
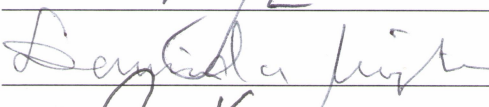
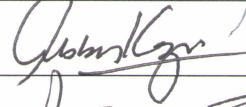

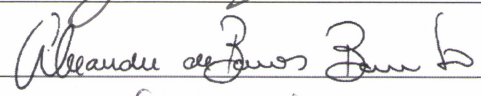
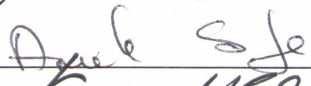
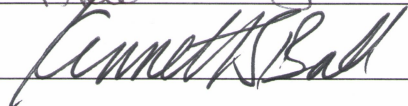


COLLABORATIVE MOBILE AD HOC INTRUSION DETECTION SYSTEM

by

Jeronymo M. A. de Carvalho
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Systems Engineering and Operations Research

Committee:

| | |
|---|---|
|  | Dr. Paulo C. G. da Costa, Dissertation Director |
|  | Dr. Duminda Wijesekera, Committee Member |
|  | Dr. Syed Abbas K. Zaidi, Committee Member |
|  | Dr. Jie Xu, Committee Member |
|  | Dr. Alexandre B. Barreto, Committee Member |
|  | Dr. Ariela Sofer, Department Chair |
|  | Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering |

Date: FEBRUARY 24, 2017

Spring Semester 2017
George Mason University
Fairfax, VA

Collaborative Mobile Ad Hoc Intrusion Detection System

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

By

Jeronymo M. A. de Carvalho
Master of Science
Military Institute of Engineering, 2009
Bachelor of Science
Military Institute of Engineering, 2003

Director: Dr. Paulo C. G. da Costa, Associate Professor
Department of Systems Engineering and Operations Research

Spring Semester 2017
George Mason University
Fairfax, VA

Copyright © 2017 by Jeronymo M. A. de Carvalho
All Rights Reserved

Dedication

I dedicate this dissertation to lovely wife and son.

Acknowledgments

I would like to thank the members of my Committee, who assisted me along my research. I also would like to thank my wife, who understood my absence and continued to support me at all times.

Table of Contents

| | Page |
|--|------|
| List of Tables | vii |
| List of Figures | viii |
| Abstract | x |
| 1 Introduction | 1 |
| 1.1 Intrusion Detection System on Mobile Networks | 5 |
| 1.2 Problem Statement | 10 |
| 1.3 Methodology | 11 |
| 2 Background and Related Work | 13 |
| 2.1 Wireless Networks | 13 |
| 2.1.1 MANET | 14 |
| 2.1.2 WSN | 15 |
| 2.2 Intrusion Detection Systems | 17 |
| 2.3 Position Information Estimation | 22 |
| 2.4 Key Management Protocol | 27 |
| 2.5 Automated Proof of Security Protocols | 28 |
| 3 CMIDS Framework | 33 |
| 3.1 CMIDS Premises | 33 |
| 3.2 Design Characteristics of CMIDS | 34 |
| 3.3 Architecture | 44 |
| 4 Test and Evaluation | 56 |
| 5 Discussion | 69 |
| 5.1 Military Radio Equipment Configuration | 69 |
| 5.2 Civilian Radio Equipment Configuration | 73 |
| 5.3 Overall Performance | 77 |
| 6 Conclusions | 79 |
| A Key Establishment Protocol (Scyther Source Code) | 82 |
| B MÄK VR-Forces Background Script | 85 |
| C VR-Forces/CORE Interface Source Code | 91 |
| D CMIDS Source Code | 103 |

| | | |
|---|---|-----|
| E | Background and Position Announcement Traffic Generators | 128 |
| F | CMIDS Multilateration Error Estimation Source Code | 148 |
| | References | 160 |

List of Tables

| Table | Page |
|--|------|
| 4.1 Scenario configurations. | 62 |
| 4.2 Evaluation Metrics Description | 67 |
| 4.3 Specifications of the Computers Used at the Simulation Setup | 68 |
| 5.1 Summarized Performance Metric (Military Radio Configuration) | 70 |
| 5.2 False Discovery Rate (Military Radio Configuration) | 71 |
| 5.3 Time to Detect an Attack (Military Radio Configuration) | 72 |
| 5.4 Summarized Performance Metric (COTS Radio Configuration) | 74 |
| 5.5 False Discovery Rate (COTS Radio Configuration) | 76 |
| 5.6 Time to Detect an Attack (COTS Radio Configuration) | 76 |

List of Figures

| Figure | Page |
|--|------|
| 1.1 Examples of MANETs | 2 |
| 1.2 Classification of network layer attacks in MANETs | 3 |
| 1.3 OODA Loop | 4 |
| 1.4 Basic Intrusion Detection System Architecture | 6 |
| 1.5 A hypothetical intrusion scenario | 8 |
| 2.1 Compiled WSN Taxonomy | 17 |
| 2.2 Anomaly based IDS techniques | 18 |
| 2.3 Misuse based IDS techniques | 19 |
| 2.4 Avoided Attacks using Misuse based IDS techniques | 20 |
| 2.5 Proposed IDS Taxonomy | 21 |
| 2.6 Position Information Acquisition Techniques | 24 |
| 2.7 Example of Position Estimation using Multilateration | 26 |
| 2.8 Some aspects of Security Protocols Analysis | 29 |
| 2.9 Automated Security Protocols Proof Tools | 31 |
| 3.1 Capture the Flag Scenario | 36 |
| 3.2 Highlighted Intrusion Detection Taxonomy of CMIDS | 38 |
| 3.3 CMIDS Network Topology | 40 |
| 3.4 Simulated CMIDS Position Estimation Error | 42 |
| 3.5 CMIDS Multilateration Mean Error | 43 |
| 3.6 CMIDS Architecture | 45 |
| 3.7 CMIDS Architecture at each Sensor Node | 46 |
| 3.8 CMIDS Sensor Algorithm | 47 |
| 3.9 Simplified Detection Activity Diagram | 49 |
| 3.10 Key Establishment Activity Diagram | 50 |
| 3.11 Key Establishment Protocol | 52 |
| 3.12 Protocol Security Verification at Scyther | 54 |
| 4.1 Testbed Developed to Evaluate CMIDS | 57 |
| 4.2 Simulated Scenario implemented at VR Forces | 59 |

| | | |
|-----|--|----|
| 4.3 | Position Location Message GUI | 61 |
| 4.4 | Eavesdropping Attack Activity Diagram | 64 |
| 4.5 | Example of traffic captured at sensor node | 65 |
| 5.1 | Attack Detection Timeline | 78 |

Abstract

COLLABORATIVE MOBILE AD HOC INTRUSION DETECTION SYSTEM

Jeronymo M. A. de Carvalho, PhD

George Mason University, 2017

Dissertation Director: Dr. Paulo C. G. da Costa

A Mobile *Ad Hoc* Network (MANET) is a type of network that does not require previously deployed infrastructure to operate. In such networks, every node is mobile and acts as a router and, as a consequence, MANETs can be quickly configured and ready to work. This capability suits well to scenarios in which networks only need to operate for a short period of time, and when initial investment on infrastructure is undesired or not possible. It also allowed the implementation of many applications that were not originally practical with regular networks. For instance, law enforcement missions performed by police forces into areas where the State is not well present are an example. These forces are inherently mobile and need to exchange data in a uncontrolled area in order to accomplish their mission. Another example is the use of MANETs supporting natural disaster operations. In this case, the preexisting communication infrastructure is damaged and field agents resort to MANETs as the medium for coordinating rescue missions and distribution of supplies. The military also use MANETs as a platform to perform their duties, and as a key asset supporting a number of diverse command and control activities. For instance, platoon and company level missions are very mobile and performed in areas under control of the enemy. MANETs play a major role in other domains as well, including health care and remote sensing, making mobile *ad hoc* networking a growing trend.

In most of the above application domains, a typical MANET implementation requires quick deployment, dynamic communications, and security assurance. However, these requirements are made difficult to attain due to MANETs' wireless nature and its lack of a fixed structure, which makes the information flow on an unconfined environment and undesired nodes able to directly interact with the network.

To prevent unauthorized access, substantial effort has been put into the design of strong encryption algorithms, secure protocols, and intrusion detection methods. Unfortunately, networks are still vulnerable to diverse factors such as rogue users, poor software design, and information leakage from social engineering, and even the most reliable network security technologies available today cannot make them invulnerable. Once a network is compromised, the enemy is able not only to eavesdrop sensitive information but also to mislead valid users and to harm the operations supported by the network.

This research addresses the security issue of MANETs from a new point of view. Instead of only trying to deny the network access to the enemy, it seeks to identify the adversary that already has the credentials to access the network before he can perform rogue actions. More formally, the hypothesis tested in this work is that the use of the location information of the tactical nodes enables the ability of detecting a passive intruder before it becomes active, and diminishes the impact of the adversary on the mission supported by the mobile network. The research focused on obtaining this ability through the use of a combination of different techniques, and resulted in the design of the Collaborative MANET Intrusion Detection System (CMIDS).

CMIDS is a novel, non-intrusive IDS that uses a secondary network of sensors to monitor and analyze the behavior of the nodes of the target network. A predictive location algorithm was created and, in combination with multilateration technique and basic mission knowledge, is able to detect misbehaving nodes.

In order to test the hypothesis and evaluate the CMIDS concept, a military tactical scenario simulation was developed using three distinct software tools. The results obtained through the experiments corroborate the Dissertation hypothesis and show that the CMIDS extends the state of the art of intrusion detection systems for mobile *ad hoc* networks.

Chapter 1: Introduction

The last century was marked by the emergence and proliferation of computers and data networks. They evolved over time and acquired mobile capability. A vast number of everyday activities in our society rely on not only structured mobile networks but also Mobile *Ad Hoc* Networks (MANETs). For instance, law enforcement mission performed by police forces into areas where the State is not well present is one example. The police forces are mobile and need to exchange data in a uncontrolled area in order to accomplish their mission. Another example is the use of MANETs supporting natural disaster operations. In this case, the communication infrastructure is damaged and the field agents use MANETs to coordinate the rescue missions and the distribution of supplies. The military also use MANETs as a platform to perform their duties. Platoon and company level missions are very mobile and they are performed in areas under control of the enemy. The military use mobile networks to act and to support the command and control activities.

A Mobile Ad Hoc Network (MANET) is a set of mobile devices which can form a temporary network with the absence of previous infrastructure [1]. Each device acts as a network router and communicates with the others using its wireless interface. This implies that MANETs must be resilient and have self-configuration and self-maintenance capabilities. Since the devices are mobile, the network topology may change quickly and unpredictably over time. Figure 1.1 illustrates some MANETs.

Due to the nature of its design, MANETs are very susceptible to attacks. The lack of a fixed structure combined to the wireless transmissions let the information flow on an unconfined environment and attackers are able to directly interact with the network. Roughly speaking, these attacks fall into one of two groups: passive or active attacks. Figure 1.2 presents the most common network layer attacks performed against MANETs

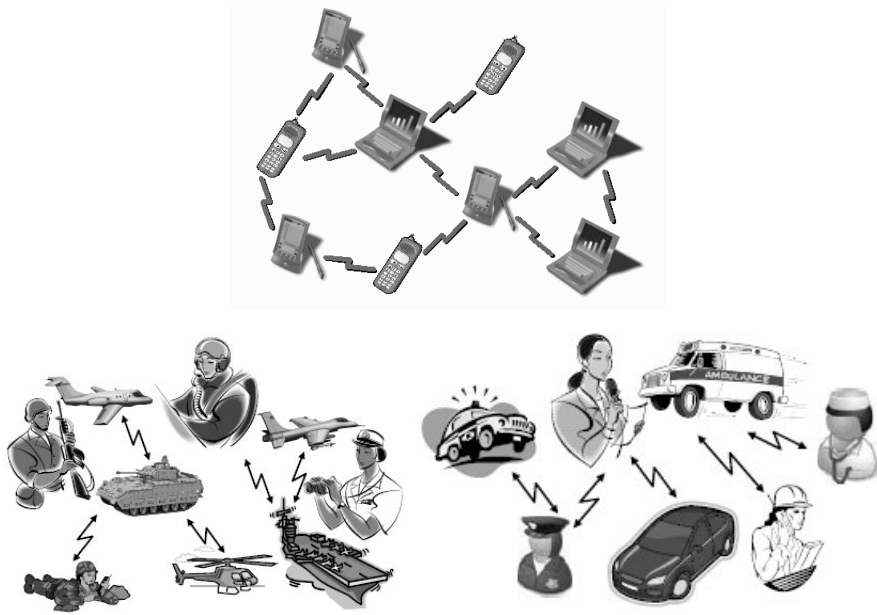


Figure 1.1: Examples of MANETs.

[2]. But other layers of OSI model are also susceptible to attacks, such as the physical layer (jamming attacks) and the application layer (data duplicity and repudiation attacks).

Passive attacks are related to the "Observe" and "Orient" steps of the OODA Loop as depicted at Figure 1.3. These attacks are those where the adversary does not disturb the network but just observes and learns from the collected data [3]. It can lead to valuable information as network topology, critical nodes, and identity discovery. These attacks usually precede the active attacks.

Active attacks are related to the "Act" step of the OODA Loop. These attacks are those where the adversary plays intrusive activities such as injecting, forging or dropping data to mislead the network [3]. Although they have the potential to be very harmful, they are not as hard to be detected as the passive attacks because they modify the regular traffic of the network while the passive attacks don't. A mechanism looking for network anomalies is able to detect an active attack but not a passive one. But even some active attacks are harder to be detected than others. A jamming attack consist of extra information injected in the

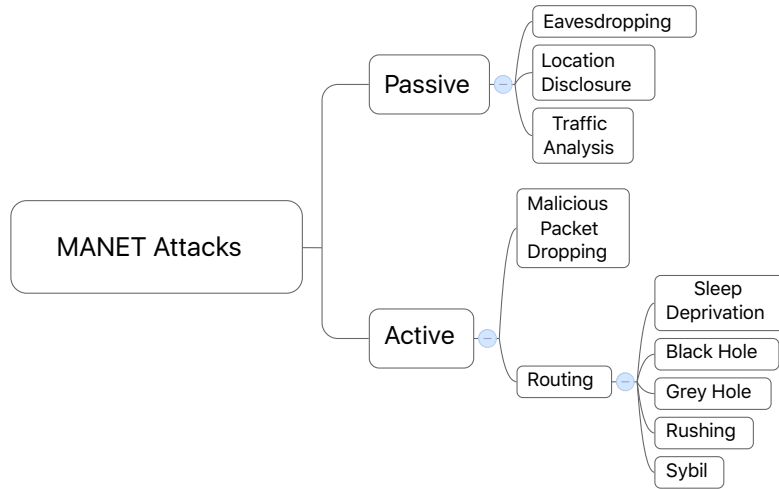


Figure 1.2: Classification of network layer attacks in MANETs.

network with the purpose of deny some kind of service. This means the target is unable to operate and the attack is easily noticed. But a deception attack injects only enough extra information to mislead the target and prevent him to operate at it should. It is hard to say if the target is under attack or is just experiencing a legitimate overhead.

A complete attack comprehends five phases [4]:

- Reconnaissance
- Scanning
- Gaining Access
- Maintaining Access
- Covering Tracks

The adversary usually desires to complete all the five phases but it is not always true on attacks performed on MANETs. Structured networks are used for a long period of time and it justify the attacker's effort to maintain the access and cover its tracks to be able to

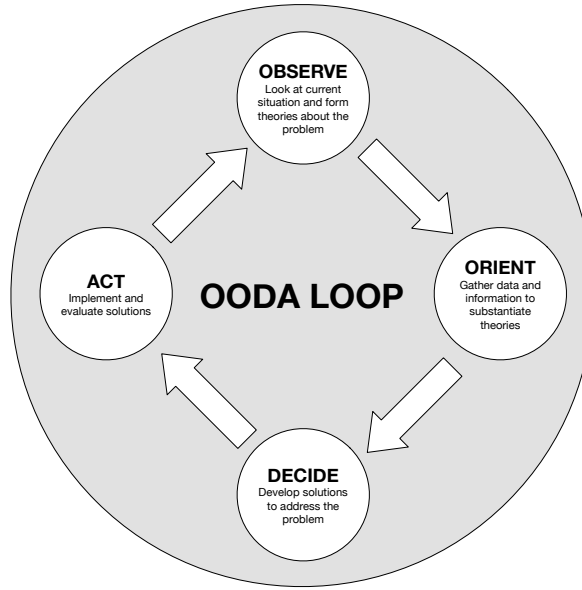


Figure 1.3: OODA Loop.

return in the future. But a MANET is used for a short period of time and the attackers usually skip the last two phases because eavesdropping sensitive information, misleading users and harming the network are the goals and not returning in the future.

Prevention methods such as strong cryptography techniques can significantly reduce the number of successful attacks in wireless networks, but are not sufficient to protect MANETs, that are frequently used to support critical operations. Successful attacks to MANETs may imply serious consequences as government classified information leakage, unauthorized access to lethal power platforms, and lost of human lives. One of the strategies available to enhance the security on networks is to use layers of protection. Each layer is responsible to protect the network against a set of types of attacks. The adversary may still be able to overcome the layers but more time is spent and higher are the chances to react against the intruder. This approach is consistent with the defense-in-depth approach adopted at the Security Requirements Guide (SRG) of the Department of Defense (DoD) [5]. One of the most effective layers to secure MANETs is the use of Intrusion Detect Systems (IDS) [6].

1.1 Intrusion Detection System on Mobile Networks

An Intrusion Detection System (IDS) is a set of the tools, methods, and resources used to identify, assess, and report intrusions [7]. In the data network context, it may be defined as software and hardware used to monitor the network and detect internal or external cyber attacks [8].

An IDS is composed of four main components:

- Sensors
- Detector
- Knowledge Base
- Response Component

Sensor components are responsible for collecting data from the monitored system. The detector component analyses the collected data to detect intrusions and uses the knowledge base component to support this task. The response component manages the response actions to the attacks [8].

A basic IDS architecture is illustrated at Figure 1.4

There are different techniques used to implement an IDS, which are categorized into three groups: anomaly-based detection, misuse-based detection, and specification-based detection. The first leverages statistical, knowledge and machine learning techniques to identify anomalous operations and transmissions at the network. The second group uses signatures and rules techniques to build the profile of known attacks and avoid them. Taking a different approach, the third group uses specification and constraints to describe the correct operation of the network and identify any different behavior.

Although all groups employ effective techniques to provide detection at a low false positive rate, most of them were designed for wired networks (closed environment) and are not convenient to wireless networks (open environment). The intrusion detection challenge becomes even harder when considering mobile *ad hoc* networks, as it introduces new issues,

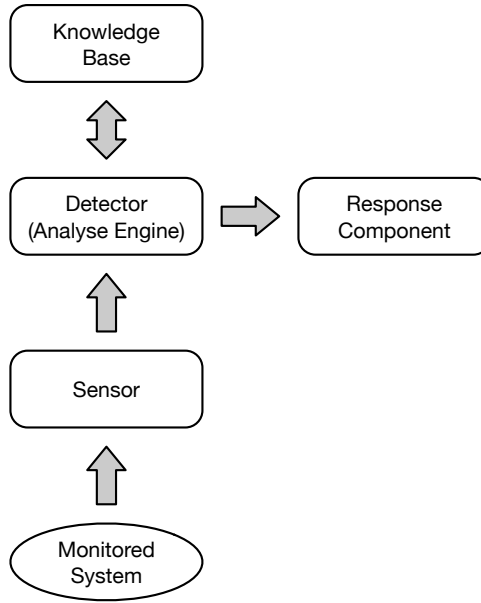


Figure 1.4: Basic Intrusion Detection System Architecture.

such as lack of concentration points to monitor and audit data, nodes acting as routers, dynamic and unpredictable topology, and limited computational ability [2].

Security issues caused by the absence of a static infrastructure are also stated in [9]. Cyber attacks usually try to compromise the confidentiality, integrity and availability of messages, but in the case of mobile wireless network an extra aspect is included to the triad: energy. Mobile hosts usually have a limited power resource, which can be exploited by provoking an abnormal use aimed at reducing the battery life so to prevent the mission from being accomplished.

Due to its intrinsic characteristics, secure mobile *ad hoc* networks are very hard to implement [6], and no system is completely secure [10]. Considering that the number of sophisticated and tailored attacks on computer networks has significantly increased [11], the defense-in-depth approach is threatened and enhancements to the IDS layer are desired.

