# Di-Sec: A *Di*stributed *Sec*urity Framework for Heterogeneous Wireless Sensor Networks

Marco Valero*†, Sang Shin Jung†, A. Selcuk Uluagac†, Yingshu Li* and Raheem Beyah†
*Department of Computer Science
Georgia State University
Atlanta, Georgia 30303,USA
{mvalero, yli}@cs.gsu.edu
†GT CAP Group, The School of ECE
Georgia Institute of Technology
Atlanta, GA 30332, USA
{sjung36@, selcuk@ece., rbeyah@ece.}gatech.edu

***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 the 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 statistics relevant for the defense mechanisms. The M-Core allows for the monitoring of both internal and external threats and supports the execution of multiple detection and defense mechanisms (DDMs) 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 basic functionality of Di-Sec by implementing and simultaneously executing DDMs for attacks at various layers of the communication stack (i.e., Jamming, Selective Forwarding, Sybil, and Internal attacks).**

***Index Terms*—Wireless Sensor Network Security, Distributed Security Framework, M-Core Control Language (MCL)**

## I. Introduction

Wireless Sensor Networks (WSNs) are no longer a nascent technology and are deployed in diverse application domains (e.g., health care, military, environment). Moreover, with recent initiatives such as *Cyber-Physical Systems* [1], *Internet of Things* [2], and *Planetary Skin* [3], sensor-based applications have gained new momentum 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].

Over the last decade, the WSNs research community has identified many unique security threats. There has been a tremendous effort to build mechanisms to defend against these threats and a myriad of security solutions have been proposed. However, the trend with different security schemes so far has been to focus on defending against *individual* threats/attacks rather than on developing 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 they are today.

However, 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 attacks (e.g., Sybil, Selective Forwarding). The traditional approach to securing WSNs requires the unrealistic assumption 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, the network must be prepared to defend against one or more attackers launching single or multiple attacks simultaneously at different places in the network. Keeping this realistic threat model in mind, WSNs must be prepared to defend against all known attacks at any given time [5].

Hence, in this work, we present a comprehensive security framework, Di-Sec, that could 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 [6]. In fact, our earlier work in this domain [5] also studied this problem. However, it was at a macroscopic level focusing on overall challenges with the framework at the network level. The work was evaluated using simulations. Therefore, in this work, we designed, developed and implemented a framework that can provide generic security to WSNs using real sensors, with the focus at the node level.

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 (cluster-heads) introduces high deployment costs.

Di-Sec was built with highly modular components and a flexible architecture atop the TinyOS operating system. The novel architecture of Di-Sec includes two fundamental components: the Monitoring-Core (M-Core) and Detection and Defense Modules (DDMs). Conceptually, the M-Core is the the heart of Di-Sec and is an extensible and lightweight layer that is 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. The M-Core can support the execution of new or existing DDMs against different threats in parallel. On the other hand, DDMs are specific attack detection 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 utilizing the M-Core services. Furthermore, to easily use the Di-Sec framework to access and activate the services provided by M-Core, we have created a new domain specific language named 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.

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 sensing components 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 simultaneously implement four detection and defense mechanisms that span different layers of the sensor communication stack (i.e., Jamming, Sybil, Selective Forwarding, and Internal attacks).We show that Di-Sec's flexible and modular architecture can be easily extended to defend against new and forthcoming attacks.

Our contributions in this paper are the following: We (1) realize an extensible architecture that can rapidly allow the implementation and execution of multiple attack defense and detection mechanisms simultaneously; (2) present a new domain specific language to significantly simplify the development of new defense mechanisms; and (3) illustrate scenarios for single and multiple simultaneous attacks and how Di-Sec can host multiple defense mechanisms to stop the attacks. Note that the code and more information about the Di-Sec are available online at [7].

The rest of the paper is organized as follows. Related work is discussed in Section II. Section III presents the network and threat model and also describes the overview of the Di-Sec framework. The details of the framework are explained in Section IV. The M-Core Control Language is formally introduced in Section V and a sample usage is also given in the same section. The performance evaluation of Di-Sec on real sensors is presented in Section VI. We conclude the paper and discuss the future work in Section VII.

