost due to the attack was approximately 15% for this specific scenario.

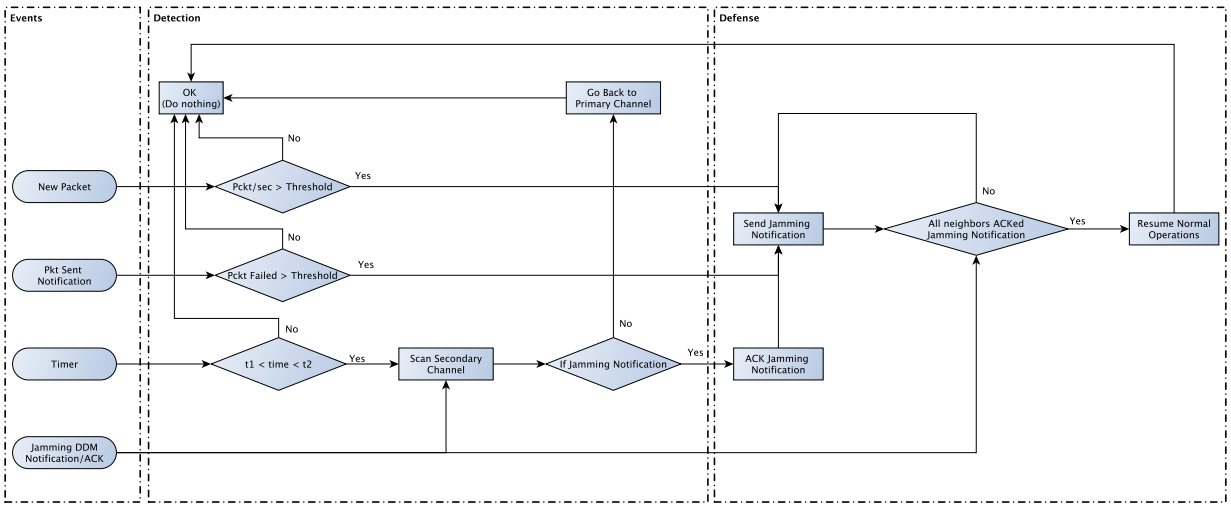

Figure 3.28 Jamming DDM Flowchart.

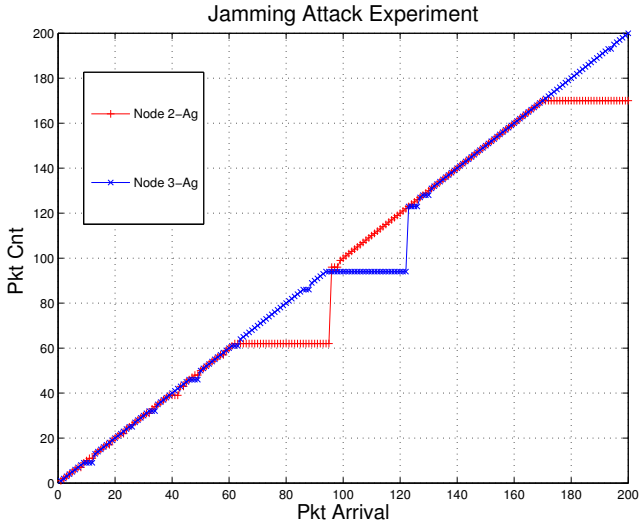

Figure 3.29 Jamming Attack Results.

3.5.2 Sybil Scenario

A case attack consists of a malicious node (Sybil node) impersonating other legitimate nodes by broadcasting messages with one or multiple node identifiers (IDs) [43]. There are different solutions for this attack including the use of shared encryption keys. In our scenario we assume that a node can be compromised and the shared key extracted, therefore, we use a RSSI-based approach to detect the Sybil attack. For the Sybil detection the M-Core provides a RSSI table containing average RSSI values for each neighbor. This table is updated every time a packet arrives to the sensor since the packet is passed to the M-Core and the RSSI value is extracted and averaged. We collect at least 10 sample packets from each neighbor to calculate the RSSI average and define an upper and lower threshold for the RSSI. Note that all values and thresholds in our framework are configurable.

For this experiment we configure the DDM to communicate directly with the upper layer (e.g., application layer) through the *ddmApp* interface to provide live feedback of the incoming packets. The setup of the experiment consists of 2 legitimate sensors: one sampler and one collector. The sampler gets light intensity measurements and transmits the values to the collector. The collector receives and displays the data. We also have 1 Sybil sensor that impersonates the sampler and injects false data into the network which is received by the collector node (Figure 3.30). When the DDM detects a Sybil message it signals an event to notify the application layer about the fraud and discard the packet.

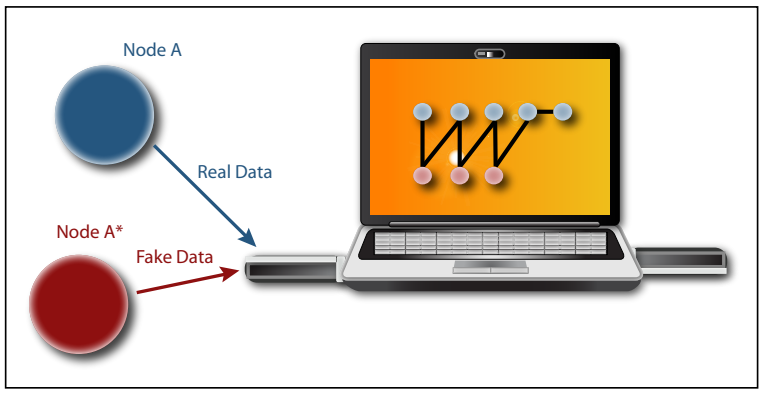

Figure 3.30 Sybil Scenario.

Figure 3.31 shows the flowchart for our Sybil DDM implementation and figure 3.32 shows the results of our experiment including the data fluctuation caused by the injections and the detection and recovery point. As seen in the figure, the Di-Sec framework is able to support Sybil scenario as well. While implementing this RSSI approach we notice that there are some false positives due to the unstable nature of the communication channels and RSSI. This is not the optimal solution to defend against Sybil attacks but we are showing that our framework provide useful services for security.