There are some well-known intrusion detection techniques specifically designed for securing MANETs. The Watchdog scheme [12] works promiscuously listening to the next hop's transmissions. Every time a node fails to forward a data package within a pre-established amount of time, the Watchdog IDS increases its believe that the neighbor is actually an adversary and an attack has been detected. Although very popular, this scheme fails to detect adversaries in the presence of collisions and limited transmission power which is the environment of application of a MANET. The EAACK scheme [13] tries to overcome these issues through the modification of the routing protocol. Each destination node of a message is forced to send back an acknowledge data package informing the sender node when the message was received. This scheme lowers the false positive detection but it requires the modification of the routing protocol, increases the network traffic, and it is still vulnerable to collisions issues. It means that EAACK is not appropriate to legacy MANETs that can not be modified due to issues with power consumption or recertification of its protocols. It also means that EAACK increases the number of collision because it requires more messages. CRADS [14] adopts a different approach and uses Support Vector Machine (SVM) to learn the regular behavior of the network and then detect anomalies. Besides been intrusive as EAACK, CRADS requires extra time spent on the learning phase and does not fit well scenarios that require fast deployment and operation because it is not available all the time. IDSX [15] is another IDS scheme designed to work with MANETs. It assumes the mobile network is divided into clusters which enable the nodes to communicate with each other. Every node of a cluster collects information and share it with the cluster head but only the last one is responsible to decide if a suspect node is an attacker. The issue with this approach is its vulnerability to cluster head failures because no recovery mechanism to elect a new cluster head is defined. The unavailability of a cluster head can let an entire cluster without protection of the intrusion detection system.

The implementation of an IDS to work with MANETs is a complex task [16]. The majority of approaches addresses the detection of active attacks against MANETs and do not take into consideration passive attacks. These approaches are ineffective because they

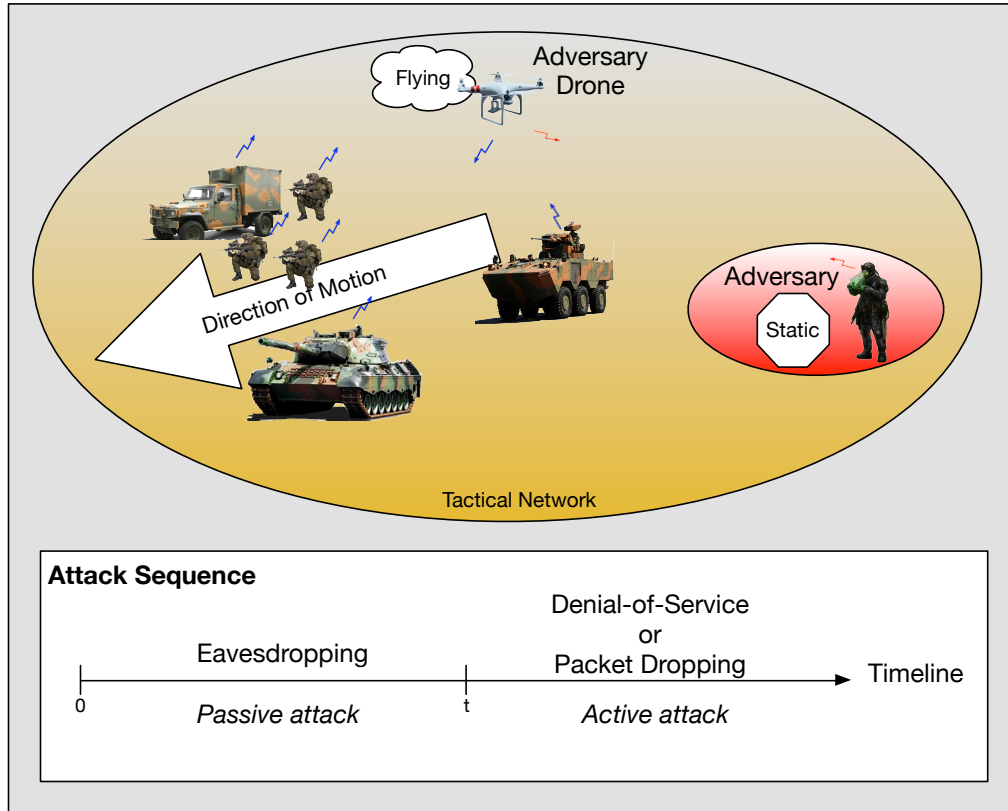


Figure 1.5: A hypothetical intrusion scenario.

are only capable of act after a certain level of harm is imposed to the mobile network.

In contrast, an IDS capable of detecting passive intruders has the potential to be really effective because it can identify the adversary before any harm is done to the mobile network. Such type of IDS, designed to operate on MANETs, could not be found on the related literature and is explored by this research. Instead of trying to deny the network access to the enemy, it seeks to identify the adversary that already has the credentials to access the network as showed at Figure 1.5. In the tactical scenario represented by the illustration, a MANET is used to support military operations and it is under attack of an adversary performing passive attacks. Unfortunately, the adversary has not yet been detected.

Although very powerful, this kind of adversary cannot perfectly mimic the moving pattern of a legitimate node. First, there are the kinetic differences resulting from the use of distinct platforms to launch the passive attack. Second, even if the adversary uses the same platform used by a legitimate node to launch the attack, as in the case of a captured node, it still need to keep some distance from the troops to avoid being seen. This difference on the mobility pattern could be used to identify intrusions to the mobile network. It is important to notice that is out of the scope of this dissertation dealing with nodes infected before the deployment of the MANET. This case would require the adversary to have physical access to the node during the configuration step of the tactical network.

This powerful adversary has a great potential to harm the network because it has already overcome the first layers of the defense-in-depth approach. An IDS capable of detecting passive adversaries would add an extra layer against rogue users, poor software design, adversaries using stolen credentials or stolen equipment, and information leakage from social engineering.

In addition to proposing a novel IDS that can be used as an extra layer to the defense-in-depth approach, this research investigates if an IDS capable of detecting passive adversaries really performs better than an IDS designed to identify active adversaries. The hypothesis here is that the ability to identify a passive intruder before it becomes active diminishes the attack impact of the adversary on the mission supported by the mobile network.

An implementation of the proposed IDS should be built and tested in order to evaluate if it really diminishes the negative impacts caused by the adversary attacks. This test should not be only about the operation of the IDS itself but also about the influence it has on the mission that the mobile network is supporting. The validation of a system usually requires a field test but this is the last test step, especially when it comes to critical scenarios. The alternative is to use a virtual simulated scenario.

Computer simulation is a good practice to study the performance of a new IDS and it is recommended by the DoD at the publications of the Modeling & Simulation Coordination Office [17] [18]. According to [19], computer simulation is advised to be used to model:

- systems with many random variables and interacting components with nonlinear relationships;
- interdependence between resources and system elements; and
- systems that need visual animation of the output.

These characteristics are present in several tactical scenarios that uses MANETs to support their operations. As a consequence, simulation is advised to model them.

Simulation also has some advantages when compared against the real implementation of the system for evaluation purposes. For instance, simulation studies consumes less time to produce results, are more cost effective, provide enhanced control of the system variables, are safer, usually scales better, and can stress the system to scenarios that would not be possible in the real world [20]. Simulation is useful to study systems with complicated behavior that does not allow a closed-form representation and at the same time it reduces research cost.

1.2 Problem Statement

A secure mobile network is a basic requirement to scenarios using wireless communications in support of tactical missions. Although approaches have been developed to secure Mobile Ad Hoc Networks (MANETs), these networks still present vulnerabilities that jeopardize critical missions. To the best of my knowledge, by the time of this writing no Intrusion Detection System (IDS) has been capable of detecting passive attacks performed against MANETs. Examples of such attacks include adversaries using stolen credentials, exploring poor software design, or using credentials obtained through social engineering. After a successful attack, adversaries can access the MANET while keeping a passive profile, which enables them to eavesdrop sensitive information, mislead legitimate users, and harm the tactical mission. In spite of such capabilities, these adversaries cannot perfectly mimic the kinetic pattern of a legitimate node from the mobile network as it physically moves through the terrain. Therefore, a key aspect in detecting passive MANET intruders is the use of the

location position of the tactical nodes in contrast to what is expected given their role in the ongoing mission. The present work investigates this aspect comprehensively, with the goal of designing a novel IDS capable of detecting passive MANET intruders.

In summary, this study addresses the problem of detecting, in a timely fashion, passive adversaries who successfully obtained access to MANETs. It tests the thesis hypothesis that it is possible to leverage specific knowledge (location, mission, capability) about the tactical nodes in order to (i) bring the ability of detecting a passive intruder before it becomes active and (ii) reduce the impact of the adversary on the mission supported by the mobile network.

1.3 Methodology

In order to verify the thesis hypothesis, a test methodology was created comprising the steps described in the follow.

First, a proper way to acquire the geospatial position information of each node of the mobile network must be defined. The position information will be calculated using multi-lateration performed by a Wireless Sensor Network (WSN).

This WSN will also be used as the platform to run the novel IDS that will be designed. Each sensor node will run algorithms to analyze the moving pattern of the tactical nodes. These algorithms will seek passive adversaries acting as legitimate nodes but forging their position, moving in a manner incompatible with the capacities of the legitimate nodes, and moving in disagreement with mission plan.

A test to evaluate this new IDS, so named Collaborative MANET Intrusion Detection System (CMIDS), will be elaborated. Such test must be able to identify if the new IDS detects intruders performing passive attacks. It also must provide metrics to compare this new IDS against other state of the art intrusion detection systems and to access the impact of the IDS on the mission.

A computer simulation testbed will be created to enable the test requirements. It will

be formed by three software packages (MÄK VR-Forces [21], CORE [22], and EMANE [23]) and one software developed during the research to be used as interface between the software packages.

The MÄK VR-Forces platform will be used to design a tactical scenario that uses MANET. This software provides an accurate, physics-grounded simulation environment as well good mobility models for ground troops. CORE and EMANE will be used to simulate the MANET communication. This two software platforms work together to offer a real-time data network simulator.

Chapter 2: Background and Related Work

This chapter provides the required background to understand the proposed Intrusion Detection System (IDS) designed to work with Mobile Ad Hoc Network (MANET). It also introduces the main work related to the problem addressed in this Dissertation. First, Section 2.1 introduces wireless networks focusing on the concepts of MANET and WSN and why they are distinct from regular wireless networks. Then, Section 2.2 provides an overview of IDS and its applicability to MANETs. Section 2.3 covers the subject position estimation, a key aspect for this research, presenting the existing techniques and how they are related to MANETs. Finally, Section 2.4, Key Management Protocol, and Section 2.5, Automated Proof of Security Protocols, cover some additional concepts necessary to evaluate the ideas proposed here.

2.1 Wireless Networks

Wireless Network (WN) is any type of network connected by radio stations [24]. Instead of cables, it uses radio waves to transmit and receive information. In the case of computer wireless network, the information is digital and protocols are responsible to control the access to the environment and to exchange the data.

Unlike wired network that have multiple topologies, wireless networks have only two types of topology. According to the presence of infrastructure, the WN topology may be considered *Infrastructure* or *Ad Hoc*. Infrastructure topology is the one that uses a central wireless device to concentrate and organize the communication between the wireless nodes. Such device is named Wireless Access Point (WAP). In contrast, Ad Hoc topology does not require a WAP. Every node on the network connects directly to the neighbor node. Connections between nodes too far apart are made possible because every node acts as a

router forwarding data.

A special case of ad hoc wireless network is the MANET. In this network, all the nodes are mobile and the disconnections are very frequently. A second special case is WSN. This network is used for the specific purpose of monitoring the environment. The following subsections provide more details about MANET and WSN.

2.1.1 MANET

MANETs have been used for a long time but only in 1997 the Internet Engineering Task Force (IETF) created a working group dedicated to the subject. It was between the IETF meetings numbers 38, in Memphis, and 39, in Munich [25]. Since then, a lot of documents, RFCs, standards and protocols have been published.

IETF defined IEEE 802.11 architecture to be used by MANETs [2]. The distinction for MANETs is made at the definition of the service set of the protocol. Instead of using the Basic Service Set (BSS), MANETs use the Independent Basic Service Set (IBSS). It allows the network to operate without a controlling Access Point (AP) and exchange messages in a distributed peer-to-peer manner.

A MANET can be defined as a set of mobile wireless devices which can form a temporary network with the absence of previous infrastructure [1]. Each node (device) acts as a network router and communicates with the others using its wireless interface.

They are very useful due to its characteristic of fast configuration and easy deployment. Several applications demanding short period of time operations and high mobility level take advantage of MANETs while other applications are only possible due to them. For instance, law enforcement mission performed by police forces into areas where the State is not well present is one example. The police forces are mobile and need to exchange data in a uncontrolled area in order to accomplish their mission. Another example is the use of MANETs supporting natural disaster operations. In this case, the communication infrastructure is damaged and the field agents use MANETs to coordinate the rescue missions and the distribution of supplies. The military also use MANETs as a platform to perform

their duties. Platoon and company level missions are very mobile and they are performed in areas under control of the enemy. The military use mobile networks to act and to support the command and control activities.

But the same design that make MANETs useful to many applications also turns them very susceptible to attacks. The lack of a fixed structure combined to the wireless transmissions let the information flow on an open environment and attackers are able to directly interact with the network.

There are two sets of attacks against MANETs: passive attacks and active attacks. Passive attacks are those where the enemy only listens to the transmissions and learns from the collected data [3]. With this kind of attack the enemy is able to collect valuable data of the network and, for instance, identify a critical node connecting two distinct clusters of the same MANET or even identify the role of some node as the troop leader of a MANET used by military.

Unlike passive attacks, active attacks are those where the enemy interferes with the network operation. It plays intrusive activities such as injecting, forging and dropping data to mislead the network [3]. This kind of attack may be very harmful to the network but is easier to be detected than passive attacks. Because they modify the regular traffic of the network, a mechanism looking for network anomalies is able to detect an active attack but not a passive one.

This research studies how to enhance the security of MANETs against passive attacks and, as a consequence, lower the adversary power of damaging the network and harming the mission supported by the MANET.

2.1.2 WSN

Wireless Sensor Networks (WSNs) have been around for quite some time. Their first recorded usage dates from the World War II, with the purpose of detecting hidden enemies [26]. Sir Winston Churchill, Prime Minister of United Kingdom at that time, used to call the race for electronic superiority the “Wizard War” [27]. The usual reference of one of

the first WSN is the Sound Surveillance System (SOSUS) installed by the US Navy during the cold war to detect soviet submarines [28]. But it was at last years that the importance of WSNs increased. This emerging technology is used not only by military forces but also by several different sectors as industry and healthy care.

A WSN is defined as a network that have many sensing devices that cooperatively collect and exchange detailed information about physical environment [29] [8]. They may sense information about temperature, sound, brightness, vibration, motion, and so forth.

Surveillance is an area that make intense use of wireless sensor networks. Depending on the application and the mission requirements, the WSNs may use static sensors, mobile sensors, or both. SOSUS is an example of static WSN. The acoustic sensors are deployed at the bottom of the ocean [28]. Mobile WSNs are composed by nodes that cooperate in a opportunistic way and may be present at mobile phones [29] or even on vehicles [30]. Hybrid WSNs makes use of both static and mobile sensors to increase information detection precision and comply with some system requirements. This is the case of the surveillance system proposed by [31] where ground sensors and air sensors cooperate to decrease the communication overhead between the nodes. A large number of cheap ground sensors are deployed to cover wide areas while a few sophisticated air sensors collect the information requested by the static sensors.

Figure 2.1 presents the compiled WSN taxonomy from [31], [32], [33], and [34] and covering the possible distinct types of WSN.

WSNs are also used to support some MANET applications. Mobile sensors placed at Unmanned Aerial Vehicles (UAVs) are able to accomplish missions in wide regions where the mobility at the ground is unsafe. They are able to collect information to support ground agents as fire fighter teams spread in an large area with several fire incidents.

This research uses a WSN as a component of the intrusion detection system designed to MANETs. Each sensor node will collect information about the electromagnetic emissions at the MANET to provide valuable information about the nodes position. It will also be used as a platform to run the detection algorithms of the proposed IDS.

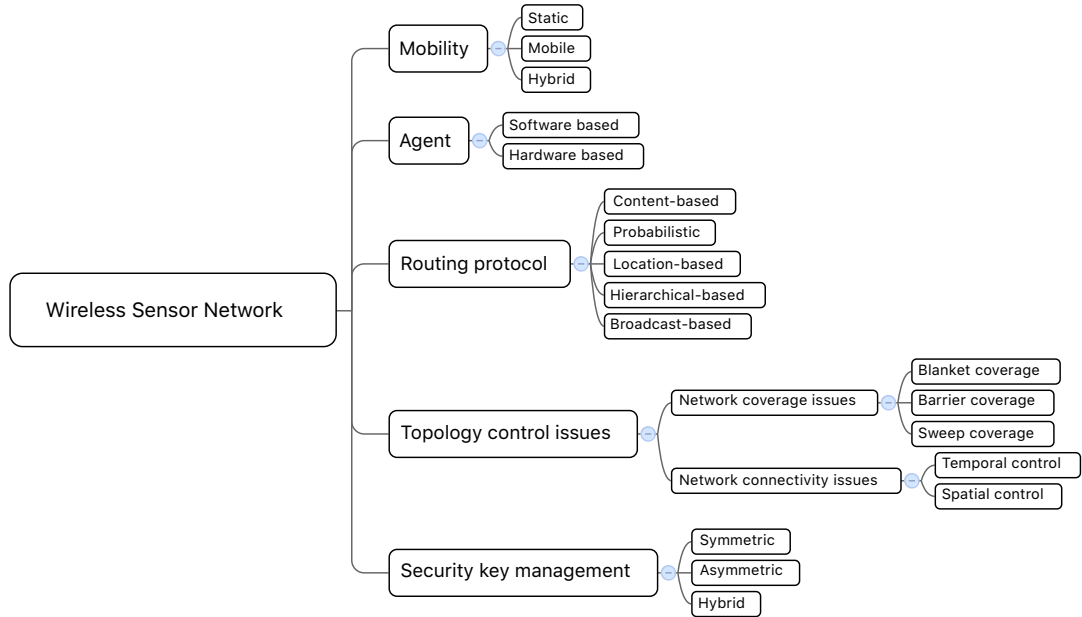


Figure 2.1: Compiled WSN Taxonomy.

2.2 Intrusion Detection Systems

It is not possible to precise when the first Intrusion Detection Systems was initially introduced but these systems started to attract more attention in the digital communication field in the 70's. An USAF paper published in 1972 stated that the US Air Force had “become increasingly aware of computer security problems” [35]. At that time, the Air Force had begun to use shared resources for both classified and unclassified information, and improper access of information became a main concern. In the following decades networks got more sophisticated, while attacks started focusing on unauthorized access. A significant amount of IDS frameworks and prototypes have been designed as well some taxonomies [2].

The modern concept of Intrusion Detection System understands an IDS as a set of tools, methods, and resources used to identify, assess, and report intrusions [7]. It is inserted into the concept of Defence-in-Depth: a defense and surveillance system should be composed by a sequence of layers that protect different aspects of the target network.

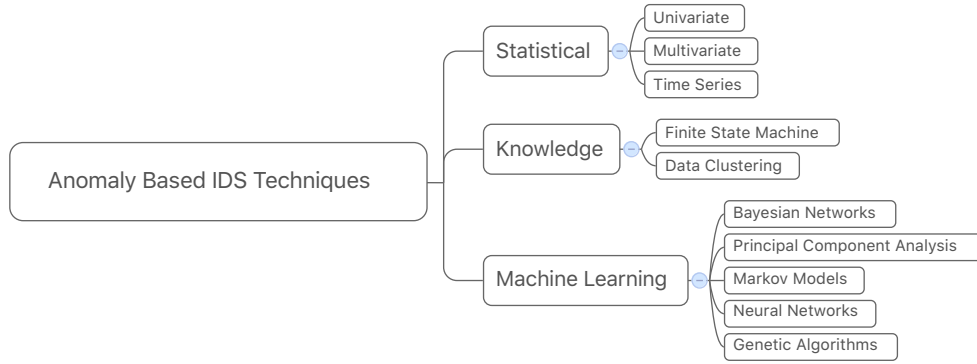


Figure 2.2: Anomaly based IDS techniques.

There are several techniques an IDS might use to detect an intrusion and they are usually categorized according to three groups [7].

The first is the *anomaly based detection group* and covers statistical, knowledge and machine learning techniques to identify anomalous operations and transmissions at the network. These techniques are accurate and work well with new attack patterns but they require an increased computational effort [36]. This over processing and extra network traffic imposes a limit to the adoption of these techniques on MANETs. Figure 2.2 outlines some of the anomaly based techniques.

The second group is the *misuse based technique group*. It uses signatures and rules techniques to build the profile of known attacks. The profiles are used to identify and avoid the attacks. These techniques are more accurate than those of the first group but the trade-off is their lower performance on unknown attacks due the absence of the profile [37]. Techniques in this group are easier to implement in MANETs since the rules tend to be relatively simple and can be used to detect attacks. Figure 2.3 shows the misuse-based techniques while Figure 2.4 outlines some of the attacks that can be avoided using misuse based techniques.

The third group is the *specification based technique group*. It uses specifications and constraints to describe the correct operation of the network and identify any different behavior.

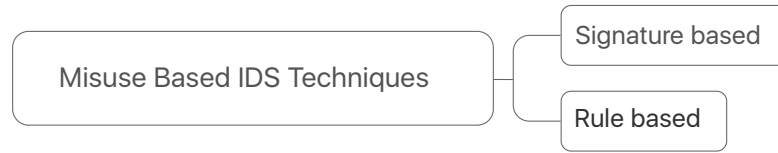


Figure 2.3: Misuse based IDS techniques.