## II. Related Work

The issue of providing security for WSNs is a significant and open research problem which has been discussed extensively in earlier studies. Some studies provide classifications and address the relevant issues from a general perspective [8], [9]. Other studies focus only on a particular layer of protocols [8], [10], [11] identifying various common attacks like Jamming (physical layer), Sybil (MAC layer), and 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.

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. In [12], 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 sensor networks 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 [13], 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 [12] the focus is on the performance (i.e., the accuracy of detection) of the IDS rather than a generic security framework. In [14] 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 [15], 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 [16], [17] 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 to develop solutions for any type of 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.

Although the general idea of Di-Sec was discussed in our earlier work [5], the focus was more on theoretical aspects at a macroscopic level focusing on overall challenges with the framework at the network level, and the work was evaluated using simulations. However, Di-Sec was implemented and tested on real sensors to evaluate its feasibility and performance, with the focus at the node level. Moreover, along with Di-Sec, a new user-friendly domain-specific language called the M-Core Control Language (MCL) was developed to expedite the development of sensor defense mechanisms.

## III. System Overview

In this section, we introduce the network model, the threat model, and briefly describe the overall architecture of the Di-Sec framework.

### A. 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 CH for further processing, analysis, and storage.

### B. Threat Model and Assumptions

We assume the malicious node is structurally the same as the regular nodes, and possess hardware capabilities either similar to or higher than that of legitimate nodes. We assume that an adversary can compromise a node. An attacker can launch multiple attacks on the cluster and also may change his position to target other regions of the cluster.

### C. Di-Sec Overview

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. Using this operating system feature, 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.

To create a comprehensive security solution, we analyzed 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. The latter one is 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. Therefore, we created a communication module that controls everything that is transmitted and received through the radio transceiver. Accordingly, the communication module is the main data source component that feed the Di-Sec framework. Moreover, at the heart of Di-Sec we store and analyze all the collected data to provide useful information for security. Our framework is flexible enough to be integrated with existing security solutions and to be used to create new detection and defense mechanisms using the provided services. The Di-Sec framework is completely invisible to the upper layers since it does all the data collection, processing and security execution independent of the upper layers.

## IV. Di-Sec Framework

In this section, we discuss the architecture of Di-Sec in detail. It consists of four main components that have a unique and important role in the framework: the Monitoring-core (*M-Core*), Communication Module (*COMM*), Sensing Module (*Sense*), and Detection and Defense Modules (*DDMs*). The complete framework was implemented in TinyOS-2.x and tested using Tmote Sky sensors. The general Di-Sec architecture is shown in Figure 1. Along with the framework, we implemented four default DDMs using Di-Sec to defend against Jamming, Sybil, Selective Forwarding, and Internal attacks.

### A. The Monitoring Core (M-Core)

The M-Core is the heart of the Di-Sec framework. It provides a novel way to aggregate and distribute information used for the defense against both internal and external threats. All the data and packets going through the Sense and COMM modules are also passed to the M-Core for collection and analysis. In order to reduce the complexity of the implementation and increase the flexibility and modularity of our framework, the M-Core was divided into sub-components, each of which provide some specific services to the DDMs. Table I summarizes some of the services provided by the M-Core. 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.