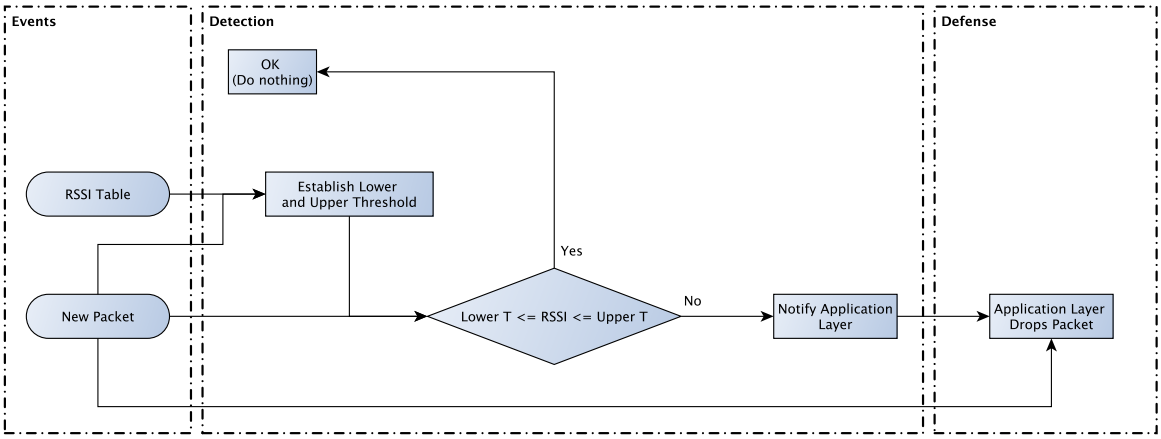

Figure 3.31 Sybil DDM Flowchart.

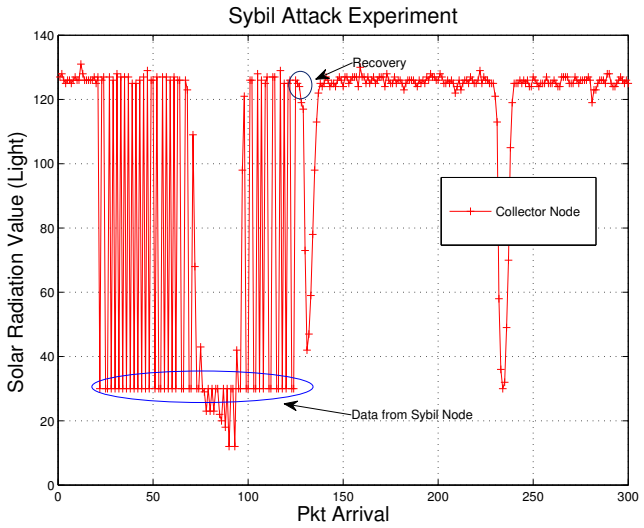

Figure 3.32 Sybil Attack Results.

3.5.3 Selective Forwarding Scenario

In the Selective Forwarding attacks, malicious nodes behave like regular nodes most of the time but selectively drop sensitive packets [44]. In order to detect this attacks we used the number of lost packets provided by the *packetcount* module in the M-Core services. The *packetcount* module creates and maintains a lost packet table which is updated every time a packet is received by the node. The table consists of four main fields: packet origin O (the node who created the message), forwarding node F (the node sending the message), previous sequence number for the combination (O, F), and number of lost packets for the combination (O, F). Using this table we can identify if one or more packets created by node A were lost by node B for example. The COMM module overhears all the messages in its transmission range and passes a copy to the M-Core module (e.g., *packetcount*). The *packetcount* parses the message and compares the Di-Sec sequence number with the previous sequence number from the table, updates the sequence number and increases the lost packets if necessary. The flowchart for our Selective Forwarding DDM implementation is presented in figure 3.33.

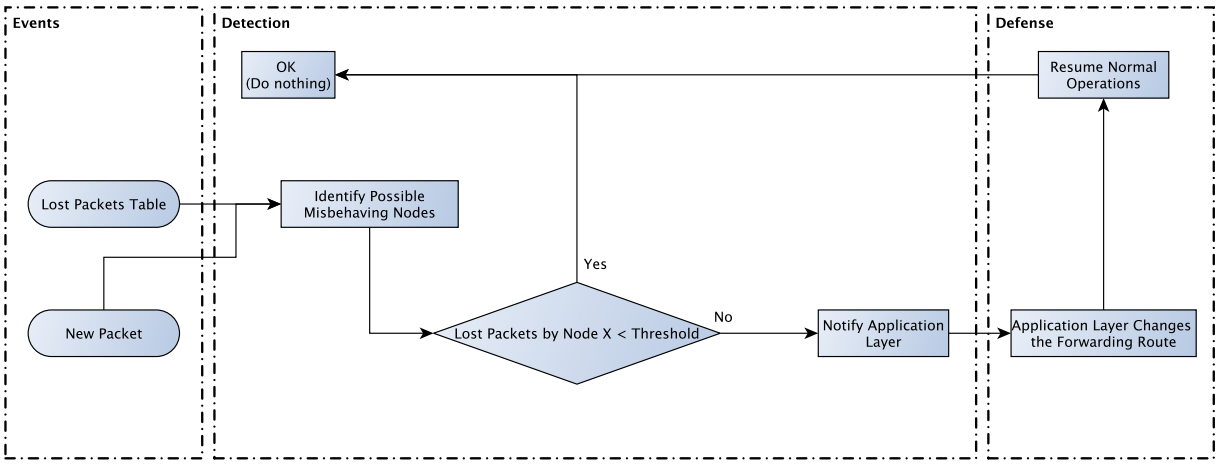

Figure 3.33 Selective Forwarding DDM Flowchart.