It may be seen as a combination of the advantages of the others two groups but designed with a different approach. When compared against the misuse based technique, one key difference is its use of “signatures” and “rules” to verify whether a given behavior is legal. This leads to the advantage of the technique being able to identify previously unknown attacks. When compared against the anomaly based technique, a key difference is its use of manually developed specifications, which leads to a more accurate and less computational demanding technique. The downside of the specification based technique is the development effort of the specifications and constraints. This phase is usually very time-consuming [38]. The specification based technique is the one used by the CMIDS Framework and further details can be found in Chapter 3.

A lot of distinct IDS have been implemented along the time and the work of [39] lists more than ninety. Besides the classification of an IDS according to the detection techniques used, there are other characteristics that can be used to differentiate and compare any two IDS against each other. The taxonomy presented by [7] and [9] covers the most important aspects used to classify intrusion detection systems. The joint taxonomy covering the work of both papers is pictured at Figure 2.5.

Although the classification of the proposed IDS according to the joint taxonomy does not directly influence on the results of the research, it is important to properly classify it in order to enable a easier comparison between CMIDS and other intrusion detection systems.

From the several IDS that exist, only a few are designed taking into consideration the particularities of MANETs as the lack of concentration points to collect and analyze the

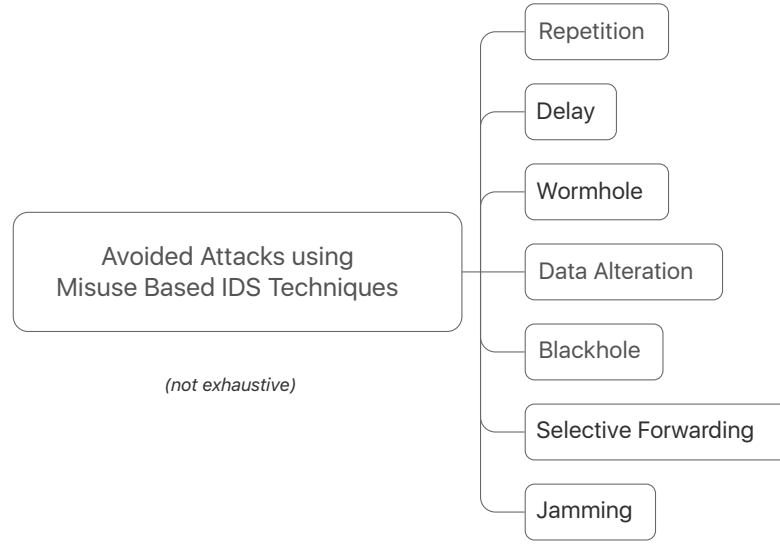


Figure 2.4: Avoided Attacks using Misuse based IDS techniques.

data, the attack opportunities created by the routing role performed by every node, the dynamic and unpredictable connections and disconnections between nodes, the usual CPU limitation power of the nodes, and the facilitated access to data traffic due to the use of an open medium to transmit the information. [2].

One of the most popular IDS specifically designed for securing MANETs is the Watchdog [12]. This IDS works promiscuously listening to the next hop's transmissions. Every time a node fails to forward a data package within a pre-established amount of time, the Watchdog IDS decreases its confidence on the node until the point it decides an attack has been detected. Although very popular, this scheme fails to detect adversaries in the presence of collisions and limited transmission power.

The EAACK scheme [13] tries to overcome the Watchdog issues through the modification of the routing protocol. Each destination node of a message is forced to send back an acknowledge data package informing the sender node when the message was received. This scheme lowers the false positive detection but it requires the modification of the routing protocol, increases the network traffic, and it is still vulnerable to collisions issues.

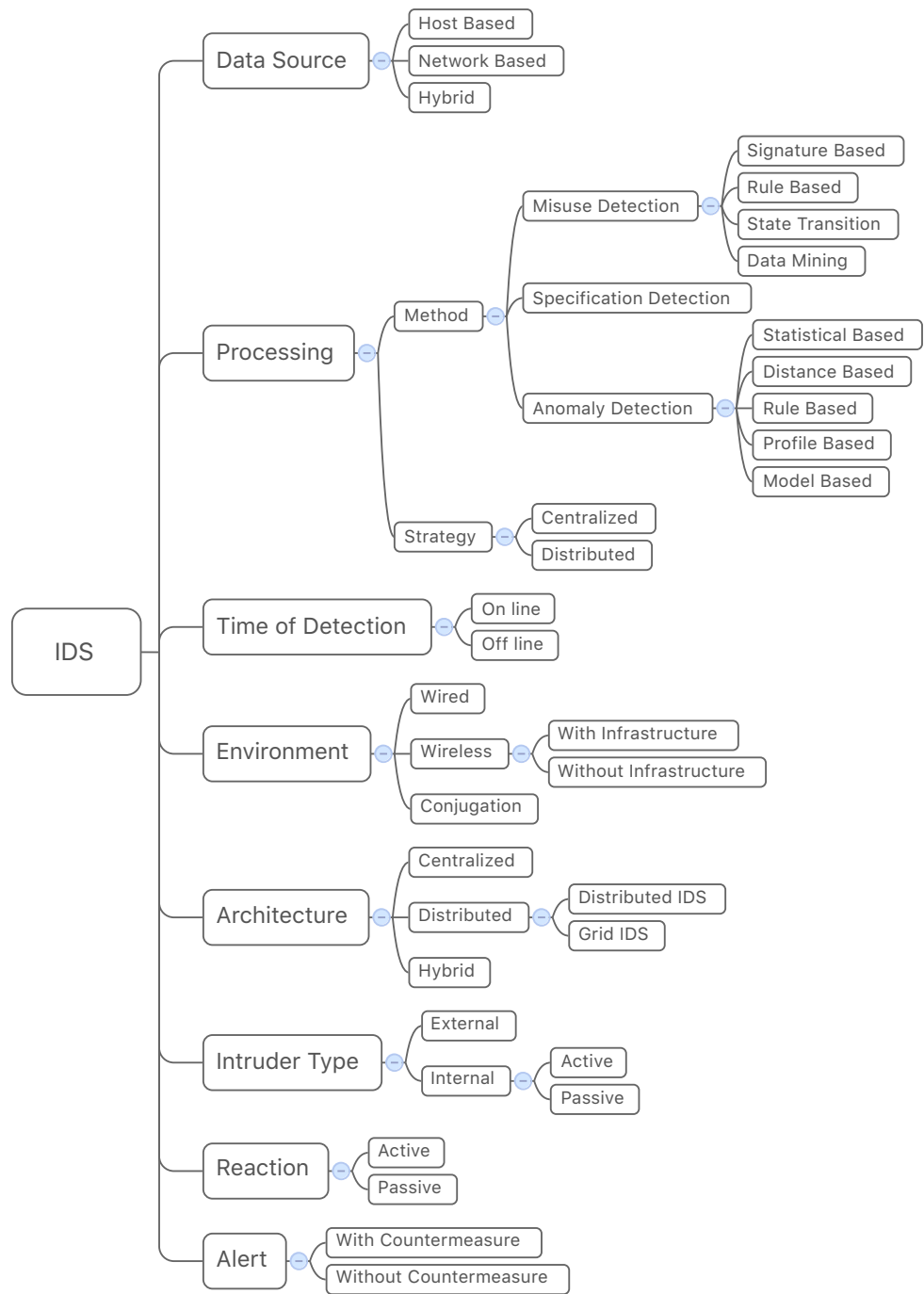


Figure 2.5: Proposed IDS Taxonomy.

CRADS [14] adopts a different approach and uses Support Vector Machine (SVM) to learn the regular behavior of the network and then detect anomalies. Besides being intrusive as EAACK, CRADS requires extra time spent on the learning phase and does not fit well scenarios that require fast deployment and operation.

IDSX [15] is another IDS scheme designed to work with MANETs. It assumes the mobile network is divided into clusters which enable the nodes to communicate with each other. Every node of a cluster collects information and share it with the cluster head but only the last one is responsible to decide if a suspect node is an attacker. The issue with this approach is its vulnerability to cluster head failures because no recovery mechanism to elect a new cluster head is defined. The unavailability of a cluster head can let an entire cluster without protection of the intrusion detection system.

Unfortunately, the majority of intrusion detection systems try to detect active attacks against MANETs instead of passive attacks. Watchdog and EAACK seek to detect package dropping attacks. CRADS seeks to detect routing attacks as sinking, spoofing, and rushing. IDSX works as a framework and defines an algorithm to share collected information from the nodes but do not define the IDS mechanism or the attacks it seeks to detect.

This research proposes a novel IDS that considers the characteristics of MANET since the design phase and seeks to detect passive attacks instead of only active attacks. The inference here is that an IDS trying to detect passive attacks would enhance the security on MANETs because the adversary would be detected before damaging the MANET.

2.3 Position Information Estimation

The position information estimation is the key element of the proposed IDS in this dissertation. As mentioned before, this novel IDS seeks to detect passive attacks performed against MANETs. In order to achieve this capability, the IDS will use the real position of the nodes as input data to algorithms that will analyze the moving patterns to detect the adversary. It is necessary to understand how the position information estimation methods work to elect the one that better complies with MANETs.

In fact, MANETs already use position information data but never with the purpose of detecting intrusions. It is used in geographic routing protocols, navigation systems, and command and control applications [40]. It is also used to safeguard the physical integrity of mobile nodes at Vehicular Ad Hoc Networks (VANETs) [30].

In all cases where MANETs already use the location information, the data is acquired using self-position propagation. It means that each node broadcasts its own position. As noted at [41], the problem with this approach is that any node can forge its position and this false information has a severe impact regarding both security and performance of mobile ad hoc networks. Therefore, all position information should be securely verified before being used.

In consequence, it is necessary to use a position information estimation approach capable of calculating the position of nodes at the MANET without trusting the position announced by them. Some methods have been proposed to overcome this problem and some of them use sensors to estimate the nodes' positions. These methods are based on traditional triangulation, scene analysis, and/or proximity [42]. There are several techniques related to these methods and they all use at some point the radio signal emitted by the nodes as source of information.

Scene analysis algorithms make use of environment features to detect radio waves signatures. The collected signatures are used to estimate the position of new emitters at the network.

Proximity algorithms uses relative position information provided by a grid of antennas. It evaluates which antenna is closest to the emitter and stores the known antenna's position as the emitter's one. The method is relatively simple to implement, but has a drawback on its limited accuracy.

Triangulation algorithms are more popular than the other two methods due to their robustness and good accuracy. They use geometric properties of triangles to estimate the emitter position [41]. Figure 2.6 captures the main techniques used by the discussed methods.

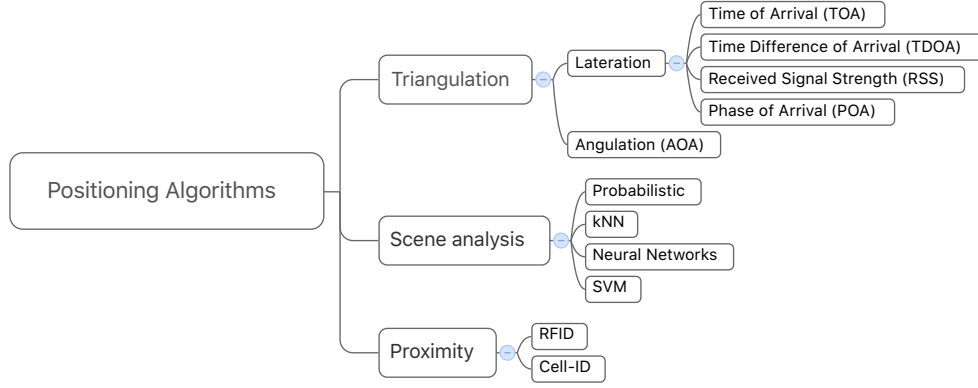


Figure 2.6: Position Information Acquisition Techniques.

The triangulation is the method more indicated to be used with MANETs. The scene analysis method requires previous detailed knowledge of the environment to work properly which is not always available. For instance, after a natural disaster the urban environment is altered and updated details of it are not yet available at the beginning of the search and rescue missions. Proximity methods are not indicated because they require fixed infrastructure. Besides contradicting the purpose of using MANETs, reliable fixed infrastructure is not always available at tactical scenarios as natural disaster operations and military operating in hostile territory.

There are five basic types of triangulation methods. The first one analyzed is the Time of Arival (TOA). This method estimates the distance between two nodes based on the time elapsed to receive an emission. This method by itself does not estimates the position of a node but its principles are used by other methods as TDOA to calculate the position of a node [43].

The Received Signal Strength (RSS) method uses the measured power received at the antenna to estimate the distance between two nodes. It uses the fact that radio emissions strength reduces along with the growing distance of the radio propagation [44]. The position information is estimated using the distinct measures of distance between one node and

other sensors. Clock synchronization between the sensors is required in order to calculate the position of the node. The disadvantage of this method that prevents it to be used with MANETs is the fact that radio emissions strength measurements tend to fluctuate according to changes in the environment or multipath fading. As the node moves, different obstacles interpose between it and the sensors and let distinct sensors use different environment conditions to estimate the position of the same emission. As a consequence, false distance estimations prevent the position estimation of the node.

The Phase of Arrival (POA) method uses the phase difference of the same emission received by an array of antennas to calculate the position of the emitter [45]. This method requires line of sight (LoS) between the sender and the receiver node and is not applicable to MANETs because these networks are used at scenarios where it is not always possible to keep LoS between the nodes.

The Angle of Arrival (AoA) method is used by a node to determine its own position and requires at least two fixed emitters which are used as reference points [46]. The node evaluates the direction of the incoming signals from the fixed emitters and then calculates its own position using the angles formed between itself and the reference points. Although it is a position estimation method, AOA does not satisfy the requirement of the proposed IDS. This study needs a position estimation method to calculate the position of other nodes instead of itself position.

At last, the Time Difference of Arrival (TDOA) method estimates the position of a node through the examination of the differences in time of the same emitted signal received at different sensors [47]. The conventional TDOA position estimation technique is a problem of solving a set of hyperbolic equations. A common time reference must be shared by the sensors and clock synchronization between them is required in order to calculate the position of the node [43]. Figure 2.7 from [48] shows the example of the two dimensional position estimation of emitter "X" performed by sensor 4 using the time difference related to the sensors 1, 2, and 3. According to [47] [40], the distance based technique TDOA is typically referred as multilateration and this nomenclature is also adopted in this dissertation.

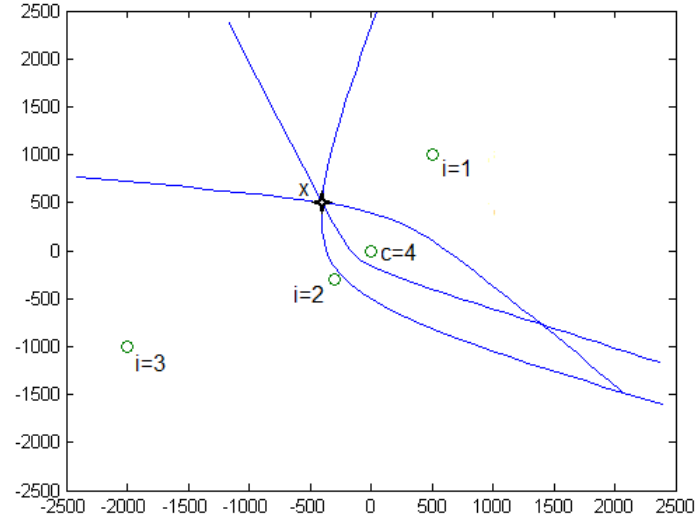


Figure 2.7: Example of Position Estimation using Multilateration.

A multilateration system may use a 3D or 2D-solution to estimate the emitter's position according to the number of antennas that capture a particular signal. When four distinct antennas capture the same signal, it is possible to estimate the 3D position of the emitter (latitude, longitude, and altitude). But if only three antennas capture the same signal, the 3D position cannot be directly estimated. In such cases, one dimensional component must be formerly know and this approach is called 2D-solution. For instance, the altitude of a emitter may be acquired using a secondary system and used as input to the multilateration algorithm.

Multilateration has been successfully used as a navigation technology and surveillance tool [49]. An important success case is the use of the multilateration method by the Federal Aviation Administration (FAA) to track aircraft close to airports [50].

The multilateration method also has a great potential to be used as a source of position information to detect intrusions on MANETs. The closest there exist of an IDS using position information is the work of [41] but they focused on not using extra sensors to

evaluate possible cheating nodes. Two problems comes with this approach. The first one concerns the impossibility of some mobile nodes carrying extra sensors. The second problem is the over computational processing imposed to existing nodes and error estimation due to processing delay [40].

Different from previous attempts of introducing position information to intrusion detection systems on mobile network, this research addresses the specific problem of MANET IDS from an novel perspective. It uses a wireless sensor network to estimate the position on the nodes of the MANET. This non-intrusive approach avoids modifications to the existing MANET and error estimation due to processing delay on overloaded nodes. This research also opted to use multilateration. The study of existing methods for position estimation reveals that this is the most appropriate method to be used with MANETs.

2.4 Key Management Protocol

This dissertation proposes a MANET IDS that uses a WSN to collect, share, and process location information in order to detect passive attacks. Besides the definition of the communication protocol of the IDS at the application level, it is also necessary to define the security scheme concerning the key management protocol for data encryption.

A key management protocol is the set of previously established definitions, techniques, and procedures used to generate, save, distribute, and replace cryptographic keys among a group of users [51]. It encompasses three main phases according to [52]: establishment, management, and certification. The key establishment is the set of process whereby a cryptographic key becomes available to the group. The key management is the set of procedures supporting the establishment and the maintenance of ongoing keying relationships between the group users. The key certification is the mechanism responsible to ensure the integrity, confidentiality, and authenticity properties related to the cryptographic keys.

There are different key management schemes according to the type of cryptography used. The authors in [53] analyze the advantages and disadvantages of using symmetric key, public key, hybrid key (symmetric key + asymmetric key), and group key schemes with

mobile ad hoc networks like WSNs.

The computation cost of symmetric key schemes is lower than public key schemes. Also, the former is considered a better fit than the latter for resource constrained networks such as WSNs. Another disadvantage of public key schemes is that they require a greater number of messages to establish a private communication, which makes it more vulnerable to denial of service attacks. On the other hand, public key schemes are more secure and provide one-to-many verifiability, while symmetric key schemes can only support one-to-one.

Group key schemes have low computation cost and require a smaller number of messages to establish private communication between the network users, but they are not resilient against inside attackers. Hybrid key schemes are very advisable provided that the use of public key cryptography is restrict as far as possible [53].

This dissertation opted to use a hybrid key scheme. The initial communication between two sensor nodes will use a public key scheme to establish the symmetric key that will be used for the rest of the communication or while it is valid. In such way, the overall scheme will benefit from the strength of the public key scheme to define the session key while using it only for a short period of time to limit the message exchange cost and the time exposed to DoS attacks. But at the same time, the overall scheme will also benefit from the lower complexity of the symmetric key scheme while using a session key that requires less computation resources.

2.5 Automated Proof of Security Protocols

The key management protocol that will be suggested at this dissertation will need to be tested against bad implementation. The use of strong cryptography is no guarantee of a good key management protocol [54]. The test will be performed using an automated security protocol proof tool. But before discussing about the available tools it is important to understand the security notion used by an automated security protocols proof.

There are two notions of security. The first notion considers the issues of complexity and probability. It provides guaranteed security against all probabilistic polynomial-time

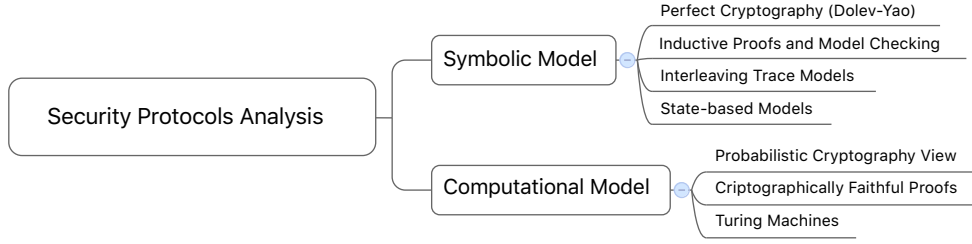


Figure 2.8: Some Aspects of Security Protocols Analysis.

attacks. The second notion considers the protocol as a sequence of modules performing some kind of process and the cryptographic primitives are treated as black boxes. It provides guaranteed security against all attacks that explore the design of the protocol but not against the strength of the used cryptographic algorithms.

According to the security notion adopted, two distinct models are used to analyze the security of a protocol. They are known as *computational model* and *symbolic model* [55]. The former model adopts the probabilistic notion of security while the latter model adopts the second security notion discussed. Figure 2.8 highlights some important aspects of the two models presented by [56] and also discussed here.

The computational model is sometimes also referred to as cryptographic model. It captures a stronger notion of security and protects against probabilistic polynomial-time attacks. Messages are modeled as bit strings and the adversary as a polynomial-time Turing machine. This is a detailed approach, very close to the real execution of the protocols, and provides powerful proofing capabilities [57]. However, the problem formalization is a tedious and meticulous work that is very susceptible to errors.

The symbolic model, also referenced as the Dolev-Yao model [58], uses an abstraction of the protocol execution and captures the protocol behavior through a derivation of a language of algebra terms where the exchanged messages are symbolic terms. Since this model deals with abstractions, the security primitives are assumed to be perfect black boxes [57]. This makes the symbolic model not as sound as the computational model, but its formalization

is much less tedious and error prone. The symbolic model also makes it possible to build automatic verification tools.