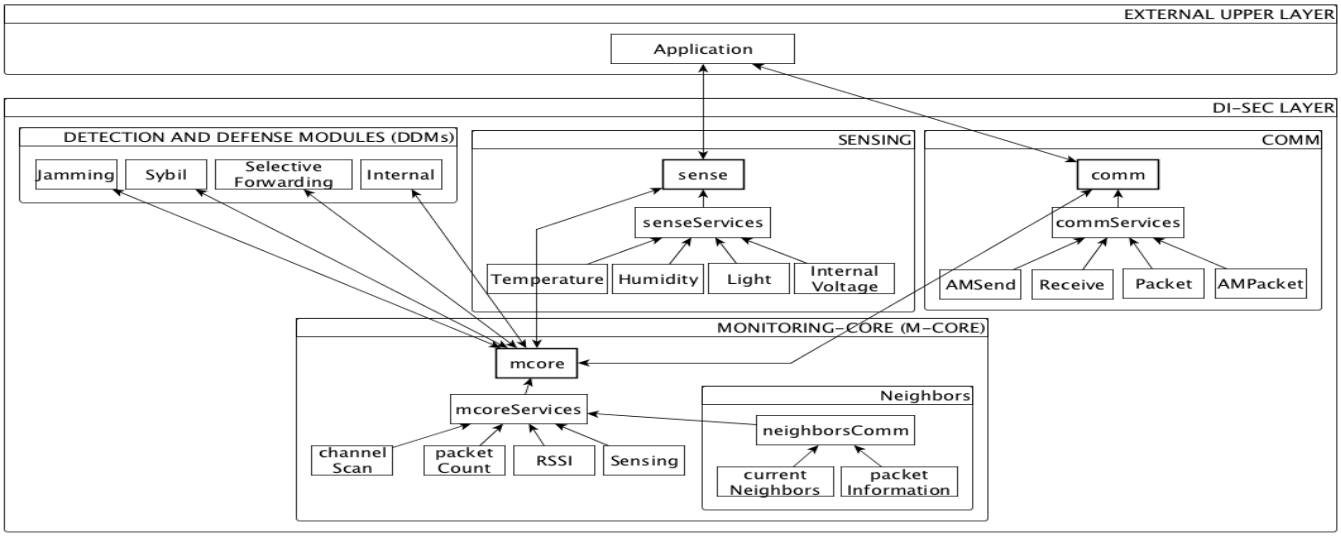



Fig. 1: Di-Sec Architecture.

TABLE I: M-Core Sub-Components

| Component | Interface | Commands/Events | Action |
|---|---|---|---|
| **channelScan** | channelinfo | getConsecutiveSuccess | Returns the number of consecutive successfully sent packets |
| | | getpps | Returns the number of received packets per second |
| | | setThreshold | Sets threshold for acceptable consecutive sent packets rate |
| **packetCount** | packetcount | getPacketCount | Returns the total number of received packets |
| | | lostPacket | Returns the number of packets lost by each node |
| **RSSI** | rssivalue | getRssiTable | Returns neighbors RSSI table |
| | | initRssiTable | Initializes the neighbors RSSI table |
| **Sensing** | sensingstat | getAvgSenseValue | Returns the average sensed value aggregated at the M-Core |
| **neighborsComm** | neighbors | request | Triggers a neighbor discovery message |
| **currentNeighbors** | neighborsinfo | getNeighbors | Returns the number of current neighbors |
| | | initNeighbors | Initializes current neighbors table |
| **packetInformation** | packetsinfo | getTable | Return the packets information table |
| | | initTable | Initializes packet information table |

### *B. The Communication Module (COMM)*

The communication module provides the main communication interfaces: AMSend, Receive, Packet, and AMPacket. 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. 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. 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. The purpose of Di-Sec headers is to facilitate control of the communications and also for the multiplexing of the messages. The defense, detection and other modules inside the M-Core can also communicate securely with the same M-Core modules in other sensors through the COMM module. All the packets sent by the COMM module are encrypted using the embedded AES-128 encryption provided by the CC2420 radio transceiver.

To increase the simplicity 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 1. Note that this setup is automatically generated by the M-Core Control Language introduced in section V.

Listing 1: How to Enable Di-Sec.

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

### *C. 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 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 DDMs security mechanisms.

### D. The Detection and Defense Modules (DDMs)

The final components of the Di-Sec framework are the detection and defense modules. DDMs are specific attack detections and/or defenses 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 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 DDMs 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 a secure ad hoc on demand distance vector (AOVD) routing algorithm 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.

Moreover, 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 default DDMs.

Note that the details for the behavior and implementation of each individual attack are discussed in the performance evaluation section along with the results.

## V. M-Core Control Language (MCL)

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 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.

### A. Rationale for the 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 DDMs or to 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). The MCL is a language that consists of a small set of keywords. The formal definition of the grammar of MCL using the Extended Backus-Naur Form (EBNF) is given in Listing 2. Also, the list of all the keywords in the MCL and their descriptions are tabulated in Table II. 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 *NEW* to activate, deactivate or create the modules respectively. A programmer can even define his/her own variables using the *SET* keyword.