The scenario for the experiments is shown in Figure 3.34. For this scenario we deliberately modified node 4 to drop 66% of the received packets. We set a threshold for the maximum acceptable packets dropped by a relaying node to 25 packets. The first time the

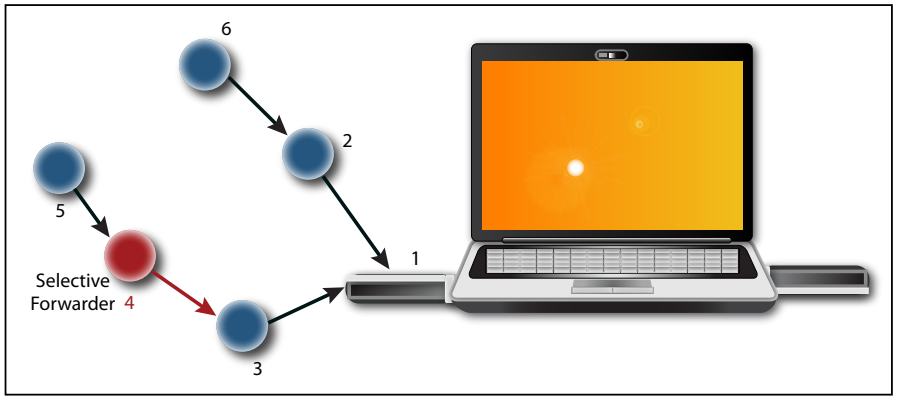

Figure 3.34 Selective Forwarding Scenario.

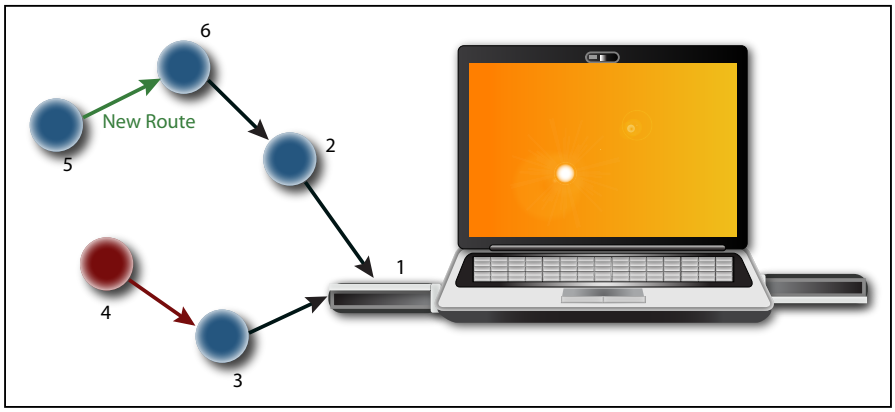

Figure 3.35 Selective Forwarding Recovery.

neighbors detect this irregularity the threshold have not been reached therefore, no action is taken. The second time the misbehavior is detected, node 5 assumes that node 4 is a selective forwarder and changes its relaying node to be node 6 to reach the base station through node 2 (Figure 3.35). Figure 3.36 shows that packets from node 5 are being dropped.

As expected, the services provided by the M-Core facilitate the implementation of security measures for selective forwarding attacks. The aggregated traffic received from node 2 and 3 at the base station is shown in Figure 3.37. If we compare Figures 3.37 and 3.29 and ignoring the jamming fluctuations on 3.29, we observe that node 2 transmitted more aggregated traffic in the selective forwarding scenario as node 3 did in the jamming scenario. These results are expected since in the selective forwarding scenario, node 5 redirected all its traffic to node 6 after detecting node 4 as a selective forwarder.

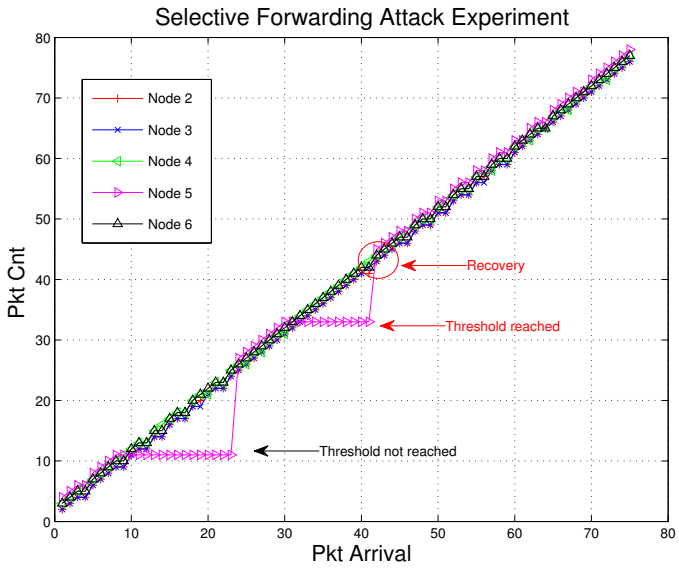

Figure 3.36 Selective Forwarding Attack Results.

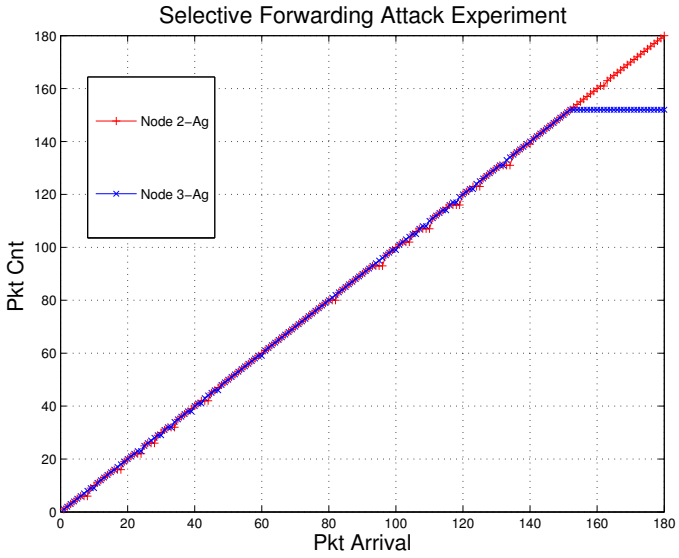

Figure 3.37 Selective Forwarding Aggregated Traffic.

3.5.4 Internal Threat Scenario

For this attack scenario, we focus on the sensing component. We created a malicious sensing component that returns forged sensed values to the application layer to simulate a misbehaving component or internal threat. We set up 3 sensors with different configurations to collect and display the total solar radiation values from the light sensor. The first sensor is not compromised and collects data from the legitimate sensing component. The second node is compromised and collects the data directly from a compromised light component (e.g., *HamamatsuS10871TsrCompromised*). The third sensor also collects the data from the malicious component but uses Di-Sec sensing component to verify the collected values. During the experiment, we placed the three sensors next to each other in our laboratory and turn the lights off and on repeatedly. A low measurement value activates the malicious module and all subsequent measurements returned by the module are fixed to a value of 10. Figure 3.39 presents results of this experiment and shows that the compromised application relying on our framework services identifies and recovers from the malicious attack while the compromised node keeps recording a value of 10 during the rest of the experiment.