The two models have evolved through the years and a considerable amount of effort was made in developing methodologies that would capture the best of each model. The first attempt to unite the computational and the symbolic approaches is found at [59]. In order to combine the two approaches, the authors propose that secrecy properties proved using a symbolic model are considered as true in the computational model. This way, one could achieve benefits of the latter by using the model construction advantages of the first. The result is the possibility of building automatic proofs of protocols in the computational model [60].

Efforts attempting to bridge the gap between the two models resulted into the *Computational Soundness Approach* and the *Direct Approach*. Computational soundness is the approach first suggested at [59]. It assumes an adversary with control of the communication channel and able to capture any message. The main concern is to identify the conditions where equivalent symbolic messages are also computationally equivalent. In contrast, the direct approach uses logics with semantics or symbolic calculi to design proof-rules that are computationally sound [55]. The logics are not complete and the calculi provides just a high-level cryptography construction but they are powerful enough to prove several security properties according to the security notions described at the beginning of this section.

Besides the model, security protocol analysis is also evaluated according to the proof method and the boundedness used [61]. The proof method may be *Model Checking* or *Theorem Proving*. Model Checking is a form of algorithmic verification that seeks for known flaws at the protocols while Theorem Proving reduces verification to proving theorems in first-order logics. The *boundedness* is related to the number of messages used by the adversary at a replay attack. The proofer is bounded if it is not capable of considering replay attacks and unbounded otherwise.

Numerous tools have been built implementing different approaches. Figure 2.9 was extracted from [61] and provides some examples of tools and the corresponding classification.

| | Model checking | Theorem proving |
|---------------|----------------------|---|
| Symbolic | NRL FDR AVISPA | Isabelle/HOL |
| Cryptographic | | BPW(in Isabelle/HOL) Game-based Security Proof (in Coq) |

Figure 2.9: Automated Security Protocols Proof Tools.

The use of automated security protocols proof tools to seek for flaws at the design phase of a protocol is a good practice advised at ISO/IEC 29128:2011 [62]. The choice of the tool, however, depends on how the protocol was designed and the kind of security notion adopted. For instance, CryptoVerif is a good choice for protocols that do not require formal proof and are being designed with new cryptographic algorithms [63]. Although the use of this tool is tedious and time consuming, the user benefits from a tool that tests the entire protocol, including the cryptographic algorithms, and provides security guaranties about polynomial-time attacks. However, if the guaranty is required against any type of attack, a Theorem Proving tool like Isabelle/HOL is indicated, although much more tedious, meticulous and time consuming [64]. In both cases, specific programming languages are used to code the tested protocol according to the environment of tool.

The key management protocol that will be proposed by this dissertation will not use new cryptographic algorithms. Therefore, it is reasonable to consider this algorithms as black boxes and, in consequence, it is sufficient to test only the main steps of the protocol. In this case, a symbolic model checking tool is indicated to verify the protocol. The SCYTHER tool will be used to test the proposed protocol due to its friendlier interface, larger support

community, and higher number of publications when compared against the other available choices [65].

Chapter 3: CMIDS

This chapter presents the Collaborative MANET Intrusion Detection System (CMIDS) as a solution for the research problem posed in Section 1.2, which is to detect passive adversaries who successfully obtained access to MANETs. The first step towards understanding the CMIDS is to establish a distinction between its two key interrelated components: (i) the framework for developing secure MANETs; and (ii) the intrusion detection system that supports it. The first comprises the architecture designed to fulfill the capabilities and the requirements that fit the problem scope and enable the improved IDS. The latter leverages the architecture to detect passive intruders by adopting particular technologies in a synergistic fashion.

The following of this chapter presents the basic premises adopted by CMIDS, the design characteristics that differentiate CMIDS from other intrusion detection systems, and the explication of its architecture.

3.1 CMIDS Premises

The proposed CMIDS Framework addresses the problem of how quickly and efficiently identify an intrusion into a mobile network. It was developed under two main premises. First, it assumes an adversary having access to the network. The second premise is that modifications to the mobile network are not allowed. This means that the solution will **not** be able to (1) replace or even modify the routing protocol, (2) increase the CPU processing at any network node, or (3) incur in any kind of overhead to the network traffic. These premises may be seen as very restrictive, but they are truly necessary to make the solution compatible with the current implementations of tactical networks, especially those used in military applications.

The adversary under the first premise is powerful enough to have access to the mobile network. It may be an authentic node captured by the enemy and turned into an adversary, it may be a new node using stolen credentials, or it may even be a rogue user. In common, all these types of adversary need to keep a minimal safety distance from the other network nodes to perform the attack without being captured. No further distinctions were made between the adversary types. That is, the only considerations in this work resulting from the first premise are the fact of an adversary having access to the mobile network and will keep a safety distance from other network nodes.

The second premise addresses legacy systems. It was meant to ensure the solution is compatible with existing MANETs, instead of imposing restrictions that could only be applicable to new ones. This is especially significant when considering that many already existing MANETs would not be capable of adopting a solution that did not follow the second premise, such as those for which modifications would be technically or commercially unfeasible due to certification or technology issues. For instance, some tactical nodes such as aircraft need to be re-certified before adopting a new equipment or even a new protocol - a time consuming and expensive process. Other examples include network nodes that could not afford modifications impacting the time to process messages and the equipment battery life.

3.2 Design Characteristics of CMIDS

Along the design of CMIDS some decisions about the following aspects were taken:

- Data source
- Detection method
- Processing strategy
- Time of detection
- Environment

- Architecture
- Adversary type
- Reaction and alert

The data source aspect is related to the information available to CMIDS. The data could be captured by the node running CMIDS or it could be collected from the network. CMIDS uses both: data collected by its own and by other nodes. It is also important to notice that CMIDS uses two type of source data. The first type is the self-position data announced by each tactical node. The second type are the emissions from the tactical network captured by each CMIDS node. The emissions' information are shared between the CMIDS nodes during the position estimation process of the tactical nodes.

The detection method aspect comprehends the technique used by the engine of CMIDS to detect an adversary. It could be a misuse-based technique, a specification-based technique, or an anomaly-based technique. The latter was excluded as an option because is too computational demanding and not recommended to power restricted nodes of CMIDS. The misuse technique only detects adversaries actively attacking the MANET and is not indicated to detect passive attacks. CMIDS uses specification-based technique. Although it requires extra development effort to specify the correct behavior of the tactical nodes, it can be used to detect passive attacks and it is not as much CPU demanding.

The processing strategy adopted by CMIDS is distributed. As mentioned before, each node captures emissions from the tactical network, processes it, and then shares the related information with other nodes.

The proposed IDS constantly listens the environment seeking for new emission and after processing and sharing it, CMIDS uses detection algorithms to detect intruders. It means that CMIDS is capable of detecting on-line adversaries because every time a new data is captured CMIDS updates its situational awareness and verifies if the MANET is under a passive attack.

The network protected by CMIDS is a MANET. In consequence, CMIDS must be able

to operate on a mobile wireless environment without infrastructure. It also implies that centralized architecture is not appropriate. In a centralized architecture, the detection decision is made on a fixed location, but in a distributed architecture, the detection decision is made at several locations. Considering the MANETs nature in which the nodes arrangement is constantly changing and the disconnections are frequent, centralized architecture could cause the isolation of the decision maker and the inability to detect passive attacks.

CMIDS is designed to detect passive attacks performed by an adversary with high level of advantage according to the first premise. This powerful adversary is an internal intruder that has already joined the mobile network. It can be a rogue user, an adversary using stolen credentials, or an adversary using stolen equipment. Although powerful, the one thing this adversary can not do is to mimic the movements of a tactical node performing legitimate actions according to the mission. Taking as an example a capture the flag mission as represented at Figure 3.1, the adversary must keep a safe distance in order to not be recognized by the tactical nodes and it is prevented from moving directly to the flag location.

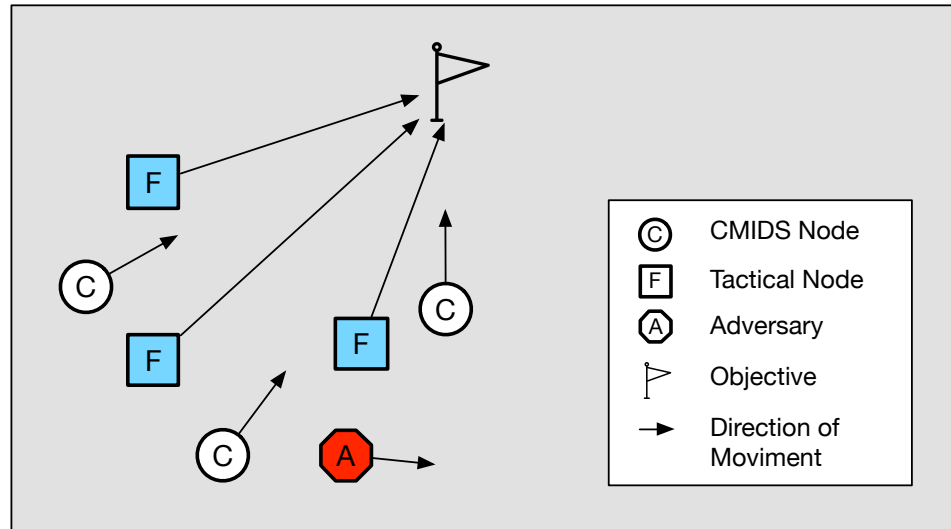


Figure 3.1: Capture the Flag Scenario.

Although particular to scenarios containing attrition between distinct forces, like the ones used by law enforcement teams and military, the mission represented at Figure 3.1 contains several common aspects to tactical scenarios that use MANETs. For instance, like in emergency situations, search and rescue missions, or natural disaster events, the tactical nodes must move from one location to other, messages are exchanged between them to coordinate the mission, there are different possible routes, no infrastructure is available, and a delimited area of action is imposed.

The type of reaction is another aspect that must be treated by an IDS. An active reaction means the IDS is able to adopt procedures to block the attack. Therefore, CMIDS would have to interface with the tactical node which contradicts the second premise. The alternative is to adopt a passive reaction. When a CMIDS node detects a intruder it emits an alert to every other CMIDS node but does not block the intruder. Although the alert does not contains any countermeasures, this alert can be used to trigger a third-party system responsible to disable the intruder.

All these design decisions define the type of intrusion detection system CMIDS is and a classification system is needed to position CMIDS in the context of the subject on IDS.

Taxonomies are used to classification purposes and it was observed during the design of CMIDS the nonexistence of a taxonomy covering all the important aspects related to the research problem. In order to fulfill this gap, a new IDS Taxonomy is proposed at this dissertation. It facilitates the understanding of the architecture of CMIDS and how it is related to other existing intrusion detection systems.

The taxonomy presented in [7] and [9] covers the most important aspects used to differentiate intrusion detection systems from each other but they are not complete. The work of [7] does not capture the reaction and alert features of an IDS while the work of [9] does not consider how the intruder type affects an IDS.

The joint taxonomy build from the work of these authors is used to classify CMIDS and is depicted in Figure 3.2. From the taxonomy it can be seen that CMIDS is an on-line, hybrid, distributed, wireless, specification based IDS without countermeasures detecting

internal adversaries.

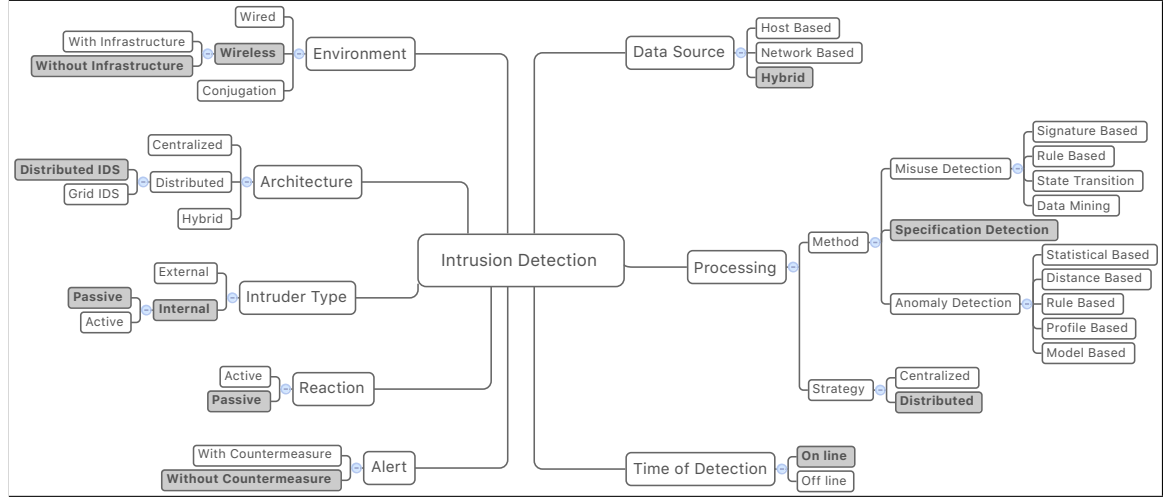


Figure 3.2: Highlighted Intrusion Detection Taxonomy of CMIDS.

It was observed that the second premise influenced the kind of reaction adopted by CMIDS. According to the taxonomy, CMIDS is a passive IDS and does not interface with the MANET. CMIDS is able to receive information from the tactical network but is not allowed to send information to it. This is referenced in this dissertation as a non-intrusive behavior.

The non-intrusive behavior is also noticed in the choice of the platform used to run CMIDS. To adhere to the second premise, CMIDS uses a distinct wireless network. This secondary network uses sensors to capture the emissions from the MANET turning the former on a Wireless Sensor Network. The captured data is shared between the sensors nodes and multilateration is used to estimate the geographic location of the tactical nodes. The WSN nodes are also used to run the detection algorithms and they are distributed in the tactical scenario in the same way as the nodes from the tactical network, but not necessarily at the same positions.

Although CMIDS supports several types of arrangement of the network nodes, it is designed focusing on mesh and hybrid topologies, which are very common in tactical scenarios. Figure 3.3 shows an example of a possible topology combining the tactical and the sensor networks. The tactical node is a regular node of the tactical network while the sensor node is a regular node of the sensor network. The hybrid classification of a node is not related to the taxonomy used to classify CMIDS. The hybrid node is a tactical node able to carry a sensor node of the sensor network. In this case, both nodes share a common mobile platform, such as a vehicle, but work independently from each other, meaning that no modification is imposed to the tactical network. The use of hybrid nodes are desirable for two reasons. First, it uses the available room of the platform to hold the sensor and, as a consequence, the position of the tactical node holding the sensor does not need to be evaluated using the position estimation feature. Second, the presence of a sensor close to the tactical nodes avoids the case of having a group of tactical nodes out of the range of the sensor network.

It is outside the scope of the CMIDS Framework specification to define the exact amount of sensors that are needed. This would cause unnecessary technical constraints and likely incur in suboptimal solutions, since there are many scenario dependent variables influencing this requirement, such as network size, distance between nodes, radio frequencies used, topology, vegetation, local attenuation, and so forth. Instead, the CMIDS specification defines five as the minimum number of sensors. Although only four of these are really need to properly calculate the tactical nodes position, an extra sensor is used (i) as contingency, (ii) to fulfill casual communication disconnections, and (iii) to prevent the sensor network from splitting.

As mentioned before, CMIDS is positioned as a specification based IDS. The specifications and constraints describe the correct network operation. It uses the nodes' real position as input to look for misbehaving nodes that are reporting a location that is not consistent with the multilateration results, are showing mobility patterns not compatible with its capabilities, or are not acting in accordance with the mission plan. The mobility patterns are

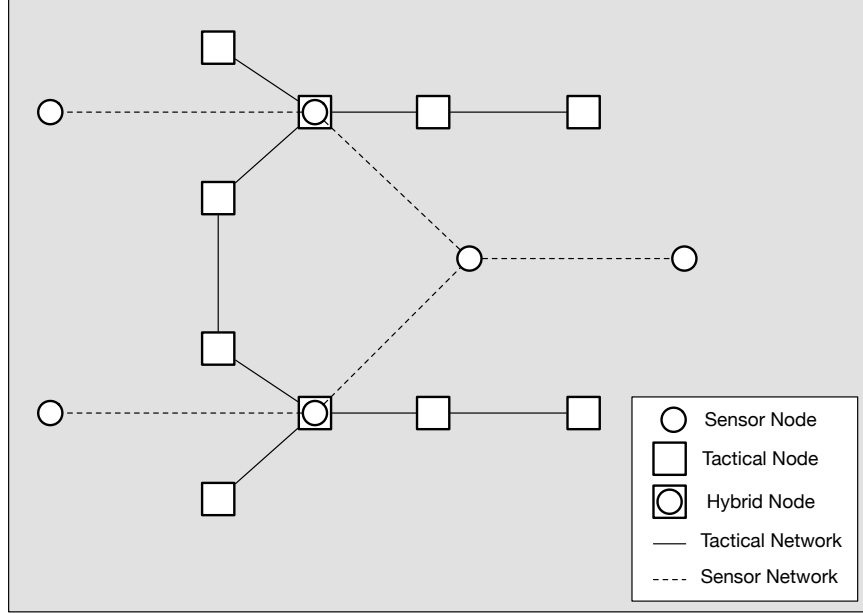


Figure 3.3: CMIDS Network Topology.

checked using the kinetic envelope built by CMIDS. Basically, the kinetic envelope is a set of the possible positions a node can reach from its current position in a certain amount of time. The reached positions must be consistent with the capabilities of the node, terrain and elapsed time. There are four detection specifications defined by CMIDS:

1. $broadcastLocation_{nodeID} \leq |estimatedLocation_{nodeID} - \epsilon|$
 ϵ : multilateration error
2. $speed_{nodeID} \leq maxSpeed_{nodeID}$
3. $speed_{nodeID} \leq maxSpeed_{embarkationID}$
4. $estimatedLocation_{nodeID} \in designatedVolume_{nodeID}$

Specification 1 declares that the broadcast position of a tactical node must be inside the

interval defined by the estimated position and the related multilateration error. Specification 2 checks if the speed of a tactical node is less or equal to the maximum speed of the node according to the terrain at the estimated position. Specification 3 performs the same verification as the former specification but to the case when the tactical node is embarked in another node. Specification 4 verifies if the tactical node is located at a valid position according to the mission plan.

The CMIDS Framework uses true position information of each tactical node to detect intrusions. In order to acquire the true geographic position, the CMIDS implementation uses the emissions' time of arrival (TOA) at different sensors as multilateration input. This approach is convenient to scenarios that mix ground and airborne nodes, as pointed in [47]. According to the authors, using a proper amount of sensors and measurements of the time difference of arrival (TDOA) between sufficient TOA measurements would result in significant improvements to the estimation accuracy of the tactical nodes position. But the sensors are usually static which is not the case of CMIDS sensors. They are mobile and the distance between them change along the time. So, in order to evaluate the position estimation error of CMIDS, I conducted some simulation experiments using the software MATLAB. The source code of the program that calculates the CMIDS position estimation error is at Appendix F and uses part of the work of [66].

A tactical scenario using the simulation tool VR Forces [21] was designed and the position of the tactical nodes, sensors nodes, and hybrid nodes of a hundred simulation runs were saved and then loaded into the program coded using MATLAB to calculate the position estimation error. The simulation varied the sensor's TOA measure accuracy from 95% to 98% with a precision variability normally distributed as $N(0,1)$. The maximum distance between any two nodes was 7 km and it made use of 3 hybrid (collocated) nodes or none. Figure 3.4 shows the simulation results of the position estimation error of CMIDS according to the TOA measures accuracy and the number of hybrid (collocated) nodes used. The 95% accuracy is conservative when compared to the 99.93% used at the work of [47] but it was chosen for the sake of caution.

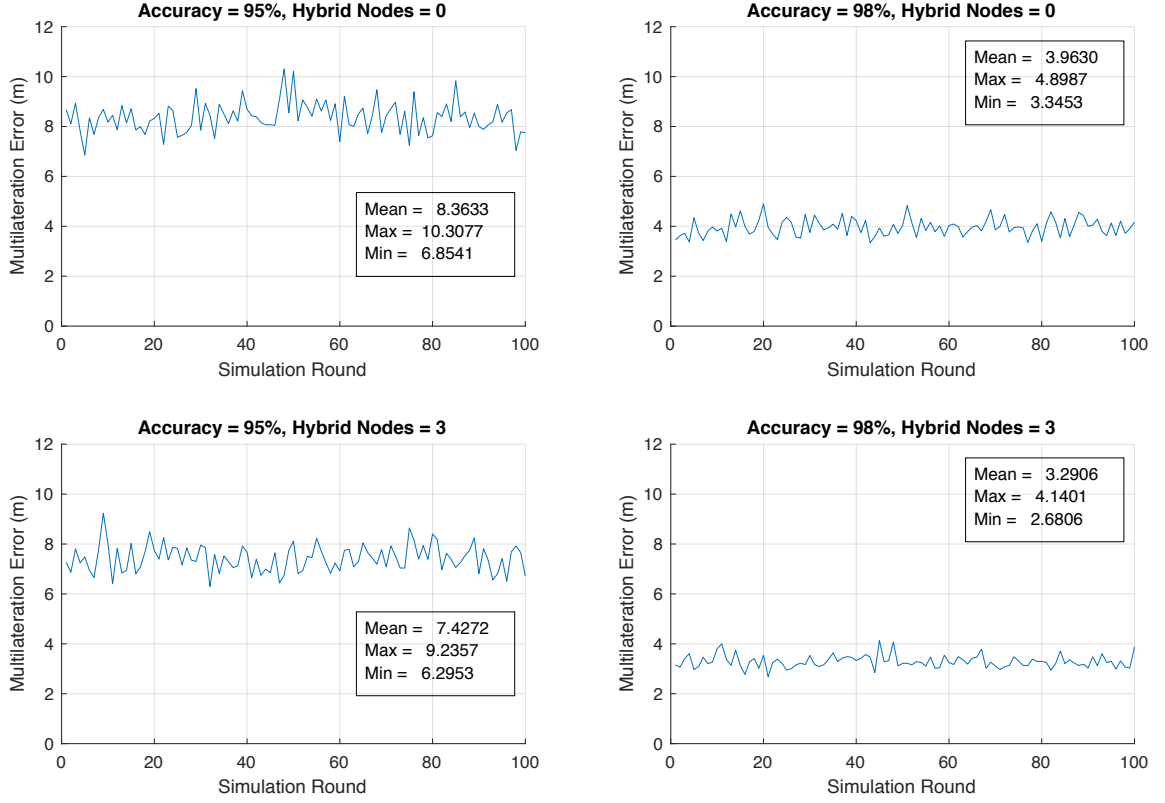


Figure 3.4: Simulated CMIDS Position Estimation Error.