TABLE II: 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* |
| NEW(*module name*, *interface name* ...) | Creates a new detection and defense module |

Listing 2: Formal definition of MCL with EBNF.

```
MCL ::= 'START', SPACES,
        { KEYWORDS, '(', EXPRESSIONS, ')', SPACES },
        'END' ;

KEYWORDS ::= 'ACTIVATE' | 'STOP' | 'SET' |
             'ASSOCIATE' | 'DISSOCIATE' | 'NEW' ;

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

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

VALUE ::= \d* ;

SPACES ::= ' '* ;
```

### B. Sample Usage

In this sub-section, we show a sample usage of the MCL. In our realistic scenario, the user implements a secure WSN program using the MCL to protect against several attacks. The

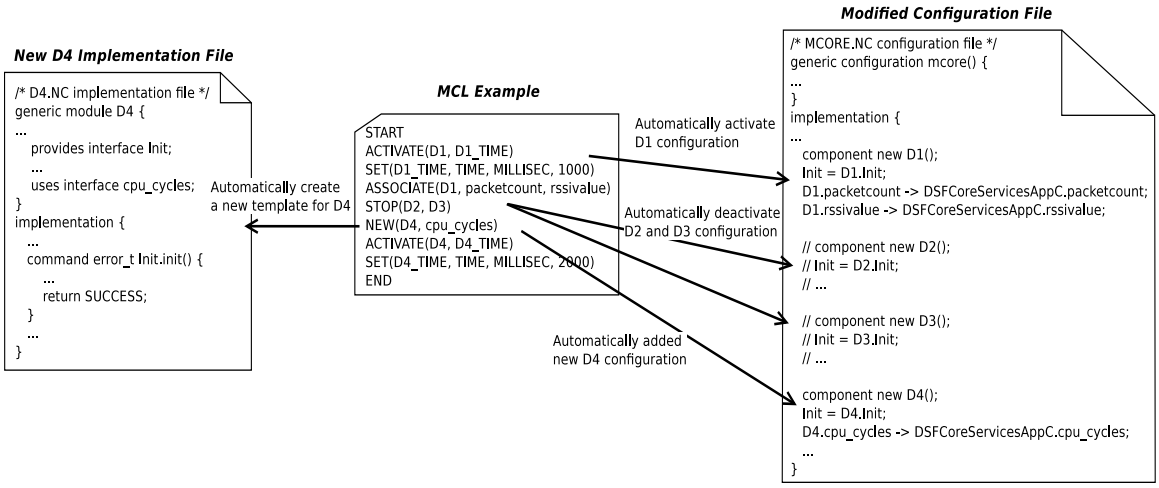



Fig. 2: A realistic example usage of MCL.

MCL written by the user is given in Figure 2 (code snippet in the middle). Specifically, the user instructs 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 D1 and specifies D1's starting time. *ASSOCIATE* is used to connect the D1 to the sub-modules of M-Core in Di-Sec. Also, *STOP* simply disables D2 and D3 which will not be used at run time and disconnects them from the sub-modules. Moreover, *NEW* adds the new D4 module configurations into Di-Sec and generates a new template file for the D4 module implementation. With this one keyword (*NEW*), the users can start writing their own DDMs 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 DDMs and create new ones with simple keywords. Most importantly, the conversion from the MCL to the necessary underlying components (i.e., side files in Figure 2) of the Di-Sec framework and the integration with Di-Sec are automatically handled by the MCL.

## VI. Performance Evaluation

In this section, we evaluate the performance of the Di-Sec framework on real sensors in two dimensions: (1) We evaluate the different components' storage costs (RAM and ROM), the CPU overhead, and communication overhead on individual sensors; (2) We analyze and present the results of the Di-Sec framework in realistic attack-defense scenarios demonstrating Jamming, Sybil, Selective Forwarding, and Internal attacks. We verify that the Di-Sec framework can successfully defend against these and other attacks.

### A. Individual Sensor Evaluation

In our evaluation, we present 9 configurations with different components and analyze the cost of each of them. We have 2 configurations for the upper layer and 7 Di-Sec configurations. In this experiment, the upper layer is an application layer provided with the default installation of TinyOS: *RadioCountToLeds*.