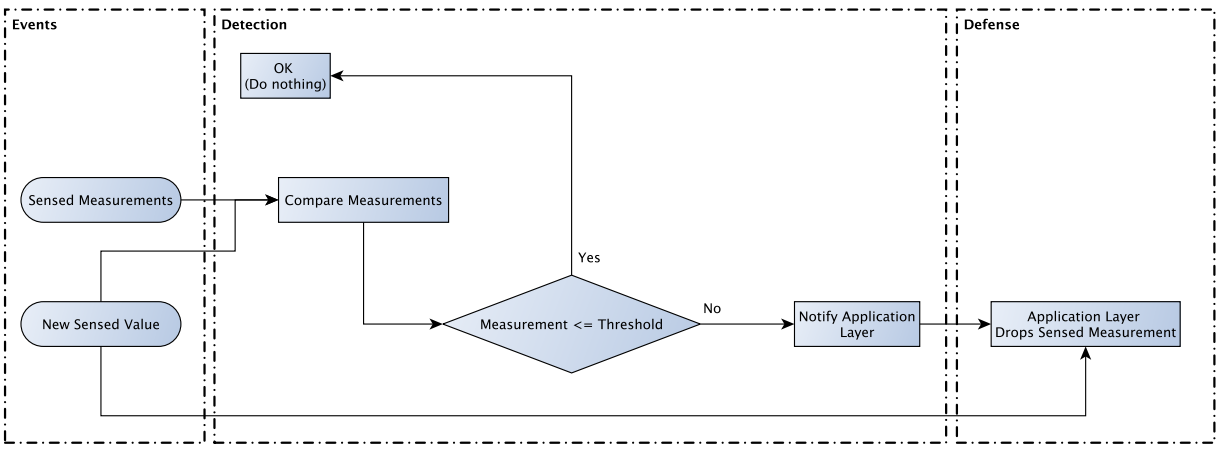

Figure 3.38 Internal DDM Flowchart.

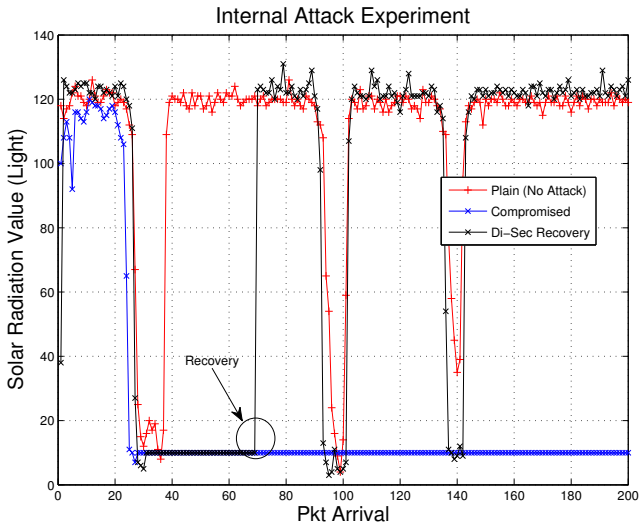

Figure 3.39 Internal Attack Results.

3.5.5 Combined Attacks

For the sake of completeness we combined and launched two of the previous attacks in a single experiment. Specifically, we combined Jamming and Selective Forwarding attacks to demonstrate that Di-Sec framework successfully defends and recovers from any combination of attacks. Figure 3.40 shows the overall traffic behavior per node and aggregated at the gateways (node 2 and 3). Figure 3.41 highlights the impact of the Selective Forwarding attack on the individual traffic. Again node 5 traffic is dropped by the selective forwarder (node 4) and after detection the route is changed and all the traffic is redirected to the base station via node 6. Right after the first attack, we launched the jamming attack which was also handled by our framework to finally resume the normal communications.

As we can see from the previous scenarios, the Di-Sec framework provides the services required to identify and defend against different attacks. Moreover, the M-Core architecture allows more sub-modules (services) to be easily developed and integrated into the framework, according to the requirements of the DDMs.

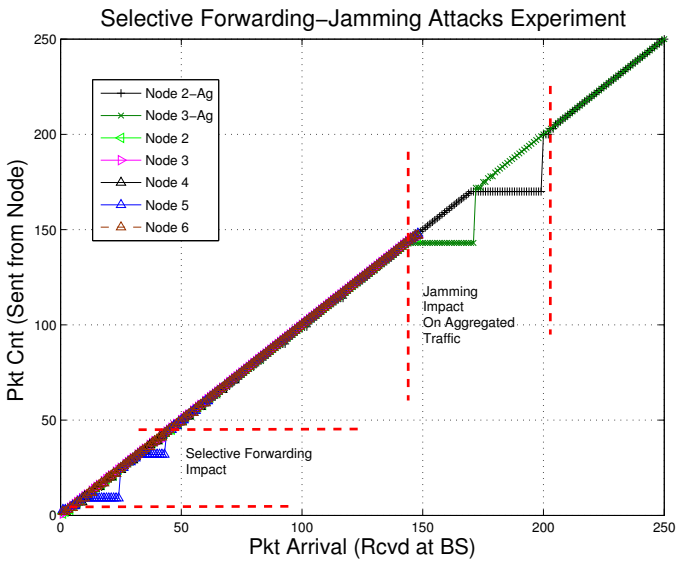

Figure 3.40 Combined Selective Forwarding and Jamming Attacks.