The results summarized at Figure 3.5 show that the use of hybrid nodes improves the position estimation by reducing the multilateration error. Furthermore, the error also reduces when the TOA accuracy is improved. From the simulation results it is noticed that the worst position estimation mean error is 8.36 meters and it occurs when a 95% accuracy and no hybrid nodes are used. The best position estimation mean error is 3.29 meters and is obtained when a 98% accuracy and three hybrid nodes are used. As a consequence, CMIDS uses a position estimation error tolerance of 10 meters, which is sufficiently small to encompass the worst case scenario because just 2% of the position estimation measures are greater than this threshold.

The proposed taxonomy classifies the CMIDS alert as *without countermeasures* but it does not mean the alert messages cannot be used by other IDS. The CMIDS Framework

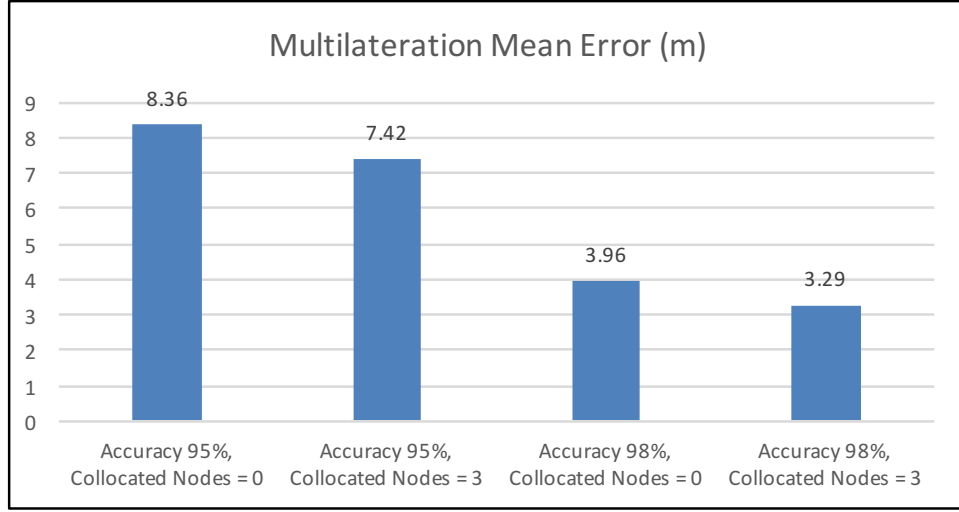


Figure 3.5: CMIDS Multilateration Mean Error.

supports other existing intrusion detection systems already deployed at the mobile network thought the use of a module responsible for translating messages from distinct IDS. From a System of Systems Engineering (SoSE) standpoint, this newly formed system should avoid a breach of security at a subsystem level threatening the entire System of Systems (SoS) [67]. This implies the existence of several networks that must securely exchange messages to prevent cyber attacks from one to the other. Although the IDS message translation module is part of the CMIDS framework design, it is left outside the development scope because its implementation would not affect the hypothesis verification of this research.

Still on the subject about the security of the IDS itself against attacks, advanced approaches were considered to enhance the security of CMIDS. For instance, it was studied the use of physical layer schemes that provide identification assurance such as radio equipment fingerprints identification [68]. This approach uses the fact that channel responses of wireless transmissions are location-specific in multipath environments. The channel frequency responses for different paths are not correlated if the paths are more than one wavelength distant. In consequence, it is very hard for the adversary to spoof another node identity.

Unfortunately, this approach does not work well with mobile nodes because the channel frequency response for the same pair of nodes changes while the nodes move.

In order to enhance the security level of the IDS, CMIDS uses the same intrusion detection engine on both Tactical Network and Sensor Network. But the second premise does not apply to the WSN and countermeasures are used to block compromised sensor nodes. The blocking is performed by a hybrid key management protocol. The advantages of this type of protocol were discussed in Chapter 2 and includes a good trade-off between the use of the public key scheme (more secure and CPU demanding) to define the session key and the symmetric key scheme (less CPU demanding) used during the session. The details of the designed key management protocol are in the Session 3.3.

3.3 Architecture

The CMIDS architecture is a consequence of the design characteristics discussed at Section 3.2. In summary, the architecture must present appropriate components to capture emissions from the tactical network, to share data between CMIDS nodes, to perform multilateration, to calculate the kinetic envelope, to detect an intrusion, and to emit alerts.

Figure 3.6 represents the architecture of CMIDS. The wireless sensor network is used as the platform to run CMIDS. Each sensor node runs the same code and performs the same activities. Figure 3.7 brings the detailed architecture of CMIDS, while Figure 3.8 represents the high-level pseudo-code of the main routine running on each sensor node. Both sensor node types (hybrid nodes and sensor nodes) carry the same structure, and therefore must process the information acquired from the two networks. Each sensor node shares the wireless emissions information captured from the environment with the Sensor Network and evaluates its own belief on the tactical nodes' position. The CMIDS Engine receives the tactical nodes informed position that propagates through the Tactical Network and checks it against the real position calculated using multilateration at the Sensor Network. This procedure identifies obvious discrepancies between the reported and the actual position, which is an indication of a possible adversary trying to spoof a genuine tactical node.

Another possible attack uses captured tactical nodes to gain illegal access to the Tactical Network. These nodes have unusual behavior related to their movements since the adversary must get out of the range from friend troops. A predictive location algorithm inside the Engine evaluates if the node movement is in accordance to its physical capabilities, terrain description, embarkation opportunities, and mission plan. Each sensor node is able to emit an alert to other sensor nodes.

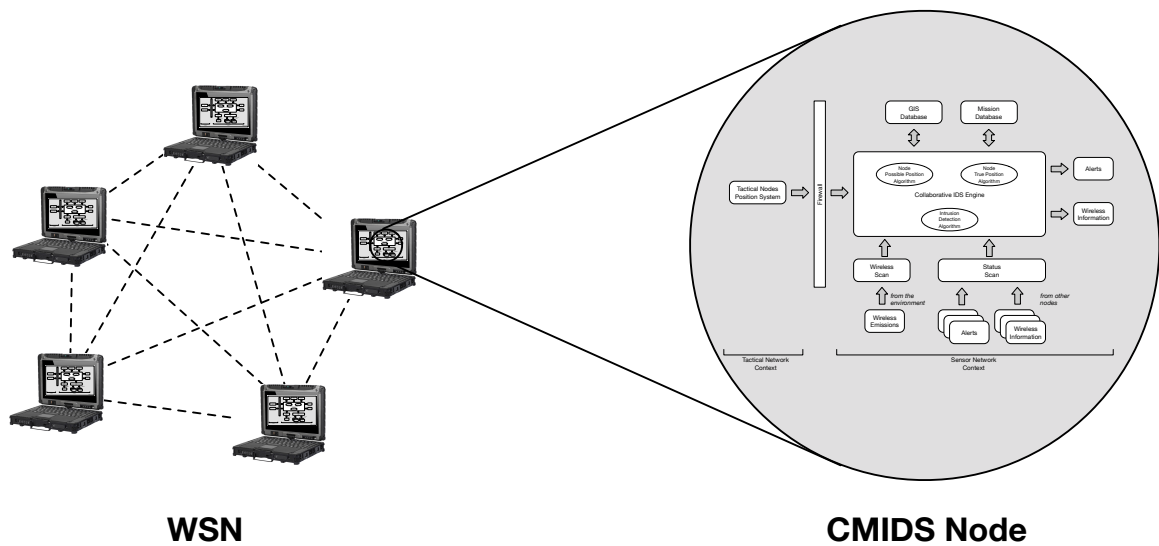


Figure 3.6: CMIDS Architecture.

It is important to notice the existence of two different contexts on Figure 3.7 although it represents the architecture and the functions performed by the CMIDS nodes. The leftmost box belongs to the tactical network context and represent the kind of external information exchange supported by the framework. The remaining boxes represent the functions and the data handled by the CMIDS nodes. The proper way to better understand it is to read both Figure 3.8 and Figure 3.7 together. Line 2 of the algorithm represents the listening to the environment step and it gathers information from different sources according to boxes

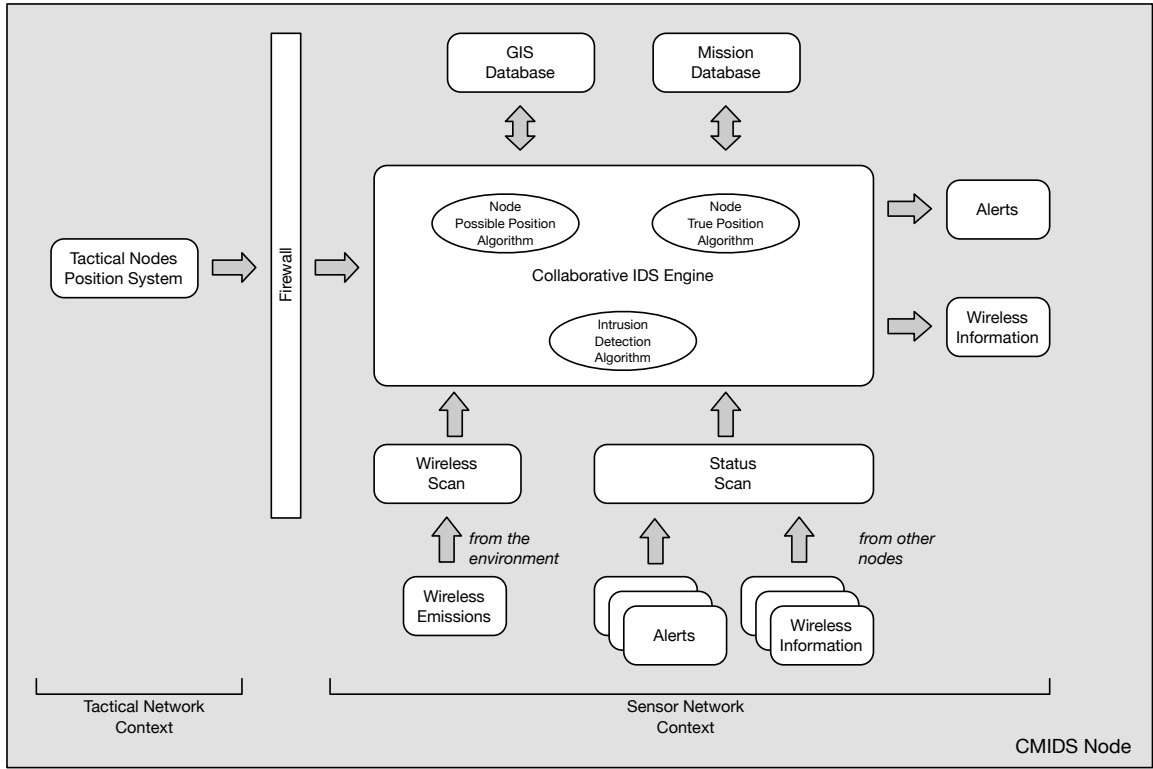


Figure 3.7: CMIDS Architecture at each Sensor Node.

"Traditional IDS", "Tactical Nodes Position System", "Wireless Scan", and "Status Scan". The information serves as input to the algorithm as pictured at the box "Collaborative IDS Engine" and it is processed according to its source at lines 4, 9, and 11. If the information comes from a tactical node, the CMIDS node broadcasts it to all the others sensor nodes as showed at the box "Wireless Information" and the lines 5 to 7. But if the position information comes from a sensor node, it is used but not broadcast as indicated at line 10. The same behavior is verified at line 12 when the information is an alert emitted by another CMIDS node. In order to determine the position of tactical nodes, the algorithm performs multilateration using its own position and the saved information as indicated at lines 3 and 13, and illustrated at "Node True Position Algorithm". Lines 14 to 19 are performed by the ellipse "Intrusion Detection Algorithm" which in turn uses the other ellipse "Node Possible

Algorithm 1: CMIDS Sensor Algorithm

```
1 while true do
2   capture( $\mathbf{e}_i$ ) //  $\mathbf{e}_i$  is a wireless emission;
3    $\mathbf{L}_i \leftarrow$  getSensorCurrentLocation();
4   if comesFromTacticalNetwork( $\mathbf{e}_i$ ) then
5      $\mathbf{E}_i \leftarrow$  extractParameters( $\mathbf{e}_i$ ); // e.g., time,
        sender, destination, payload
6     updateTacticalNodesInformation( $\mathbf{E}_i$ ,  $\mathbf{L}_i$ );
7     broadcastToSensorNetwork( $\mathbf{E}_i$ ,  $\mathbf{L}_i$ );
8   else if comesFromSensorNetwork( $\mathbf{e}_i$ ) then
9     if isSensorBroadcastedEmission( $\mathbf{e}_i$ ) then
10      ( $\mathbf{E}_i$ ,  $\mathbf{L}_i$ )  $\leftarrow$  extractBroadcastedInformation( $\mathbf{e}_i$ )
11      updateTacticalNodesInformation( $\mathbf{E}_i$ ,  $\mathbf{L}_i$ )
12    else if isAlert( $\mathbf{e}_i$ ) then
13       $\mathbf{A}_i \leftarrow$  extractAlertInformation( $\mathbf{e}_i$ )
14      updateTacticalNodesInformation( $\mathbf{A}_i$ )
15    performMultilaterationWithSavedInformation();
16  if broadcastedLocationAtTime[t]  $\neq$ 
    realLocationAtTime[t] then
17     $\mathbf{A}_i \leftarrow$  createAlert(tacticalNode)
18    broadcastToSensorNetwork( $\mathbf{A}_i$ )
19  else if realLocationAtTime[t] not inside
    kinectEnvelope(t - 2, t - 1) then
20     $\mathbf{A}_i \leftarrow$  createAlert(tacticalNode)
21    broadcastToSensorNetwork( $\mathbf{A}_i$ )
22  else if realLocationAtTime[t] not inside
    designatedVolumes then
23     $\mathbf{A}_i \leftarrow$  createAlert(tacticalNode)
24    broadcastToSensorNetwork( $\mathbf{A}_i$ )
```

Figure 3.8: CMIDS Sensor Algorithm.

Position Algorithm”. Alerts are created every time an intrusion is detected at lines 15, 17, and 19 and broadcast at the sensor network as pictured by the single box ”Alerts”.

Figure 3.9 shows the simplified activity diagram of CMIDS. It illustrates the functions performed during an iteration of the main routine loop according to the type of emission received. The diagram provides further details not covered at the high-level pseudo-code. For instance, the multilateration is only performed if the sufficient amount of TOA information from other sensors is available. Otherwise, the iteration ends (final node of the activity diagram) and the loop restarts (initial node of the activity diagram). The details about a

received alert are provided later in this chapter.

Each sensor node shares the wireless emissions information captured from the environment with the Sensor Network and is able to emit an alert to other sensor nodes. As reviewed earlier in Chapter 2, the protocols used to exchange messages should be carefully designed, and automated security protocols proof tools should be used to avoid design flaws in the exchange protocol as well as vulnerabilities to attacks.

Furthermore, CMIDS also uses a hybrid key management protocol to enhance the security of the WSN. In reality, CMIDS monitors the tactical and the sensor networks and is able to issue alerts concerning intruders on both networks. But the WSN is created with CMIDS and is not subject to the second premise. In consequence, it is possible to create countermeasures related to an intrusion detection in the WSN. When an alert is issued, all the sensor nodes save the identity of the intrusion node and no further communication is done with the compromised sensor node.

The communication blockage is easily and quickly implemented because the messages in the WSN are encrypted at the application layer using a unique session key for every pair of sensor nodes. Once a node receives an alert, it saves the intrusion identification, revokes the current session key (if any), and does not establish new connection in the future.

This approach is feasible because the design of CMIDS results in a relatively small number of sensors to achieve the intended coverage, therefore reducing the likelihood of scalability issues. It allows CMIDS to substitute broadcast messages by multiple unicast messages. The communication between two neighbor nodes is performed through a public key scheme to establish the symmetric key that is used to exchange information. The key expiration time is defined at the same opportunity. It is important to notice that the public and private keys of each sensor node are generated during the initial configuration phase of the Sensor Network while it is under complete control of a certification authority.

The main objective of using two different security schemes with the proposed key management protocol is to implement the blockage mechanism used as a countermeasure feature of CMIDS and to use a less demanding encryption scheme during the communication

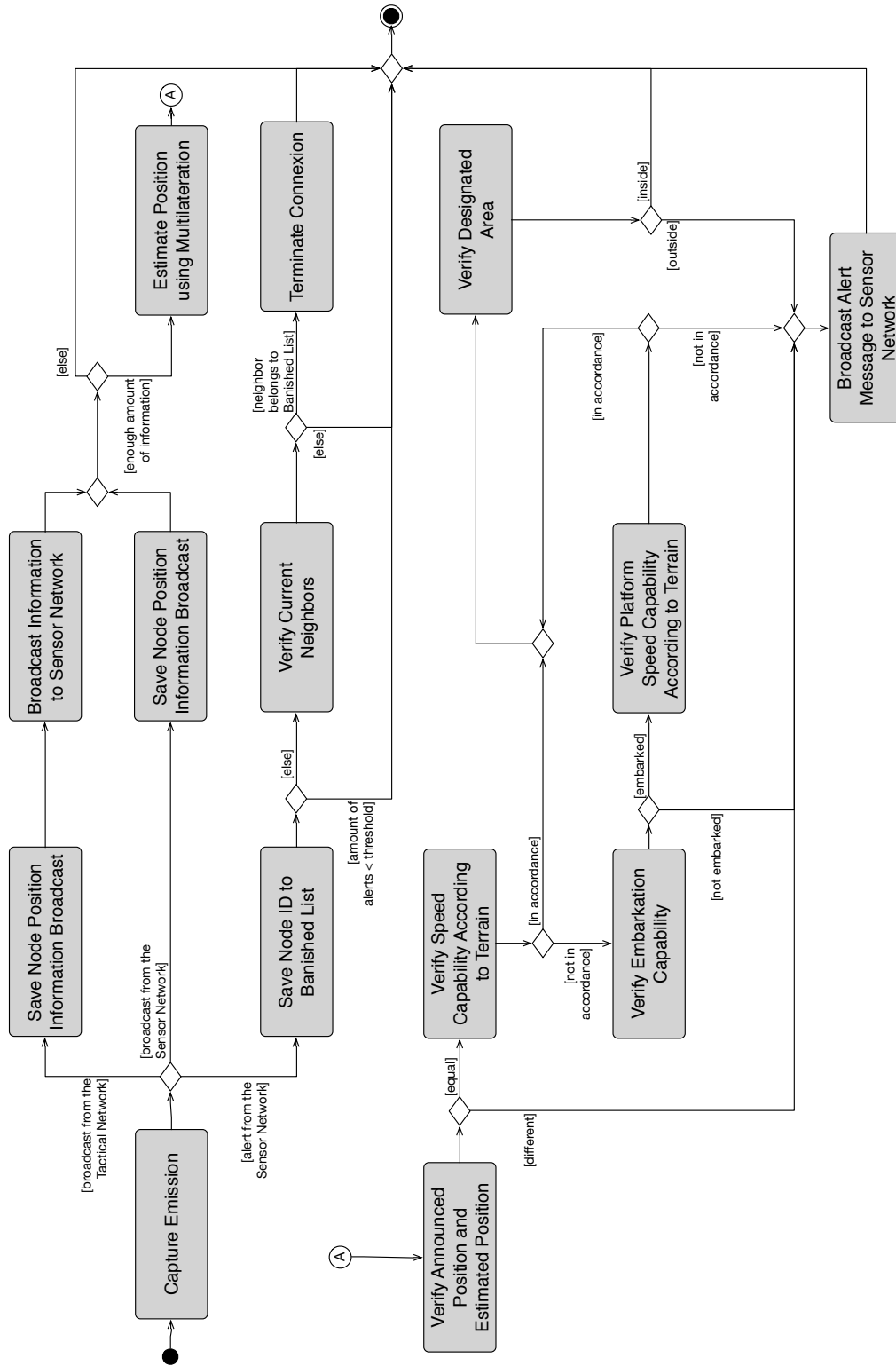


Figure 3.9: Simplified Detection Activity Diagram.