The *2* application layer configurations are *Plain* and *Plain with Security*. The first one represents the plain application layer without any additions as provided with TinyOS. For the second one, we enabled encrypted communications provided by the CC2420 chip through the *SecAMSenderC* component. On the other hand, the *7* Di-Sec configurations include: *M-Core-Plain* which is the basic M-Core configuration with no other enabled components. *M-Core(Security)* has encryption enabled. *M-Core(Security+Sensing)* adds the sensing component to the previous configuration. *M-Core(Security+Sensing+DDM1)* is the previous configuration plus jamming defense and detection module. In the same way, DDM2, DDM3, and DDM4 represent detection and defense modules against Sybil, Selective Forwarding, and Internal threat respectively.

The costs of the different components in terms of storage are presented in Table III. For ROM, we observe that the sensing component and the security (encryption) component have the largest storage costs with 7930 bytes and 3696 bytes, respectively. When considering RAM utilization, M-Core-Plain has the greatest cost (598 bytes) which is expected because the plain M-Core includes all the submodules previously discussed.

In Table IV the CPU overhead when sending and receiving packets, as well as collecting sensing values from the physical sensor are shown. We compare CPU ticks of the plain application configuration and the full M-Core configuration. The results show that the M-Core adds a reasonable amount of 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 scenario, there is minimal overhead since the request has to pass through Di-Sec's Sense component.

TABLE III: Di-Sec ROM and RAM Footprint (Bytes)

| TYPE | ROM | ΔROM | ROM-FREE | RAM | ΔRAM | RAM-FREE |
|---|---|---|---|---|---|---|
| Plain | 18172 | 18172 | 30980 | 1641 | 1664 | 8599 |
| Plain w/Security | 20838 | 2666 | 28314 | 1743 | 98 | 8497 |
| M-Core-Plain | 19328 | 1156 | 29824 | 2236 | 598 | 8004 |
| M-Core(Security) | 23024 | 3696 | 26128 | 2346 | 106 | 7894 |
| M-Core(Security+Sensing) | 30954 | 7930 | 18198 | 2507 | 166 | 7733 |
| M-Core(Security+Sensing+DDM1) | 32058 | 1104 | 17094 | 2690 | 186 | 7550 |
| M-Core(Security+Sensing+DDM1+DDM2) | 32352 | 294 | 16800 | 2715 | 24 | 7525 |
| M-Core(Security+Sensing+DDM1+DDM2+DDM3) | 32710 | 358 | 16442 | 2807 | 98 | 7433 |
| M-Core(Security+Sensing+DDM1+DDM2+DDM3+DDM4) | 32940 | 230 | 16212 | 2823 | 16 | 7417 |

| Time (us) | Length | Frame control field | Sequence number | Dest. PAN | Dest. Address | Source Address | MAC payload | LQI | FCS |
|---|---|---|---|---|---|---|---|---|---|
| +0 | | Type Sec Pnd Ack req Intra PAN | | | | | 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 |
|---|---|---|---|---|---|---|---|---|---|
| +17189672 | | Type Sec Pnd Ack req Intra PAN | | | | | 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 |
|---|---|---|---|---|---|---|---|---|
| +12595802 | | Type Sec Pnd Ack req Intra PAN | | | | | 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 |

Fig. 3: Same Information Passed to Application Layer when Using Different Configurations.

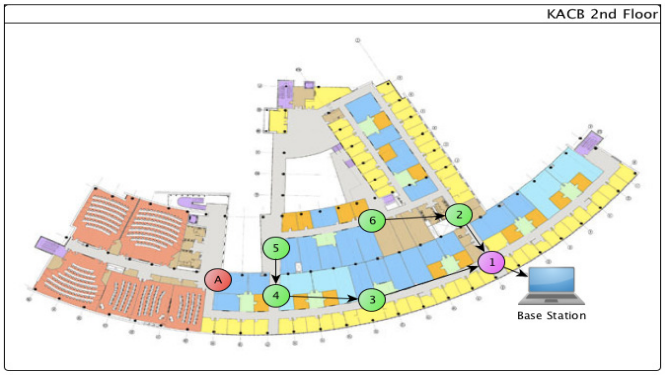



Fig. 4: Experiment Setup.