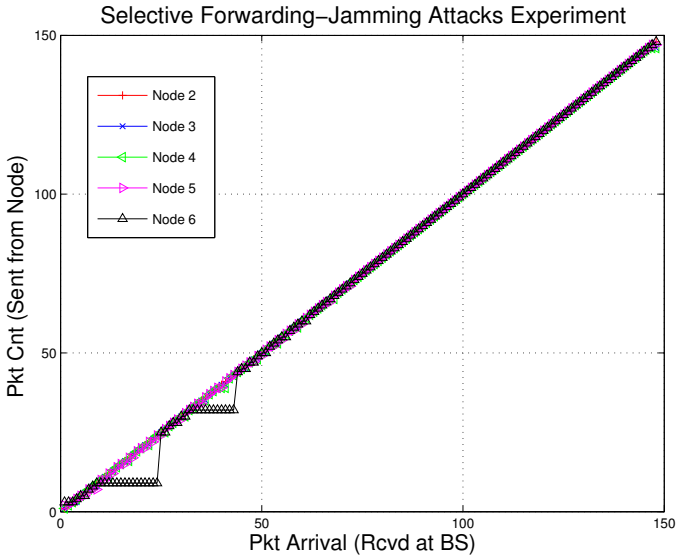

Figure 3.41 Impact of the Selective Forwarding.

3.6 MCL Evaluation

To evaluate the benefits of the Di-Sec and the MCL we describe the amount of work (i.e., costs) required to develop new DDMs with and without our framework. We divided the development costs in two categories: learning costs and implementation costs.

3.6.1 Learning Costs

For the development of new WSN software programs in TinyOS, one needs to understand the concepts of *modules, configurations, interfaces* and *wiring*. *Modules* (or *components*) are the basic building blocks of a TinyOS program since they implement the program's executable logic and include some specific behaviors. For one module to be able to call and use the functions provided by another module, we need *configuration* files to map the set of provided functions in one component to a set of required functions in another component (*interfaces*). In TinyOS, connecting two components through an interface is called *wiring*. Whenever a developer wants to create a new program, he must define the program requirements and then identify the components that provide the required functionality. Once the components are identified, the developer needs to find the interfaces to communicate with those components and implement their events and learn how to use their commands. Moreover, when some functionalities are not implemented in the required components, the developer has to implement it himself. For instance, to obtain the RSSI value from a packet, a developer needs to identify the radio transceiver used by the sensor, discover the structure of the message provided by the radio, parse the message, and extract the RSSI value.

Given that the M-Core provides all the required information (services) for the development of new programs within a single component, a developer only needs to add an M-Core component in their programs and call any service directly through the M-Core. All the message and sensing values parsing is done inside the M-Core to reduce the developer's effort. Therefore, the amount of work required to create new DDMs is much smaller when using Di-Sec since it reduces the complexity of finding and learning how to use new components.

Table 3.8 Implementation Comparison

	No Di-Sec	**Di-Sec & No MCL**	**Di-Sec & MCL**
Lines of code	20 for new component	20 for new component	5 for new program
	8 for new configuration	8 for new configuration	
	4 for component wiring	4 for M-Core wiring	
	8 for every additional event per interface	8 for every additional event per interface	
	55 for RSSI extraction and tables		
Total	**91**	**36**	**5**
Files to modify	2 modules	1 New module	1 MCL program
	1 configuration	1 configuration	1 module
	1 interface	1 M-Core	
	1 headers		
Total	**5**	**3**	**2**

3.6.2 Implementation Costs

The Di-Sec framework provides a simple architecture that eases the design of new DDMs. Di-Sec allows the developers to create multiple DDMs running individually or in parallel, and it is also possible to have a stack of layers using the M-Core services. To compare the implementation costs, we discuss the amount of work to create a simple DDM that maintains a table with the node neighbors' RSSI information. The evaluation is based on the number of lines of code to write and the number of files to modify for a basic code setup of a new program. As seen in Table 3.8, using Di-Sec and MCL only takes 5 lines of code to develop our simple program compared to 91 lines of code without Di-Sec. For our evaluation we used a simple scenario, but savings are amplified when developing more complex DDMs.

PART 4

MCORE FOR RAPID SENSOR APP DEVELOPMENT

In this section, we discuss the use of the Monitoring-Core (M-Core) for the development of new application programs in different domains. The information provided by the M-Core in the form of services can be accessed by multiple processes simultaneously and can be used to develop a wide range of software solutions. Similar to how Rhodes Framework [45] provides modules and procedures for building native applications for smartphones, the M-Core can be used to facilitate the design and reduce development time of new WSN software.

Figure 4.1 shows the M-Core architecture used to create Application Modules (AMs), where we also illustrate five application modules (AMs) implemented at the Communications Assurance & Performance Group at Georgia Tech. The details of these implemented programs across multiple applications domains (security, networking / routing, host application) are presented in section 4.1.

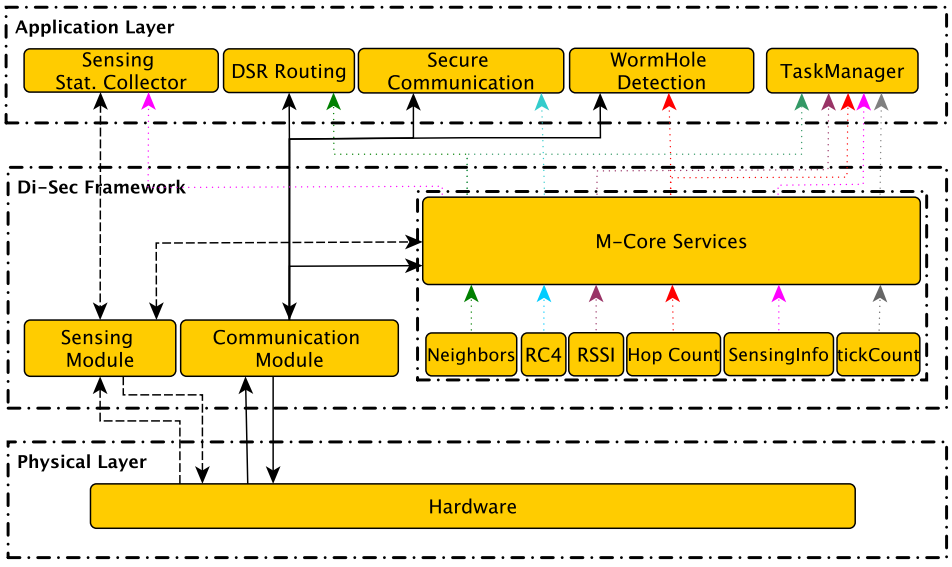

Figure 4.1 M-Core Architecture.