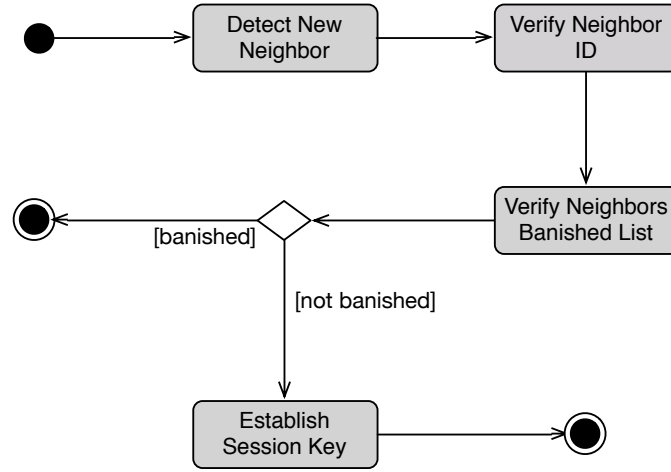


Figure 3.10: Key Establishment Activity Diagram.

between any two nodes of CMIDS. The public key scheme provides the privacy and the necessary authenticity check to verify the identity of the nodes starting a new communication. If an alert has been issued regarding one of the nodes before the start of the communication, the other one do not continue the key negotiation and terminates the connection before establishing a session key. Figure 3.10 shows the activities performed during the key establishment. But if a node receives an alert regarding other node with whom the communication has been established before, the node increments the number of alerts received about the neighbor and terminates the connection if the number is greater than the threshold, revokes the current session key, and mark the other node to avoid future connections. The latter sequence is illustrated at Figure 3.9.

One list and two tables are maintained by each sensor node. The first table contains information about the public keys of all other nodes of the Sensor Network while the second table contains the information about the shared keys between the sensor node and its neighbors. The list contains the nodes that have been compromised and labeled as adversaries. The field *numberOfAlerts* is used to evaluate the belief that a node is an adversary and to avoid an attack from an adversary inside the sensor network. The data structure details

are as follows:

- *Public Key Table* - A table with two fields
 $\langle nodeID, publicKey \rangle$
- *Neighborhood Table* - A table with three fields
 $\langle nodeID, sharedKey, keyExpirationTime \rangle$
- *Banished Nodes List* - A list of all known adversaries
 $\langle nodeID, numberOfAlerts \rangle$

During the operation phase of the Sensor Network, every time a node detects a new neighbor it starts the key establishment protocol. The first step is to verify the *nodeID* using asymmetric cryptography and the *Public Key Table*. Once the identification has been verified by both sensor nodes, they check against each *Banished Node List* to verify if the other node is a known adversary. The next step is to agree on a new shared key. This is done using a signed variant of the Diffie-Hellman protocol, which has been shown to improve the security [69]. The new shared key and its expiration time are saved at the *Neighborhood Table*. Figure 3.11 illustrates the main steps of the key establishment protocol. Variables ni , nr and s are nonces¹ used to support the handshake and the key establishment. On the other hand, although the variables a and b are also nonces used to support the protocol, they are never exchanged between the sensor nodes. Instead, they are used as input to a function and the output is loaded into the messages. The end result of this approach is that both sensor nodes are capable of agreeing on a common key without having full knowledge of a and b .

The proposed protocol may be seen as the concatenation of two security schemes with a checking procedure in the between. Although secure schemes are used, it is still necessary to check the new protocol against design flaws. The check must provide guaranties regarding privacy, authenticity, and integrity. As discussed at Section 2.5, protocols not using new

¹nonce is short for "number used *once*". It is an arbitrary freshly created number used only once in a security protocol.

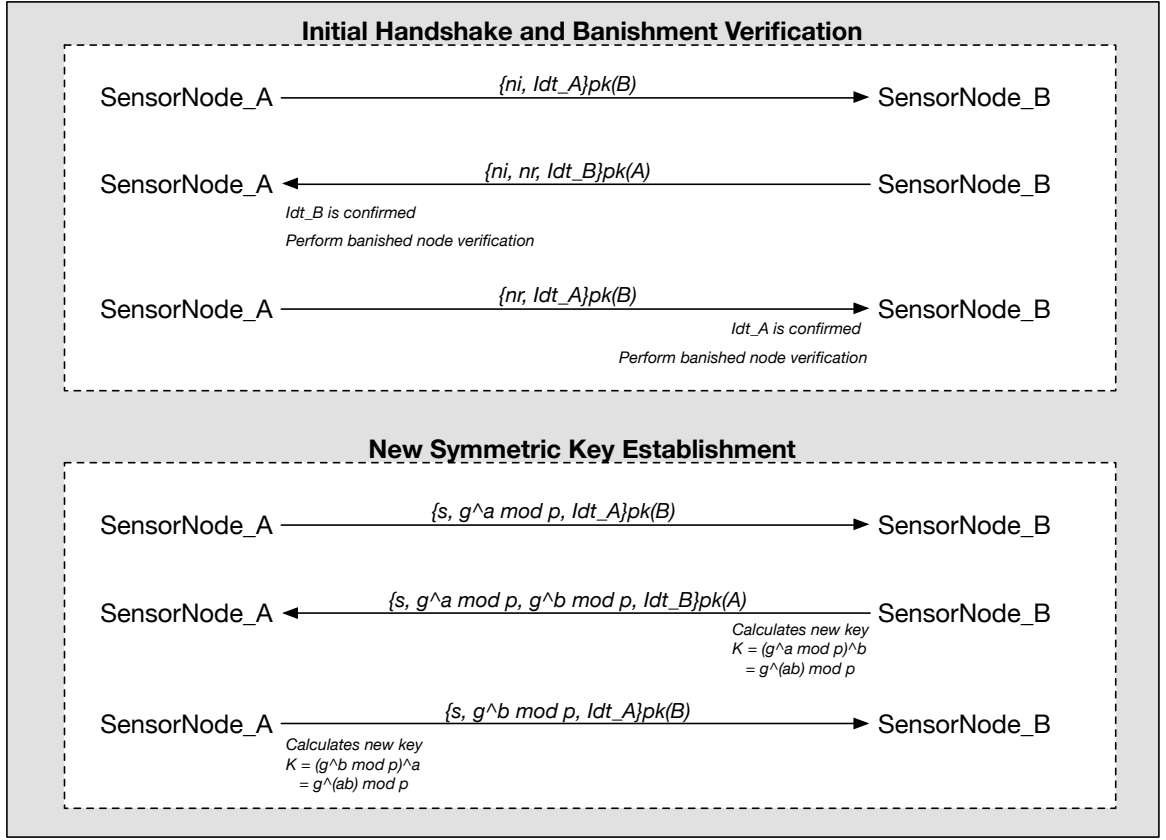


Figure 3.11: Key Establishment Protocol.

cryptographic algorithms may benefit from *symbolic model checking* tools. They provide security assurance for specific claims and are less tedious, less meticulous and less time consuming when compared against *cryptographic* and *theorem proving* tools.

The SCYTHER protocol verification tool [65] was chosen among existing symbolic model checking tools due to its extensive documentation, friendly interface, large support community, and high number of publications. The tool is used to test the proposed key establishment protocol of CMIDS. SCYTHER proves the security properties of a protocol using claim events. Each claim is related to a security property like privacy, authenticity, and integrity. A claim event analyzes if a agent (network node) can be sure about some property of the global state of the system based on the messages it receives (local view).

Four claims defined by SCYTHER are used to check the key establishment protocol of CMIDS: *secret* claim, *non-injective synchronization* claim, *non-injective agreement* claim, and *session-key reveal* claim. The secret claim (*Secret*) is related to the privacy property and states that an information is not revealed to an adversary even when an insecure network is used. The non-injective synchronization claim (*Nisynch*) is related to authenticity property. It ensures that the protocol is executed exactly as it is supposed to be executed. The non-injective agreement claim (*Niagree*) is related to integrity property. It ensures that the nodes running the protocol agree on the values of variables. At last, the session-key reveal claim (*SKR*) is a variant of the secret claim used to additionally mark a variable of interest as a session key. It states that the variable of interest is not revealed to an adversary but the auxiliary variables used to calculate it can be revealed to the adversary.

The source code used to model the protocol using the SCYTHER tool can be found at Appendix A. It creates two main roles: one regarding the node that starts the communication, role *I* at line 30, and one that receives the new communication request, role *R* at line 60. The third role created at line 12 does not correspond to a third node participating on the key establishment protocol. Actually, this third role belongs to the definition of the exponential function using modular arithmetic starting at line 9. Each role *I* and *R* performs three steps: banished node verification step, key establishment step, and claims step. Taking the role *I* as example, the code sequence from line 32 to line 39 defines the variables and the messages used to start the communication and check the identity of the node. The verification if the node belongs to the Banished Nodes List is performed at line 38. The key establishment is performed from line 41 to line 49. It corresponds to the implementation of the messages regarding the signed variant of the Diffie-Hellman protocol. The last step is the definition of the claims being tested. Line 53 and line 54 test if privacy is achieved during the banished node verification step while line 55 and line 56 test if authenticity and integrity are achieved, respectively. Line 57 tests if the established session key is secret. Similar steps are implemented to the role *R*.

It is important to observe that the test result of the security claims at roles *I* and *R*

can be different. The main idea of a claim event is locality. It means that a variable can be considered secret for a node performing role I but not to the node performing role R because the nodes have different local views of the global state of the system. A security property is only achieved if the claim is verified at both nodes.

Figure 3.12 shows the verification results of the tool for both sensor nodes A and B. Here, node A played the initiator role I while the node B played the receiver role R . The results confirm the protocol is immune to attacks. The confirmation of the secret claims for the support variables ni and nr shows they are kept in secret from the adversary. The following claims, $Niagree$ and $Nisynch$, attest the protocol is executed exactly as it was designed and both sensor nodes agree on the values of the variables. At last, the confirmation of the SKR claims proves a new secret shared key was established between the parties.

Scyther results : verify

| Claim | | | | Status | Comments |
|-------------|---|----------------|-----------------|--------|---------------------------|
| newNeighbor | I | newNeighbor,i1 | Secret ni | Ok | No attacks within bounds. |
| | | newNeighbor,i2 | Secret nr | Ok | No attacks within bounds. |
| | | newNeighbor,i3 | Niagree | Ok | No attacks within bounds. |
| | | newNeighbor,i4 | Nisynch | Ok | No attacks within bounds. |
| | | newNeighbor,i5 | SKR g2(beta,x) | Ok | No attacks within bounds. |
| | R | newNeighbor,r1 | Secret ni | Ok | No attacks within bounds. |
| | | newNeighbor,r2 | Secret nr | Ok | No attacks within bounds. |
| | | newNeighbor,r3 | Niagree | Ok | No attacks within bounds. |
| | | newNeighbor,r4 | Nisynch | Ok | No attacks within bounds. |
| | | newNeighbor,r5 | SKR g2(alpha,y) | Ok | No attacks within bounds. |

Done.

Figure 3.12: Protocol Security Verification at Scyther.

In this chapter I presented the Collaborative MANET Intrusion Detection System (CMIDS) as a solution for the research problem. I described the premises shaping the concept of CMIDS and discussed the design characteristics adopted. I presented alternative design options and the arguments supporting my choices. I presented the taxonomy I created to classify and to position CMIDS in the context of the subject on IDS. I also explained the architecture I created and implemented to verify the thesis hypothesis. I proposed a new key management protocol that incorporates a countermeasures mechanism used to block intruders at the CMIDS network. At last, I used a security verification tool to test the security of the proposed key management protocol. All these achievements needs to tested and evaluated in order to verify if they solve the problem of detecting passive adversaries who successfully obtained access to MANETs. The next chapter uses simulation to test and evaluate the CMIDS.

Chapter 4: Test and Evaluation

An IDS designed for MANETs, as mentioned earlier in this work, is a concept for which no specific implementations exist yet. This novelty poses significant challenges on its evaluation, as well as the fact that an actual implementation of the framework is neither practical nor part of the scope of this research. Therefore, computer simulations are used to test and evaluate the CMIDS performance and the contributions claimed by this research. This approach is consistent to a vast body of literature its use for systems similar to CMIDS. As an example, [19] advises employing computer simulations to assess models in which one finds: i) complex systems with many random variables and interacting components with nonlinear relationships; ii) interdependence between resources and system elements; iii) systems that need visual animation of the output. These are all present in CMIDS. Simulation also has some advantages when compared against the real implementation of the system during a first analysis procedure. For instance, simulation studies consume less time to produce results, are more cost effective, provide enhanced control of the system variables, are safer, usually scale better, and can stress the system in scenarios that would not be possible to replicate in the real world [20].

In addition to using simulation as its main approach, the CMIDS evaluation has an important requirement that is not commonly seen in regular network IDS evaluations. More specifically, the research hypothesis being tested requires assessing whether the intruder was capable of performing damaging acts to the network. In order to quantify the intruder's inflicted damage, the evaluation must include an assessment on how much his or her actions were capable of impacting the mission supported by the MANET. As a result of this requirement, a key aspect of the experimental design adopted in this evaluation was that, contrary to most data network evaluation experiments, it aimed for assessing not only the data network characteristics but also their impact on a tactical mission relying on it.

In summary, the CMIDS evaluation requires a simulation environment capable of simulating the operational domain and the communication domain. Unfortunately, any open source simulation tool with this capability was available by the time of this dissertation and, as a consequence, I developed a testbed to evaluate CMIDS. The testbed uses three existing simulation tools, MÄK VR-Forces platform [21], CORE [22], and EMANE [23], and its architecture is depicted at Figure 4.1.

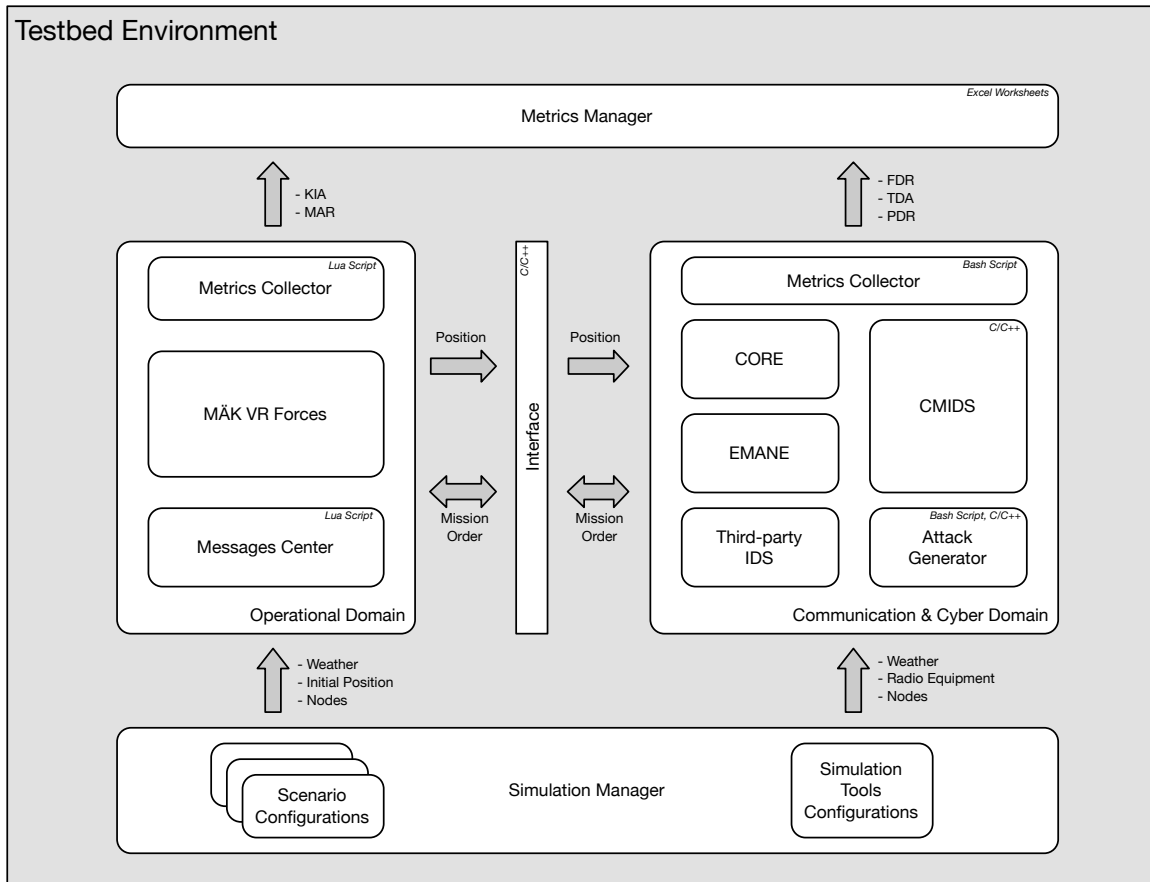


Figure 4.1: Testbed Developed to Evaluate CMIDS.

The test and evaluation of the proposed IDS was conducted through a simulation of a

capture the flag mission, in which the Blue forces (allies) must get inside a small building located at a village under control of the Red forces (adversaries). Although the simulated scenarios are an illustration of military operations, the characteristics of both civilian and military missions as mobility, equipment configuration, and weather conditions are tested. The simulation was designed to replicate the action (e.g., movement of the troops), the data network supporting the action (e.g., communication between troops), as well as the physical characteristics of that network (e.g., timing and propagation issues between wireless nodes). In order to meet the requirements for simulating this environment within the level of resolution necessary to draw valid conclusions on the performance of the CMIDS, three distinct simulation tools had to be used. First, the tactical scenario was conceived and implemented using MÄK VR-Forces platform, which provides an accurate, physics-grounded simulation environment as well as mobility models for ground troops that are precise enough for the goals of this evaluation. Figure 4.2 shows some captured screens from the simulated scenario implemented at VR Forces. The other two simulation tools, CORE, and EMANE, were needed to simulate the data communications network and to implement the CMIDS. CORE is a network simulation tool focused on layer 3 of the Open Systems Interconnection (OSI) model, while EMANE is a real-time simulator of link and physical layers designed to emulate mobile *ad hoc* networks. Both tools are supported by the U.S. Naval Research Laboratory (NRL).

An important aspect of choosing CORE is its capacity of running executable code. CMIDS is coded using the C language (source code at Appendix D), and CORE works by running lightweight Linux virtual machines to simulate the network nodes. This setup allows for running CMIDS without any modifications, which proved to be a key positive aspect in the experiments. Each simulated node of the WSN is able to run the implementation of CMIDS without modification of the source code of the proposed IDS. This allows to test the same implementation of CMIDS that would be used in a real scenario.

Two computers were used at the simulation setup: one was used to run the simulated scenarios at VR-Forces and the other one was used to run the network simulation tools

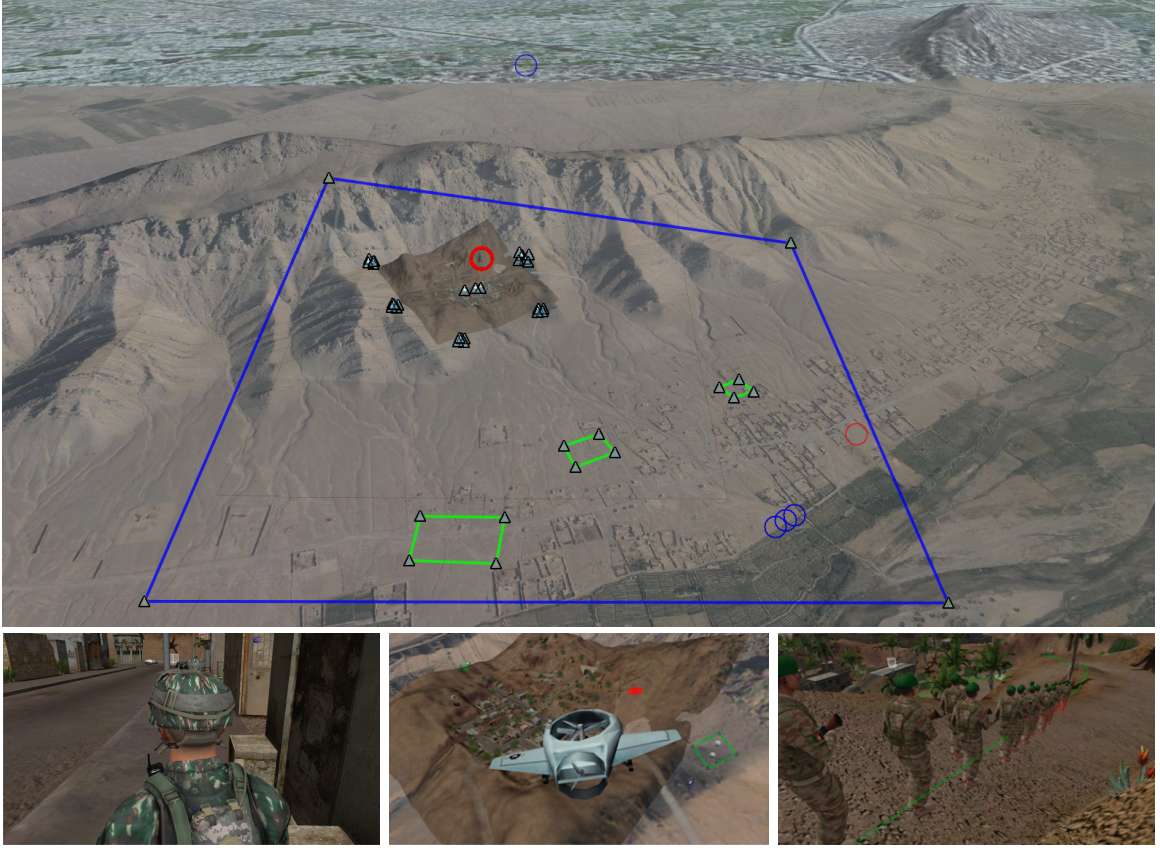


Figure 4.2: Simulated Scenario implemented at VR Forces.