TABLE IV: Di-Sec CPU Ticks

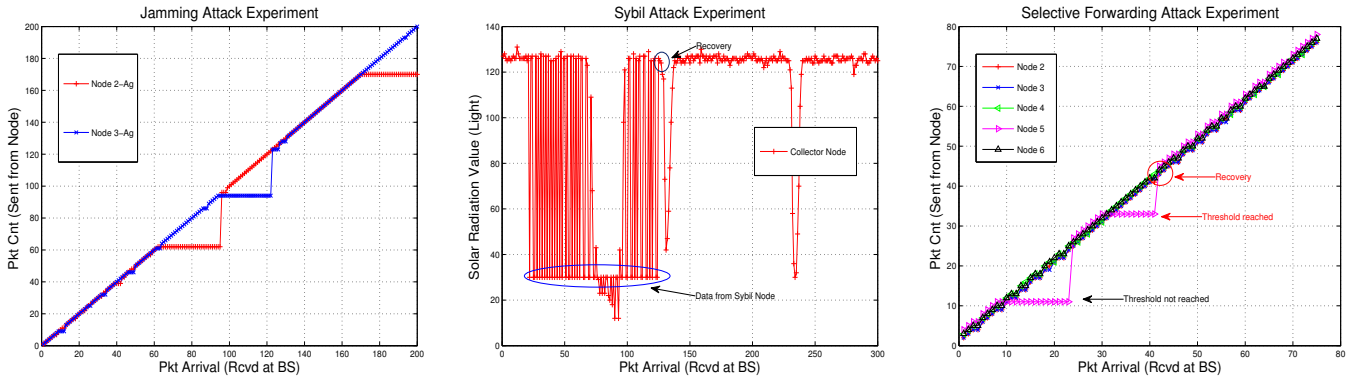
| | M-Core | Plain | Diff |
|---|---|---|---|
| **TX** | 352 | 239 | 113 |
| **RX** | 1600 | 1600 | 0 |
| **Sensing** | 550 | 546 | 4 |

The Di-Sec header is 6-bytes long and includes a 2-byte source node ID and 1-byte Di-Sec sequence number that are 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 shows the difference between the packets sent with the plain configuration, 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.

### B. Experiments Evaluation

To test the Di-Sec framework, we created a 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 4. 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 attack scenarios will be explained along with the results.

*1) Jamming Scenario:* In this experiment, the complete jamming DDM was implemented inside the Di-Sec framework. As shown in Figure 4, an attacker was placed next to nodes 4 and 5 to jam the communication channel. Figure 5(a) shows the aggregated packet count and arrivals at the base station. We see that node 3 has a higher traffic intensity than node 2 which is expected since node 3 is forwarding packets coming from nodes 4 and 5 and node 2 is only forwarding packets generated at node 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




(a) Jamming Attack Results. (b) Sybil Attack Results. (c) Selective Forwarding Attack Results.

Fig. 5: Attack Experiments.

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 defends against Jamming attacks and the packet loss due to the attack was approximately 15% for this specific scenario.