4.1 WSN Application Development for Different Domains

To show the M-Core's functionality we implemented several application modules (AMs) in multiple application domains (security, networking / routing, host application). For the security domain we have implemented a stand-alone application and a collaborative attack detection and defense. The first is a secured communication application, and the second is the Wormhole attack detection and defense. For the network / routing domain, we implemented the Dynamic Source Routing (DSR) protocol [46], which is used in wireless ad-hoc and sensor networks. And, for the host application domain, we implemented a task manager for sensors that we call *TinyOSTaskManager*.

4.1.1 Security Domain

Secure Communication (A Stand-alone Application) The M-Core provides the RC4 stream cipher [40] encryption and decryption algorithm as a service by providing the *encryptI* interface. Since the RC4 ciphers are generated using a symmetric operation, the AM can use the same *encrypt* function call of the *encryptI* interface to encrypt and decrypt the data. To perform this encryption or decryption operation, the AM passes the secret key, size of secret key, plaintext and size of plaintext as input to the *encrypt* function call. After encrypting the plaintext, the RC4 service will overwrite the input plaintext with the resultant ciphertext data. Hence applications have to make only one function call before sending the data and after receiving the data to encrypt and decrypt the data stream respectively. This RC4 encryption is a simple algorithm which is platform independent and can work with all versions of TinyOS. Using this sub-component, a sample Radio Count Application implementing an 8-bit counter secures its broadcast data, which has the counter value as payload. Table 4.1 shows the sizes of our program with and without M-Core.

Table 4.1 Secure Communication Program Size

	M-Core	**Plain**
ROM (bytes)	18718	11598
RAM (bytes)	1293	310

Wormhole Detection and Defense (A Collaborative Program) The Wormhole attack [47], is one of the potential threats targeting ad hoc and sensor networks. To launch this kind of attack, an adversary connects two distant points in the network using a *tunnel* with low latency, to deceive distant sensors and make them believe they are neighbors. Once the routes are established and the network traffic starts using the wormhole link, the attacker can retrieve sensitive information or disrupt the network communication.

For this experiment, the complete wormhole detection and defense mechanism was implemented using the M-Core. As shown in figure 4.2, the scenario consisted of four legitimate sensor nodes and an attacker node placed near node 4. Node 1 is the cluster head and the base station is in charge of collecting all the data and the rest of the sensors communicate with the base station through multiple hops. Using the attacker node, the wormhole link (i.e., faster link) is established between node 4 and the cluster head (node 1).

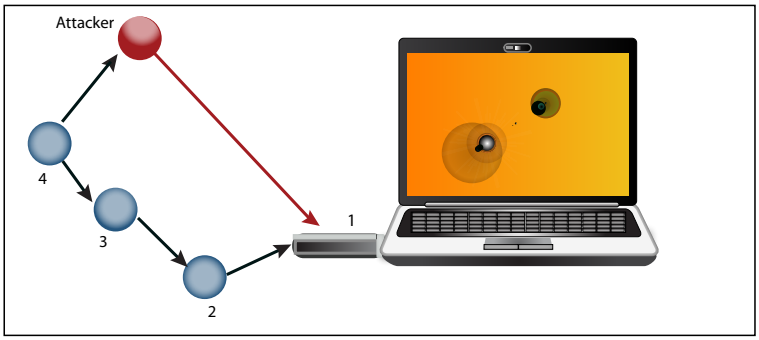

Figure 4.2 Wormhole scenario.

For the wormhole detection mechanism, we use the *hop-count service* provided by the M-Core. During the initialization of the topology, the *hop-count* service is called by the cluster head and all the nodes estimate their corresponding hop-counts from the cluster head. The hop-count service can also be called periodically or whenever there is a change in the topology (new nodes added). At the end of each hop-count service, all nodes will update their corresponding *node id* of their preceding node towards the cluster head and cache it.

In the normal communication procedure, each node which initiates a transmission towards the cluster head sets its *current hop* field to its estimated *hop-count* and sends the

packet. Each subsequent node decrements this *current hop* value and compares with its own estimated *hop-count*. It forwards the packet only if both the values match. If not, it immediately broadcasts an alert message. On reception of the alert message, the nodes revert back to their cached *preceding node id* for sending packets towards the cluster head.

Whenever a node receives an advertisement packet for a shorter (or faster) route towards the cluster head, it starts sending packets to the cluster head through this new route. However, it still holds the previous *preceding node id* in its cache. Now, when the attacker (node placed near node 4) advertises a shorter route to the cluster head, node 4 sends packets to the cluster head through the attacker. When node 1 receives these packets, it identifies a mismatch between its own *hop-count* and the *current hop* associated with each of these packets. It broadcasts an alert message once the threshold (number of packets arriving with a *hop-count* mismatch) is reached, which directs node 4 to revert back to its cached *preceding node id* and resume transmission. Now, the route through the attacker is avoided and also both ends of the wormhole link are identified using the *hop-count* and *current hop* parameters of the node which initiated the alert message.

4.1.2 Network / Routing Domain

Dynamic Source Routing (A Network Routing Protocol) The Dynamic Source Routing (DSR) protocol [46] is a type of reactive (on-demand) routing protocol for multi-hop wireless ad-hoc and sensor networks, which uses source routing. Source routing is a method in which the sender specifies either the complete or partial path a packet has to take to reach the destination. Whenever an application needs to route a packet to a particular destination, the DSR protocol looks into its *route cache* for any cached path to the requested destination. If the route for the requested destination is not found in the route cache, then a *route discovery* process is initiated by the DSR protocol in the source node. The *route discovery* process of the Dynamic Source Routing protocol was implemented using the *neighbors* service provided by the M-Core. This *neighbors* service gives the list of Node ID's of all reachable neighbors from the current node. In the modified DSR route discovery implementation, using M-Core,

the intermediate node queries the *neighbors* service to determine whether the destination is an immediate neighbor of this current node. If the destination is an intermediate neighbor, the intermediate node generates the route reply message to the source instead of forwarding the request. So the intermediate nodes which have the destination node as their neighbor will generate the route reply instead of forwarding the route request, optimizing the route discovery process. Table 4.2 shows the sizes of our program with and without M-Core.