CORE and EMANE, and to run the CMIDS. CORE's ability to run executable code was also used to run part of the interface application built to exchange messages between VR-Forces and EMANE tools. The interface was designed as a modular application, with one module running along VR-Forces at the primary computer and the other module running along CORE at the secondary computer. These modules of the same interface system exchange messages using UDP packages and integrate VR-Forces to CORE and EMANE. Scripted tasks coded in Lua [70] were developed to allow VR-Forces to interface with the modules. The modules were coded using C++ language and built using Qt [71], a cross-platform software development environment. The use of Qt was required to allow the same code to be compiled to the Windows machine running VR-Forces and also to the Linux

virtual machines running CORE. The scripted tasks and the source code of the modules interface are in Appendix C.

The interface uses two types of message: position location message; and mission order message. The former is used to send and receive the position information of the nodes. The latter is used to send and receive orders exchanged between tactical nodes. The UDP payload size varies according to the message type and usually consumes 34 bytes. The message fields are described in the following.

- *Position Location Message Payload*
 $\langle NEM, latitude, longitude, altitude \rangle$
- *Mission Order Message Payload*
 $\langle NEM, orderType, objectType, objectID \rangle$

The NEM field is the unique identification number of each simulated node. The order-Type field contains the order name and can be a movement or a disembark order. The objectType field specifies the type of object handled by the order and can be a wait area, a way point, or an entity (tactical node). The objectID field contains the name of the simulated object. A few examples are provided for each message type in the following while Figure 4.3 shows the GUI built to monitor the Position Location Messages traffic.

- Position Location Message Payload
 $(4, 47.577131636, -122.129702752, 2)$
- Mission Order Message Payload
 $(2, \text{move-to, waitArea, Area 2})$
 $(5, \text{move-to, waypoint, Waipoint 6})$
 $(1, \text{disembark-entity, entity, Soldier 3})$

The simulation setup used 12 different configurations of the scenario. The distinguish between them is made according to the presence of rain, starting point of the Blue forces, and equipment radio frequency used. Table 4.1 shows the possible combinations. For all

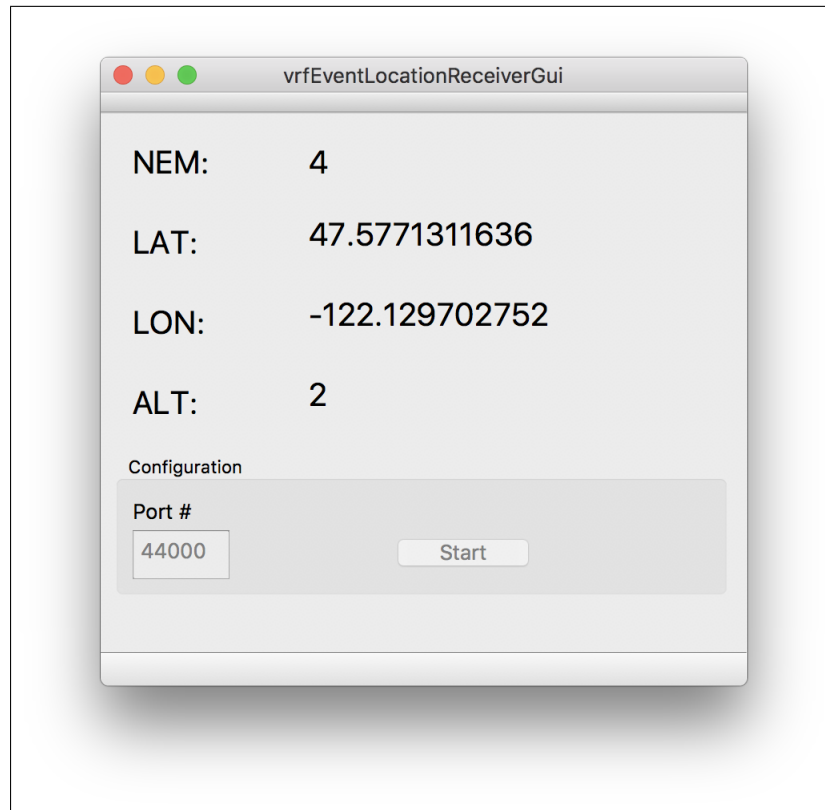


Figure 4.3: Position Location Message GUI.

configurations the simulated scenario was limited to a squared area having 7 kilometers of size.

The purpose of having different configurations of the same scenario is to capture the variables that could affect the efficient of CMIDS and impact the mission. The criteria used to chose the variables was the influence they have on military and civilian missions. The objective is to evaluate CMIDS in the most common environments it would be applicable to.

The rain variable is considered binary in the simulation. A scenario can be sunny or rainy. For the latter, it was considered a heavy rain rate of 50 mm/h. The reason for choosing this extreme environment is to be able to capture how sensitive is CMIDS to rainy weather condition.

Table 4.1: Scenario configurations.

| CONFIGURATION | WEATHER | RADIO EQUIPMENT | INITIAL DISTRIBUTION |
|---------------|------------|-----------------|----------------------|
| 1 | sunny | military | distribution A |
| 2 | sunny | military | distribution B |
| 3 | sunny | military | distribution C |
| 4 | sunny | civilian | distribution A |
| 5 | sunny | civilian | distribution B |
| 6 | sunny | civilian | distribution C |
| 7 | heavy rain | military | distribution A |
| 8 | heavy rain | military | distribution B |
| 9 | heavy rain | military | distribution C |
| 10 | heavy rain | civilian | distribution A |
| 11 | heavy rain | civilian | distribution B |
| 12 | heavy rain | civilian | distribution C |

Three different starting points for the Blue forces are used during the simulation. Although the routes to reach the objective point of the capture the flag mission are random and chosen by the simulation engine of VR-Forces, I chose to use different starting points to enhance the variability of the simulation. Another configuration used with the same purpose is to modify the seed number of the pseudo-random algorithm of VR-Forces for each simulation run.

The simulation setup also used two distinct radio equipment configurations. The objective is to test the performance of CMIDS in MANETs with different range and throughput. A typical military radio equipment configuration was used during the first simulation stage while a commercial off-the-shelf (COTS) radio equipment configuration typical of civilian use was utilized during the second simulation stage. The main difference between the configurations is the used radio band. The former configuration used very high frequencies (VHF) centered at 108 MHz. The latter configuration used ultra high frequencies (UHF) centered

at 2.4GHz. As a consequence, the military radio configuration provides much longer communication range between the tactical nodes compared to the COTS radio configuration. The trade-off is the higher network traffic throughput acquired with the COTS radios.

The number of simulated Blue forces nodes was chosen according to the typical size of a tactical team. Using the platoon size for military missions as a reference, the tactical team size varies approximately from 20 to 50 nodes [72]. Unfortunately, due to limitations of processing capacity of the hardware used to simulate the scenarios, I simulated 24 Blue forces nodes and 5 CMIDS nodes, making a total of 29 virtual machines at CORE. This amount of simulated nodes consumed all the resources available.

The three most common types of attacks to MANETs were performed during the simulation: eavesdropping, denial of service (DoS), and packet dropping [73]. These attacks are common to both civilian and military scenarios. The adversary launched attacks at distinct moments of the simulation from an unmanned aerial vehicle (UAV). During the eavesdropping phase, the adversary learns the Blue Force target location and informs the Red troops about the assault. Figure 4.4 shows the activities performed during the eavesdropping attack. The adversary captures packages during 35 seconds and then decides to attack the node receiving the largest amount of packages. Then, one of the remaining two attacks is launched. During the DoS phase, the adversary tries to delay the Blue forces by injecting extra traffic to the network, which allows for more time to the Red troops for reacting to the assault. The packet dropping phase is more sophisticated. It assumes that either the adversary was able to capture a tactical node or that it is a rogue node. In both cases, the adversary pretends to be a reliable tactical node but silently discards packets instead of forwarding them according to the MANET protocol. To be more realistic, every attack stops 20 seconds after being detected by an IDS. The reason is that, although the IDSs used at the simulation do not have countermeasures, in the real world a detection alert would generate a reaction performed by an external entity.

The packet drop attacks were performed using the blackhole technique, while the DoS attacks were accomplished using application layer DoS attacks targeted to specific tactical

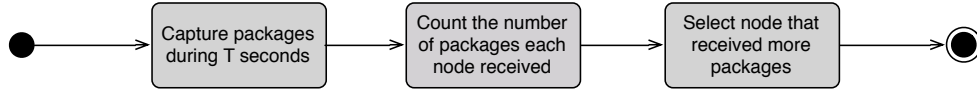


Figure 4.4: Eavesdropping Attack Activity Diagram.

nodes. This type of DoS attack was chosen instead of network layer DoS attack because the former requires less resources than the latter [74].

In order to support the decision about the type of DoS that should be used during the simulation, I decided to test both types of attacks. Figure 4.5 shows the traffic at the attacked node when using the application layer DoS. The dashed line represents the regular traffic of the tactical network while the solid line is the DoS attack performed during 10 seconds at 3 distinct moments. The application layer DoS attack launched from one enemy node was successful and required a packet rate of approximately 1.2×10^5 packet/s. In contrast, the network layer DoS attack using ping flooding technique was not successful when launched from one enemy node. Even a distributed denial of service (DDoS) using the same ping flooding technique and launched from two enemy nodes was not successful. Therefore, it is reasonable to assume that at least three enemy nodes would be required to be successful. But three nodes represents more than 10% of the simulated nodes and no further investigation was made about using network layer DoS because the hardware used to simulate the scenarios was already operating at the limit of its capacity. The choice of using application layer DoS attacks allows a greater number of simulated nodes and, at the same time, do not interferes with the evaluation of the impact of the attack in the mission supported by the network.

The background traffic of the Tactical Network was modeled according to the traffic patterns captured during real missions of the Brazilian Army while using the software of command and control *C2 em Combate*. The corresponding simulated traffic was generated

using the software MGEN [75] and several flows were required to replicate the real traffic behavior. The position announcement performed by each tactical node was treated as separated flows and not included to the general background traffic. This traffic contains the actual position information of the nodes simulated at VR-Forces and are exported to CORE and used by CMIDS. Two programs were written using C language to capture this behavior. One was responsible for broadcast the node position while the other one was responsible for capturing it. The complete source code for the background and position announcement traffic are at Appendix E.

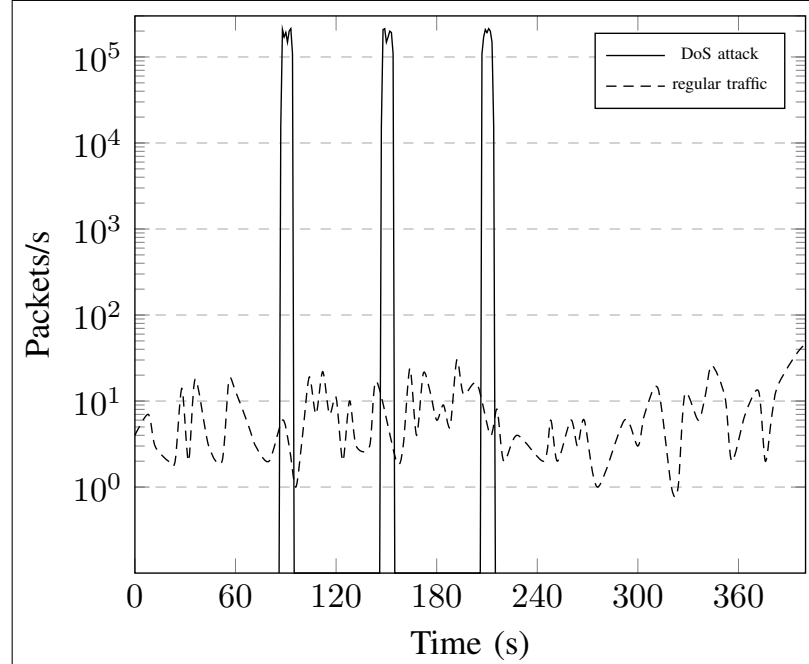


Figure 4.5: Example of traffic captured at sensor node.

Besides CMIDS, the experiments included three other IDS, Watchdog [12], SNORT [76], and SURICATA [77], which were used to provide the baseline performance level and associated metrics against which CMIDS were compared. Providing a fair basic performance

level for comparison that is both representative of the current state of the art in MANET IDS, as well as relatively easy to replicate in a simulation, was a major concern driving the experimental design choices. Just a few IDS designed for MANETs are available, and the majority of them use invasive techniques that require modifications to the network - which is outside the premises defined for CMIDS. Watchdog was chosen because it is used by the most important IDS solutions designed to MANETs [16]. It detects malicious nodes misbehavior related to package forwarding procedures of the routing protocol. In order to attain a baseline IDS system suitable to the networks used in the experiment, I used the watchdog algorithm to build a custom, watchdog-based IDS and used it against the eavesdropping and blackhole attacks. This watchdog IDS implementation continuously listens to the environment with the network interface in promiscuous mode¹ and it emits an alert if a tactical node fails to forward more than 10 packets in less than one minute. The other two chosen IDS, SNORT and SURICATA, were selected for this experiment due to their popularity and capacity of running in both host-based and network-based modes, although they were not primarily conceived for MANETs. They were used against eavesdropping and DoS attacks with an additional rule to detect tactical nodes sending more than 5.0×10^3 packet/s. This threshold is the minimum safe value to avoid detecting burst traffic as a DoS attack.

Although CMIDS is not an intrusion prevention system (IPS), in order to have a more realistic simulation it was assumed the Blue forces were equipped with weapons capable of eliminating threats similar to the UAV used by the adversary once an alert is issued by an IDS. Further, we assumed that after 60 seconds the threat would be eliminated, thus no attack to the data network could be performed after this period. This assumption was based on the author's own experience while working on projects to integrate systems of command and control, field artillery and close air support.

¹*Promiscuous mode* is a mode of the network interface adapter that disables the link layer filter and causes all traffic to be forwarded to the network layer rather than passing only the traffic it is intended to receive

The key metrics used to evaluate the CMIDS are False Discovery Rate and Time to Detect the Attack, although the Killed-In-Action (K.I.A.) rate and the Mission-accomplished rate are also important to understand the impact of the IDS on the mission. Another relevant metric computed is the Package Delivered rate, an output from CORE that is used at VR-Forces, thus impacting how the operation evolves over time. Table 4.2 contains the description of each metric.

Table 4.2: Evaluation Metrics Description

| Metric | Description |
|---------------------------|--|
| False Discovery Rate | It is the ratio between the number of alerts emitted when the adversary is not performing an attack and the number of total alerts emitted. |
| Time to Detect the Attack | It is the amount of time between the beginning of the attack and the issuing of an alert. It will be infinity in case the IDS does not detect the attack. |
| K.I.A. Rate | It is the ratio between the number of blue forces soldiers killed during the simulation and the total number of blue forces soldiers. |
| Mission-accomplished Rate | It is the ratio between the number of successful missions and the total number of simulated missions. |
| Package Delivered Rate | It is the ratio between the number of data packages that do not reach their final destination and the number of data packages sent by the sources. |

The complete simulation roadmap is as follows. The scenario is loaded to VR-Forces and the nodes' position information data is captured using a background scripted task written in Lua, which calls the interfaces modules written in C++ to export the position information data to CORE and EMANE simulation tools running CMIDS, Watchdog, SNORT and SURICATA. Then, each node broadcast its own position to the other nodes inside the CORE+EMANE environment and the package delivered rate is calculated. Finally, the

PDR is used by a script at VR-Forces to adjust the progression speed of the nodes towards its objective (flag position).

The experiment was designed to run and collect data from a total of 2400 simulation rounds for all the simulated scenarios. The time cost to run all the rounds was approximately 400 hours, or almost 17 contiguous days. The description of the computers used during the simulation is detailed at Table 4.3.

Table 4.3: Specifications of the Computers Used at the Simulation Setup

| SPECIFICATIONS | COMPUTER #1 | COMPUTER #2 |
|------------------|------------------------------|-----------------------------|
| Simulation Tool | VR-Forces 4.4 | CORE 4.8 + EMANE 0.9.2 |
| Operation System | Windows 10 64 bits | Ubuntu 14.04.1 64 bits |
| Processor | Intel Core i7-6700HQ 2.6 GHz | Intel Core i7-3667U 2.0 GHz |
| Memory | 16 GB DDR4 2133 MHz | 8 GB DDR3 1600 MHz |
| Storage | SSD internal | HDD 5400 RPM USB 3.0 |
| Graphic Card | NVIDIA GeForce GTX 970M | Intel HD Graphics 4000 |

It is worthy to notice that VR-Forces has specific requirements concerning the graphic card. In order to work, the version 4.4 of VR-Forces requires a NVIDIA graphic card.

Chapter 5: Discussion

This Chapter presents and discuss the performance results obtained during the simulation phase of the research. The discussion of the 12 simulated scenarios is organized according to the type of radio equipment used. As will be presented, this is the variable that has the greatest impact on the simulation results. Section 5.1 analyzes the simulation results obtained using a military radio configuration, while Section 5.2 provides a similar analysis for civilian radio configuration. Section 5.3 provides the attack and detection timeline for both configurations.

5.1 Military Radio Equipment Configuration

To better understand the core aspects of the experiment, this discussion begins highlighting the use of a military radio equipment configuration with the chosen scenario, which led to a network with single-hop behavior. The tactical nodes were directly connected most of the time and did not make use of intermediate nodes to forward the packages that could not be sent directly to its final destination. As a consequence, a blackhole attack would be ineffective because the tactical nodes would not be used to forward packages. Therefore, neither the watchdog IDS nor the blackhole attacks were part of the experiments with this radio configuration.

The other peculiarity of the setup using military radio configuration is the impact of noise due to rain attenuation. No relevant impact was noticed on the network performance, even when submitted to heavy rain. This observed fact is in accordance with the rain attenuation model proposed by the Radiocommunication Sector of the International Telecommunication Union (ITU) [78]. The used radio frequencies are not high enough to suffer from relevant attenuation when considered the maximum node distances of ten kilometers allowed at the

scenario.

The simulation results when using military radio configuration are summarized in Table 5.1. The performance of Snort and Suricata are very similar and therefore no distinctions between them are made during the discussion.

Table 5.1: Summarized Performance Metric (Military Radio Configuration)

| METRIC | CMIDS | SNORT/SURICATA ¹ |
|----------|--------|-----------------------------|
| FDR | 0.8% | 0% |
| TDA | 10.1 s | 36.3 s |
| PDR | 94.3% | 77.4% |
| MAR | 100% | 100% |
| KIA rate | 0.0% | 8.7% |

The first performance metric discussed is the false discovery alert rate and it is depicted in Table 5.2. CMIDS was capable of identifying the adversary along all attacks, while attaining false discovery rate of just 0.81%. In contrast, Snort and Suricata had a performance that seems much better at first but it is not. The 0.0% of false discovery rate of Snort and Suricata seems impressive but hidden is the fact that these intrusion detection systems were only able to detect the adversary while performing the DoS attack. It is also important to observe that the DoS attack was easily detected because the number of packages used is much higher than the usual background traffic. Because of that, the FDR number of CMIDS provides more value. Nevertheless, although small, the false discovery rate of CMIDS still has a great impact on the Blue forces. It means that for every 125 alerts emitted, one would turn a Blue force node into a target. Thus, a secondary confirmation mechanism is needed to avoid friendly fire.

The main cause of the false discovery rate obtained by CMIDS is related to the maximum position error adopted during the simulation trials. One of the mechanisms CMIDS uses

¹Results for the Snort/Suricata are only related to DoS attacks.

to detect an adversary is to compare the position announced by each tactical node against the position calculated using multilateration. If the difference between these two positions exceeds a predefined threshold then CMIDS emits an alert. The problem is related to the communication between VR-Forces and CORE. Every node announces the position provided by VR-Forces at every *simulation tick* but the multilateration mechanism uses the emissions provided by CORE. This characteristic of the simulation model prevents the clocks to be perfectly synchronized. Due to the computer process scheduling nature, the clock synchronization varies along time in an unpredictable way. The adopted 10 meters threshold is quite fair for ground troops but it is occasionally not enough to validate legitimate nodes. For instance, a 0.4 second difference between the clocks would lead a 11 meters position error for a node moving at 100 km/h, which exceeds the threshold. I am confident that once the synchronization issue between VR-Forces and CORE is solved, the false discovery rate obtained by CMIDS will drop but future work is still necessary to find out how much it will.