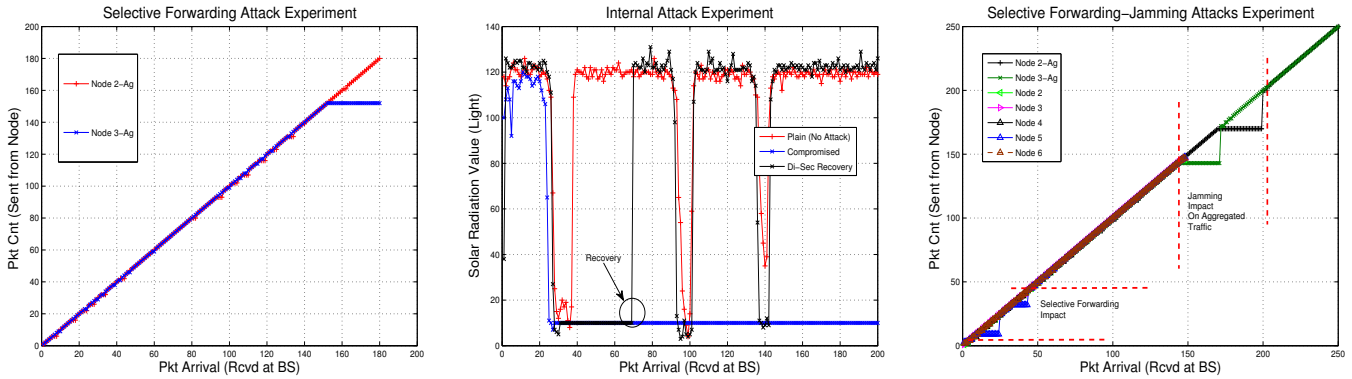
*2) Sybil Scenario:* For the sybil detection, we used a RSSI table containing average RSSI values for each neighbor. This table is updated every time a packet arrives at the sensor since the packet is passed to the M-Core and the RSSI value is extracted and averaged. For this scenario, we collect at least 10 sample packets from each neighbor to calculate the RSSI average and define an upper and lower threshold for the RSSI. For this experiment, we used two 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 one sybil sensor that impersonates the sampler and injects false data into the network which is received by the collector node. Figure 5(b) 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 attack scenario as well.

*3) Selective Forwarding Scenario:* The Selective Forwarding attack scenario also uses the the topology shown in Figure 4. For this scenario, we deliberately modified node 4 to drop 66% of the received packets. The M-Core provides the ability to set thresholds to trigger some actions. In this scenario, we set the threshold for the maximum acceptable packets dropped by a relaying node to 25 packets. Figure 5(c) shows that packets from node 5 were being dropped. The first time the neighbors detect this irregularity the threshold had not been reached, therefore, no action is taken. The second time the misbehavior is detected, node 5 changes its relaying node to be node 6 and reaches the base station through node 2. 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 6(a). If we compare Figures 6(a) and 5(a) and ignoring the jamming fluctuations on 5(a), we observe that node 2 transmitted more aggregated traffic in the selective forwarding scenario and node 3 during the jamming experiment. 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.

*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 an 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 from our 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 the Di-Sec sensing component to verify the collected values. Figure 6(b) presents results of this experiment and shows that the compromised application relying on our framework services identifies and recovers from the malicious attack.

*5) Combined Attacks:* For the sake of completeness, we combined and launched two of the previous attacks in a single experiment. Specifically, we combined the Jamming and Selective Forwarding attacks to demonstrate that the Di-Sec framework successfully defends and recovers from any combination of attacks. Figure 6(c) shows the overall traffic behavior per node and aggregated at the gateways (nodes 2 and 3). Again, node 5's traffic is dropped by the selective forwarder (node 4) and after detection the route is changed and 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 which was able to finally resume the normal communications.




(a) Selective Forwarding Aggregated Traffic. (b) Internal Attack Results. (c) Selective Forwarding and Jamming Attacks.

Fig. 6: Attack Experiments 2.

## VII. Conclusions and Future Work

In this work, we introduced a comprehensive security framework for WSNs called Di-Sec. The goal of our architectural design was to create a 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 the M-Core Control Language (MCL) to interact with the framework. 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). We study the performance of the framework in terms of storage costs (RAM and ROM), CPU overhead, and communication overhead. We also implemented DDMs against Jamming, Sybil, Selective Forwarding, and Internal attacks and show through experimentation that Di-Sec framework successfully defends and recovers from those attacks.

For our future work, we will increase the number of M-Core services and the attacks' detection and defense coverage. We will also incorporate to the framework the capability to dynamically distribute code modules at run time, to allow the exchange of defense and detection modules. We also envision our framework to work on collaborative cluster environments as proposed in our previous work [5]

## VIII. Acknowledgements

This work is partially supported by NSF Grant# CNS-1052769

## 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] 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, p. 6 pp. 1048.

[7] GT-CAP, "Di-Sec: Distributed Security Framework for Heterogeneous Wireless Sensor Networks," http://www.di-sec.org/.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.