Table 4.2 DSR Program Size

	M-Core	**Plain**
ROM (bytes)	20338	18718
RAM (bytes)	2043	1293

4.1.3 Host Application Domain

***TinyOSTaskManager* (A Host Application)** using the M-Core services we also developed a task manager-like program that allows one to monitor different activities on the sensor (*TinyOSTaskManager*). The monitor application collects information about the CPU ticks, number of neighbors, number of active M-Core services, packets in/sec, packets out/sec, and total packets received. The status of the different parameters are displayed in a java interface. Figure 4.3 shows a screen capture of the *TinyOSTaskManager*.

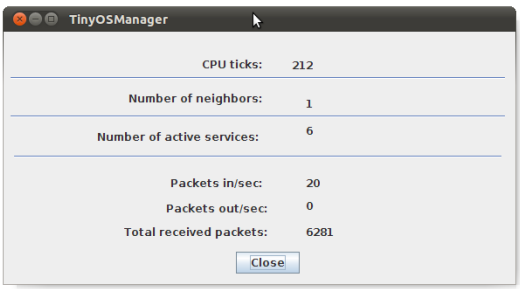

Figure 4.3 TinyOSTaskManager.

The applications from multiple domains illustrated above are based off of existing solutions in the literature and were presented as examples. The main contribution is to show that they were quickly developed using the M-Core and MCL.

4.2 Other Related Works for WSN Application Development

In this final part of the dissertation we demonstrated that the M-Core can be used as a middleware for developing new applications for sensor networks. We are aware that there exists other works that proposed a general solution for WSN development as well, and we briefly reference those works in this section.

In [48], the authors proposed OASIS, a programming framework for service-oriented sensor networks. OASIS also follows a service oriented middle-ware approach for rapid application development similar to the M-Core, but they don't offer those services in a single configuration file as the M-Core does. Additionally, they support service sharing between the motes using a Service Discovery algorithm.

The work in [49] discusses the trends and challenges of designing and developing solutions for WSN using service oriented middleware. Some of the middleware they analyze are: SStreaMWare [50], USEME [51], MiSense [52], SOMDM, and others.

PART 5

CONCLUSIONS

In this work, we introduced a comprehensive security framework for WSNs called Di-Sec. The goal of our architecture design was to create highly modular, flexible, and expandable framework to provide security against different attacks.

The overall contribution of this work is to realize an architecture that can be leveraged by researchers to expedite the development of sensor defense mechanisms and to allow their parallel execution. *We want to do for sensor security researchers what metasploit has done for hackers.*

Along with Di-Sec we also created a domain specific language called MCL to interact with the framework. Using MCL, a user can implement new defense mechanisms without the overhead of learning the details of the underlying software architecture (i.e., TinyOS, Di-Sec). We study the performance of the framework in terms of storage costs (RAM and ROM), CPU overhead, and communication overhead. We also implemented detection and defense mechanisms against Jamming, Sybil, Selective Forwarding, and Internal attacks and show through experimentation that Di-Sec framework successfully defends and recovers from those attacks. Moreover, we demonstrated that our framework, specifically the M-Core can be used to easily develop applications in other WSN domains.

REFERENCES

[1] M. Iqbal and H. B. Lim, "A cyber-physical middleware framework for continuous monitoring of water distribution systems," in *Proc. of the 7th ACM SenSys*, 2009, pp. 401–402.

[2] "The Internet of Things," http://www.theinternetofthings.eu/.

[3] "Planetary Skin," http://www.planetaryskin.org/home.

[4] J. Bort, "10 technologies that will change the world in the next 10 years," http://www.networkworld.com/news/2011/071511-cisco-futurist.html, 2011.

[5] H. Saxena, C. Ai, M. Valero, Y. Li, and R. Beyah, "Dsf - a distributed security framework for heterogeneous wireless sensor networks," in *MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010*, 31 2010-nov. 3 2010.

[6] H. S. N. P. L. An; and D. Wenliang, "Seluge: Secure and dos-resistant code dissemination in wireless sensor networks," in *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks.* Washington, DC, USA: IEEE Computer Society, 2008, pp. 445–456.

[7] I. Krontiris, T. Dimitriou, T. Giannetsos, and M. Mpasoukos, "Intrusion detection of sinkhole attacks in wireless sensor networks," in *ALGOSENSORs 2007*, July 2007.

[8] M. A. Hamid, M. Mamun-Or-Rashid, and C. S. Hong, "Routing security in sensor network: Hello flood attack and defense," in *ICNEWS '06: Proceedings of First International Conference on Next-Generation Wireless Systems 2006*, 2006.

[9] P. Traynor, R. Kumar, H. B. Saad, G. Cao, and T. L. Porta, "Liger: implementing efficient hybrid security mechanisms for heterogeneous sensor networks," in *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services.* New York, NY, USA: ACM, 2006, pp. 15–27.

[10] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: Link layer security architecture for wireless sensor networks," *SenSys '04*, 2004.

[11] A. Pathan, H.-W. Lee, and C. S. Hong, "Security in wireless sensor networks: issues and challenges," in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, vol. 2, feb. 2006, pp. 6 pp. –1048.

[12] A. Francillon and C. Castelluccia, "Code injection attacks on harvard-architecture devices," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2008, pp. 15–26.

[13] R. K.C., V. Subramanian, A. S. Uluagac, and R. A. Beyah, "SIMAGE: secure and link-quality cognizant image distribution for wireless sensor networks," in *Submitted to IEEE Globecom 2012 - Communication & Information System Security*, Anaheim, California, USA, 2012.

[14] GT-CAP, "Di-Sec: Distributed Security Framework for Heterogeneous Wireless Sensor Networks ," http://www.ece.gatech.edu/cap/disec/, 2011.

[15] X. Chen, K. Makki, K. Yen, and N. Pissinou, "Sensor network security: a survey," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 2, pp. 52 –73, quarter 2009.

[16] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *IEEE Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.

[17] F. Hu and N. K. Sharma, "Security considerations in ad hoc sensor networks," *Elsevier's AdHoc Networks Journal*, vol. 3, no. 1, pp. 69–89, January 2005.

[18] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, "A survey on jamming attacks and countermeasures in wsns," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 4, pp. 42 –56, quarter 2009.

[19] T. Roosta, S. Shieh, and S. Sastry, "Taxonomy of security attacks in sensor networks," in *The First IEEE International Conference on System Integration and Reliability Improvements*, Hanoi, Vietnam, Dec 2006.

[20] J. Newsome, E. Shi, D. Song, and A. Perrig, "The sybil attack in sensor networks: analysis defenses," in *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, april 2004, pp. 259 – 268.

[21] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, vol. 1, no. 2-3, pp. 293–315, September 2003.

[22] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," in *Communications of the ACM*, vol. 47, no. 6, June 2004, pp. 53–57.

[23] R. D. Pietro, L. Mancini, A. Mei, A. Panconesi, and J. Radhakrishnan, "How to design connected sensor networks that are provably secure," *Securecomm*, pp. 89–100, 2006.

[24] IEEE, *IEEE 802.15.4 Standard Specification for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2006.

[25] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: a secure sensor network communication architecture," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks.* ACM, 2007, pp. 479–488.

[26] Q. Xue and A. Ganz, "Runtime security composition for sensor networks (securesense)," in *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, vol. 5, oct. 2003, pp. 2976 – 2980 Vol.5.

[27] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "Spins: security protocols for sensor networks," *Wirel. Netw.*, vol. 8, no. 5, pp. 521–534, 2002.

[28] A. D. Wood and J. A. Stankovic, "Amsecure: secure link-layer communication in tinyos for ieee 802.15.4-based wireless sensor networks," in *Proceedings of*

the 4th international conference on Embedded networked sensor systems, ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 395–396. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182873

[29] M. Healy, T. Newe, and E. Lewis, "Security for wireless sensor networks: A review," in *Sensors Applications Symposium, 2009. SAS 2009. IEEE*, feb. 2009, pp. 80 –85.

[30] X. Wang, S. Chellappan, W. Gu, W. Yu, and D. Xuan, "Policy-driven physical attacks in sensor networks: modeling and measurement," in *IEEE Wireless Communications and Networking Conference (WCNC 2006)*, vol. 2. IEEE, April 2006, pp. 671– 678.

[31] S. Shin, T. Kwon, G.-Y. Jo, Y. Park, and H. Rhy, "An experimental study of hierarchical intrusion detection for wireless industrial sensor networks," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 744 –757, nov. 2010.

[32] A. Stetsko, L. Folkman, and V. Matyass and, "Neighbor-based intrusion detection for wireless sensor networks," in *Wireless and Mobile Communications (ICWMC), 2010 6th International Conference on*, sept. 2010, pp. 420 –425.

[33] Z. Yu and J. Tsai, "A framework of machine learning based intrusion detection for wireless sensor networks," in *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC 08. IEEE International Conference on*, june 2008, pp. 272 –279.

[34] M. J. Ocean and A. Bestavros, "Wireless and physical security via embedded sensor networks," in *Proceedings of the first ACM conference on Wireless network security*, ser. WiSec 08, 2008, pp. 131–139.

[35] P. Krishnamoorthy and M. Wright, "Towards modeling the behavior of physical intruders in a region monitored by a wireless sensor network," in *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, ser. AISec '10. ACM, 2010.

[36] Y. Keung, B. Li, and Q. Zhang, "The intrusion detection in mobile sensor network," in

Proceedings of the eleventh ACM international symposium on Mobile ad hoc networking and computing, ser. MobiHoc '10. ACM, 2010, pp. 11–20.

[37] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," *Proceedings of the 2nd International conference on Embedded Networked Sensor Systems*, pp. 81–84, 2004.

[38] D. P. K. H. J. W. C. D. C. C. C. D. E., "Securing the deluge network programming system," in *Proceedings of the 5th international conference on Information processing in sensor networks*, ser. IPSN '06. New York, NY, USA: ACM, 2006, pp. 326–333. [Online]. Available: http://doi.acm.org/10.1145/1127777.1127826

[39] W. Munawar, M. H. Alizai, O. Landsiedel, and K. Wehrle, "Dynamic tinyos: Modular and transparent incremental code-updates for sensor networks," in *Communications (ICC), 2010 IEEE International Conference on*, may 2010, pp. 1 –6.

[40] B. A. Forouzan, *Cryptography & Network Security (1st edition)*. McGraw-Hill, 2007.

[41] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1147685.1147768

[42] W. Xu, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *In ACM MOBIHOC*, 2005, pp. 46–57.

[43] M. Demirbas and Y. Song, "An rssi-based scheme for sybil attack detection in wireless sensor networks," in *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, ser. WOWMOM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 564–570. [Online]. Available: http://dx.doi.org/10.1109/WOWMOM.2006.27

[44] B. Yu and B. Xiao, "Detecting selective forwarding attacks in wireless sensor networks," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, april 2006, p. 8 pp.

[45] Rhodes, "Rhodes:cross-platform mobile app development framework," http://rhomobile.com.

[46] D. B. Johnson, D. A. Maltz, and J. Broch, "Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks," in *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5.* Addison-Wesley, 2001, pp. 139–172.

[47] Y.-C. Hu, A. Perrig, and D. Johnson, "Wormhole attacks in wireless networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 2, pp. 370 – 380, feb. 2006.

[48] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits, "Oasis: A programming framework for service-oriented sensor networks," in *Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*, jan. 2007, pp. 1 –8.

[49] N. Mohamed and J. Al-Jaroodi, "Service-oriented middleware approaches for wireless sensor networks," in *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, jan. 2011, pp. 1 –9.

[50] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro, and V. Olive, "Sstreamware: a service oriented middleware for heterogeneous sensor data management," in *Proceedings of the 5th international conference on Pervasive services*, ser. ICPS '08. New York, NY, USA: ACM, 2008, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1387269.1387290

[51] E. Caete, J. Chen, M. Diaz, L. Llopis, and B. Rubio, "Useme: A service-oriented framework for wireless sensor and actor networks," in *Applications and Services in Wireless Networks, 2008. ASWN '08. Eighth International Workshop on*, oct. 2008, pp. 47 –53.

[52] J. M. Prinsloo, C. L. Schulz, D. G. Kourie, W. H. M. Theunissen, T. Strauss, R. Van Den Heever, and S. Grobbelaar, "A service oriented architecture for wireless sensor and actor network applications," in *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, ser. SAICSIT '06. Republic of South Africa: South African Institute for Computer Scientists and Information Technologists, 2006, pp. 145–154. [Online]. Available: http://dx.doi.org/10.1145/1216262.1216278