Table 5.2: False Discovery Rate (Military Radio Configuration)

| STATISTICS | CMIDS | SNORT/SURICATA ¹ |
|--------------------|----------------------|-----------------------------|
| Average | 8.1×10^{-3} | 0.0 |
| Standard deviation | 7.2×10^{-4} | 0.0 |
| Variance | 5.2×10^{-7} | 0.0 |

The second performance metrics analyzed is also related to the emitted alerts. CMIDS took 10.11 seconds on average to identify an attack and emit an alert while Snort and Suricata took 36.31 seconds. This result is even better when we consider the fact that the system was also capable of identifying passive attacks (e.g. eavesdropping). In contrast, Snort and Suricata did not identify the adversary during its eavesdropping attacks. That

¹Results for the Snort/Suricata are only related to DoS attacks.

is, although they presented a 0.0% false discovery rate, it let the enemy to learn about the Blue forces transmissions and, as a consequence, to start the denial of service attack. Table 5.3 shows CMIDS metrics for time to detect an attack.

Table 5.3: Time to Detect an Attack (Military Radio Configuration)

| STATISTICS | CMIDS | SNORT/SURICATA ¹ |
|--------------------|----------------------|-----------------------------|
| Average | 1.01×10^1 | 3.63×10^1 |
| Standard deviation | 7.0×10^{-1} | 6.9×10^{-1} |
| Variance | 5.0×10^{-1} | 4.8×10^{-1} |

The network disturbance caused by the attacks is now analyzed. Considering the previously stated assumption on the Blue forces ability to terminate an attacker within 60 seconds after an alert is emitted, CMIDS was able to diminish the impact of the DoS attacks. This happened because an alert was emitted during the passive phase of the attack, thus allowing the system to provide a timely response. That is, the adversary was still able to start the active attack, but not with the same effectiveness it would have if the alert was not issued. More specifically, the package delivery rate in this case was 94% when using CMIDS. Snort and Suricata presented a worse result because they were not able to identify the adversary during the passive attack phase, thus allowing for a much more effective DoS attack by the adversary that resulted in a greater impact on the Blue forces network. The results showed a 77% of package delivery rate when using these intrusion detection systems.

The communication disturbance at the network when not using CMIDS affected the situational awareness of the Blue forces, forcing a delay at the execution of some operational activities. As a result, the Red forces had more time to respond to the Blue forces assault, which caused an increase on the Blue forces K.I.A. rate but not a decrease on the mission-accomplished rate. When CMIDS was used, the K.I.A. rate was 0% and the mission-accomplished rate was 100%. In contrast, using SNORT and SURICATA, the K.I.A. rate was 8.67% but the mission-accomplished rate remained 100%. At first glance this result

¹Results for the Snort/Suricata are only related to DoS attacks.

may be intriguing because the mission-accomplished rate was not affected by the increase of the K.I.A. rate. The explanation for this lies in the Blue forces' arrangement. One of its groups' formation has shown to be more susceptible to network disturbance and lost more nodes, although the mission was still accomplished. These results clearly suggest that CMIDS provided an increase in the effectiveness of the Blue forces resulting in no loss of human lives.

5.2 Civilian Radio Equipment Configuration

For the second simulation setup, a typical COTS radio equipment configuration was used, and it led to a more dynamic multi-hop network. The tactical nodes were not able to be directly connected to each other in many occasions, so intermediate nodes were used by the routing protocol.

In contrast to the setup using military radio configuration, in the COTS configuration the impact of noise due to rain attenuation was clearly noticeable. According to the rain attenuation model proposed by the Radiocommunication Sector of the International Telecommunication Union (ITU) [78], the specific attenuation γ is obtained from the rain rate R (mm/h) using Equation 5.1. The coefficients k and α are calculated according to Equations 5.2 and 5.3, respectively.

$$\gamma_R = kR^\alpha \quad (5.1)$$

$$\log_{10} k = \sum_{j=1}^4 a_j \exp \left[- \left(\frac{\log_{10} f - b_j}{c_j} \right)^2 \right] + m_k \log_{10} f + c_k \quad (5.2)$$

$$\alpha = \sum_{j=1}^5 a_j \exp \left[- \left(\frac{\log_{10} f - b_j}{c_j} \right)^2 \right] + m_\alpha \log_{10} f + c_\alpha \quad (5.3)$$

Although very subtle, the rain attenuation of 0.063 dB was imposed to the rainy scenarios but no significance difference was noticed in the simulation results. The rain attenuation value was obtained after the evaluation of the above equations considering a rain rate of 50 mm/h and vertical polarization of the antennas. As explained at the introduction of this Chapter, the results discussed incorporate the sunny and rainy scenarios.

The simulation results when using COTS radio configuration are summarized at Table 5.4. Once again, the performance of Snort and Suricata was very similar and no distinctions between them were made during the discussion.

Table 5.4: Summarized Performance Metric (COTS Radio Configuration)

| METRIC | CMIDS | SNORT/SURICATA ¹ | WATCHDOG ² |
|----------|--------|-----------------------------|-----------------------|
| FDR | 0.3% | 0% | 47% |
| TDA | 21.1 s | 71.7 s | 95.5 s |
| PDR | 79.9% | 71.6% | 68.0% |
| MAR | 100% | 84% | 68% |
| KIA rate | 8.3% | 46.6% | 67.3% |

CMIDS was capable of identifying the adversary along all three types of attacks launched against the tactical network, while attaining false discovery rate of just 0.30% as depicted in Table 5.5. Although this rate is lower than the one from the first simulation setup, a statistical Paired *t*-Test was conducted to evaluate if the difference between the metrics is significant. The test is formulated as follows.

¹Results for the Snort/Suricata are only related to DoS attacks.

²Results for the Watchdog are only related to Packet Dropping attacks.

FDR Paired t -Test:

- The FDR on the military setup and on the COTS setup are independent;
- The difference between these measures are also independent;
- The point estimator δ is the sample mean \bar{D} ;
- The null hypothesis is $H_0 : \delta = 0$;
- The significance level is $\alpha = 0.05$; and
- The test statistic is $t = \frac{\bar{D}-0}{s_D/\sqrt{n}}$.

The test statistic t proved to be greater than the percentage point of t -Distribution for 199 degrees of freedom with significance level at 0.05. This indicates there exists a statistically meaningful difference between the two results. In other words:

$$t = 7.748 > t_{199,\alpha} = 1.658$$

H_0 is rejected

The above result clearly suggests that CMIDS shows an improved false discovery rate when running on a network with COTS radio configuration.

Although the FDR results were improved in this configuration, a secondary confirmation mechanism is still needed to avoid friendly fire as suggested before for the military radio setup. Similarly to the rationale used in assessing the impact of FDR for the military radio configuration, in the COTS configuration results for every 334 alerts emitted by CMIDS, one would turn a Blue force node into a target.

Concerning the time to detect an attack metrics, CMIDS took 21.1 seconds on average

Table 5.5: False Discovery Rate (COTS Radio Configuration)

| STATISTICS | CMIDS | SNORT/SURICATA ¹ | WATCHDOG ² |
|--------------------|----------------------|-----------------------------|-----------------------|
| Average | 3.0×10^{-3} | 0.0 | 4.7×10^{-1} |
| Standard deviation | 5.3×10^{-4} | 0.0 | 1.3×10^{-2} |
| Variance | 2.8×10^{-7} | 0.0 | 1.7×10^{-4} |

to identify an attack and emit an alert. This result is not as good as previously obtained because the multilateration procedure took more time to estimate the tactical nodes location. In fact, the tactical nodes transmissions have a shorter range when typical COTS radio equipment are used instead of military ones. Therefore, each sensor captures less transmissions from the tactical network, and becomes more dependent of the wireless information broadcast by other sensors. Since the necessary emissions information used for the multilateration procedure takes longer to be available at each sensor, the time to detect an attacker is also higher. Table 5.6 shows CMIDS metrics for time to detect an attack.

Table 5.6: Time to Detect an Attack (COTS Radio Configuration)

| STATISTICS | CMIDS | SNORT/SURICATA ¹ | WATCHDOG ² |
|--------------------|---------------------|-----------------------------|-----------------------|
| Average | 2.115×10^1 | 7.172×10^1 | 9.551×10^1 |
| Standard deviation | 1.637 | 5.349 | 2.217 |
| Variance | 2.679 | 2.861×10^1 | 4.915 |

Snort and Suricata did not identify the adversary during its eavesdropping attacks and let the enemy to launch a more effective denial of service attack. The difference of performance between CMIDS and these two IDS in detecting the attacks is reflected at the level

¹Results for the Snort/Suricata are only related to DoS attacks.

²Results for the Watchdog are only related to Packet Dropping attacks.

of network disturbance. More specifically, the package delivery rate when using CMIDS was 79.9%, while SNORT and SURICATA only achieved a 71.6% rate.

The Watchdog IDS presented the worse performance between the three intrusion detection systems used at this stage. Its false discovery rate was 47 % and the package delivery rate was 68%. The high false discovery rate is explained by the limited transmission power of the tactical nodes when compared against the area they are deployed. This characteristic has been recognized by other researchers as one of the reasons for failure of the watchdog scheme [13].

The impact of the tactical network disturbance at the scenario is assessed through the K.I.A. rate and the mission-accomplished rate. When CMIDS was used, the K.I.A. rate was 8.3% and the mission-accomplished rate was 100%. In contrast, using SNORT or SURICATA, the K.I.A. rate was much higher, 46.6%, and the mission-accomplished rate was 84%. The rate of lost human lives increased to almost half of the troop and around 16% of missions was not successfully completed. Lastly, Watchdog proved to be the less suitable of the tested intrusion detection systems at this stage. When using it, the K.I.A. rate was 67.3% and the mission-accomplished rate was barely 68%.

These results clearly suggest that CMIDS provided a substantial increase in the effectiveness of the Blue forces, reducing the number of lives lost in combat roughly by a 5 time factor when compared to Snort and Suricata. Furthermore, CMIDS also allowed the Blue Forces to successfully complete all the missions while the other IDS did not.

5.3 Overall Performance

After all simulation runs, CMIDS was able to detect 91.75% of the passive attacks, missing just 33 attacks for every 400 passive attacks performed. Figure 5.1 represents the timeline of the attacks. It shows a passive attack been performed and followed by an active attack 35 seconds latter. It also shows the moment that any of the attacks were detected by CMIDS, Snort or Suricata (according to the mean time to detect an attack), and then prevented (blocked, according to the realistic assumption of an external entity described at Chapter

4). The results show that CMIDS prevented all active attacks at scenarios using the military radio configuration and was able to reduce the duration of these attacks at scenarios using civilian radio configuration.

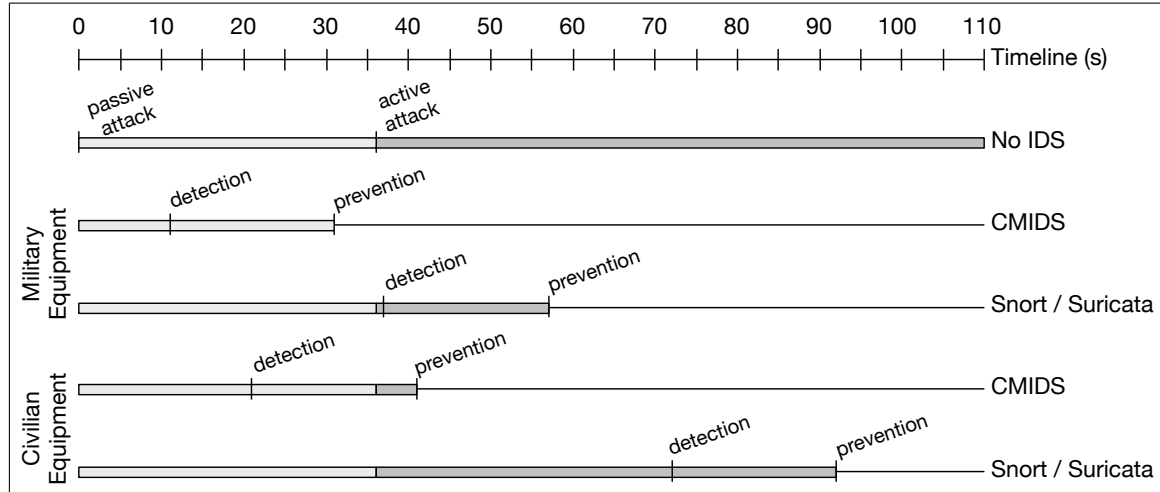


Figure 5.1: Attack Detection Timeline.

In summary, all the metrics and simulation results suggest that CMIDS is a substantially improved IDS for MANETs capable of (i) detecting passive adversaries, (ii) reducing the impact of the attack on the network traffic, (iii) reducing the capability of an adversary to harm the mission, and (iv) reducing the likelihood of loss of human lives at scenarios with attrition.

Chapter 6: Conclusions

This research presented a novel approach for addressing the problem of early identification of passive attacks against mobile networks. It proposes a new, non-intrusive collaborative intrusion detection system framework supporting legacy networks as well as existing IDSs. The framework, an implementation of the system, and its experimental evaluation are described in detail in the previous chapters of this Dissertation, while the associated source code is provided in the appendices.

The proposed Collaborative MANET Intrusion Detection System (CMIDS) uses multilateration and predictive algorithms to identify both passive and active attacks. The peculiarities of mobile *ad hoc* networks are considered and built in the CMIDS design. The CMIDS implementation was coded in C language and may be compiled for several platforms.

The impact of CMIDS to current operations depending on MANETs is systematically analyzed in a typical tactical scenario. A key aspect of the experimental design adopted in this evaluation was that, contrary to most data network evaluation experiments, it aimed for assessing not only the data network characteristics but also their impact on a tactical mission relying on it. This approach provided a more meaningful assessment of the advantages of using the system within the environment it is supposed to operate.

In order to enable such evaluation, a reusable testbed was created to replicate the aspects of the tactical scenario relevant to the operation of an IDS. The testbed is composed by three distinct simulation tools integrated through an interface designed and built as part of the research work.

The developed simulation model is flexible enough to support the evaluation of several other existing intrusion detection systems, provided they have software versions for Linux or UNIX operating systems.

The evaluation results obtained when comparing CMIDS with baseline IDS systems, suggest a significant improvement in the effectiveness of the forces using it, with a special emphasis on the reduced loss of life rate during combat. During the experiments conducted in this research, CMIDS was the only intrusion detection system allowing all missions to be successfully completed in every simulated scenario.

Although a military scenario was used to evaluate the new intrusion detection system, CMIDS may be used at civilian scenarios like natural disaster response and epidemic disease control. In common, these military and civilian scenarios use mobile *ad hoc* network to support their operations and CMIDS performs well using both military and commercial over-the-shelf radio equipment.

In summary, the work presented in this Dissertation extends the state of the art of intrusion detection systems used to support tactical missions using mobile ad hoc networks by designing and implementing a new IDS capable of detecting passive attacks, by developing new detection algorithms, and by testing and evaluating this new IDS and its impact on the mission it is supposed to support. In the process, I also extended a taxonomy to classify intrusion detection systems, and created a reusable simulation testbed through the integration of three distinct simulation tools.

The results of my research enhance the capability of MANETs of being applicable in many domains in which actions of an intruder can significantly impact the mission. Examples include various domains of application, such as law enforcement users (e.g. police forces, specialized teams like SWAT, and others), civil defense agencies in response to natural disaster or terrorism attack, military units fighting opposing forces, and other use cases where a regular wireless network would not be feasible to use. In all these cases, the results of my research support the statement that the CMIDS improves the communication, reduces the capability of an intruder to impact the mission, and does not impact the performance of the network. In some of these use cases, the use of CMIDS would reduce the likelihood of loss of human lives.

Due to the time constraints and the inherently tight focus of a PhD research, some

important aspects that are directly related to CMIDS had to be left for future work. Among these aspects, I would like to further investigate and design an improved synchronization mechanism between the simulation tools used in this work. This would provide further evidence confirming the substantial improvement in effectiveness of MANETS, which were observed via the performance metrics of CMIDS attained in the experiments so far. Another area of great interest for further investigation is the assessment of the impact of CMIDS in actual missions relying on MANETs. For instance, this can be achieved via carefully controlled field testing during military training operations, something that requires much more resources than those available in academia. Finally, I would also like to investigate alternatives for improving the process of collecting and analyzing performance metrics when using the simulation testbed. This would greatly benefit the MANET community by making the IDS evaluation more automated, less tedious and prone to errors.

Appendix A: Key Establishment Protocol (Scyther Source Code)

```
1  /*
2   * New key establishment after
3   * new neighbor discovery.
4   */
5
6  // Hash functions
7  hashfunction g1,g2;
8
9  //Simulating  $g^{ab} = g^{ba}$ 
10 protocol @exponentiation(RA)
11 {
12   role RA
13   {
14     var alpha,beta, T1,T2: Ticket;
15
16     recv_!1(RA,RA, g2(g1(T1),T2) );
17     send_!2(RA,RA, g2(g1(T2),T1) );
18   }
19 }
20
21
22 // The protocol description
23
24 const idI: Nonce; //Initiator Node ID
25 const idR: Nonce; //Receiver Node ID
26 const idB: Ticket; //Banished Node ID
27
```

```

28 protocol newNeighbor(I,R)
29 {
30   role I
31   {
32     //banished node verification
33
34     fresh ni: Nonce;
35     var nr: Nonce;
36     send_1(I,R, {I,ni,idI}pk(R) );
37     recv_2(R,I, {ni,nr,R,idR}pk(I) );
38     not match(idR,idB);
39     send_3(I,R, {nr,idI}pk(R) );
40
41     //key establishment
42
43     fresh s: Nonce;
44     fresh x: Nonce;
45     var beta: Ticket;
46     claim(I, SID, s);
47     send_4(I,R, I,s,g1(x) );
48     recv_5(R,I, R,s,beta, { R,s,beta,g1(x),I }sk(R) );
49     send_6(I,R, I,s, { I,s,g1(x),beta,R }sk(I) );
50
51     //claims
52
53     claim_i1(I,Secret,ni);
54     claim_i2(I,Secret,nr);
55     claim_i3(I,Niagree);
56     claim_i4(I,Nisynch);
57     claim_i5(I,SKR, g2(beta,x) );
58   }

```

```

59
60  role R
61  {
62    //banished node verification
63
64    var ni: Nonce;
65    fresh nr: Nonce;
66    recv_1(I,R, {I,ni,idI}pk(R) );
67    send_2(R,I, {ni,nr,R,idR}pk(I) );
68    recv_3(I,R, {nr,idI}pk(R) );
69    not match(idI,idB);
70
71    //key establishment
72
73    fresh y: Nonce;
74    var s: Nonce;
75    var alpha: Ticket;
76    recv_4(I,R, I,s,alpha );
77    claim(R, SID, s);
78    send_5(R,I, R,s,g1(y), { R,s,g1(y),alpha,I }sk(R) );
79    recv_6(I,R, I,s, { I,s,alpha,g1(y),R }sk(I) );
80
81    //claims
82    claim_r1(R, Secret ,ni);
83    claim_r2(R, Secret ,nr);
84    claim_r3(R, Niagree);
85    claim_r4(R, Nisynch);
86    claim_r5(R,SKR, g2(alpha,y) );
87  }
88 }

```

Appendix B: MÄK VR-Forces Background Script

The VR-Forces background scripted task is used to collect, send and receive data to and from CORE application.

File <date.bat>

```
1 @echo off
2 For /f "tokens=2-4 delims=/ " %%a in ('date /t') do (set mydate=%%c-%%a-%%b)
3 For /f "tokens=1-2 delims=/" %%a in ("%TIME%") do (set mytime=%%a%%b)
4 echo %mydate%-_%mytime% > simRun.txt
```

File <vrfTask.lua>

```
1 — This script is used to interface VR-Forces and CORE
2
3 require "vrfutil"
4
5 ioControl = -1
6 tickTime = 0
7 tickPeriod = 5.0
8 currentName = this.getName() —the entity name is supposed to be integer
   numbers
9 firstRun = 0
10 simRun = "ola2"
11 serverIP = "172.16.0.1"
12 serverPort = "38001"
13
14
15 — Called when the task first starts. Never called again.
```

```

16 function init()
17     — Set the tick period for this script.
18     vrf:setTickPeriod(tickPeriod)
19
20 end
21
22
23 — Called each tick while this task is active.
24 function tick()
25     if firstRun == 0 then
26
27         os.execute("..\..\JeroLogs\date.bat > ..\..\JeroLogs\runSim-
                ..currentName..".txt")
28
29         — read date
30         arquivo, msg, cod = io.open("..\..\JeroLogs\runSim-
                ..currentName..".txt"
                , "r")
31         if arquivo == nil then
32             print("Log file openning error: "..msg)
33         else
34             simRun = arquivo:read("*line")
35             arquivo:close()
36         end
37
38         — crete directory to save simulation files
39         os.execute("mkdir ..\..\JeroLogs\"..simRun)
40
41         arquivo, msg, cod = io.open("..\..\JeroLogs\"..simRun.."\\Log-
                ..currentName..".txt", "w")
42         if arquivo == nil then
43             print("Log file openning error: "..msg)

```

```

44     else
45         arquivo:close()
46         ioControl = 1
47     end
48     firstRun =1
49 end
50     currentLocation = this:getLocation3D()
51     currentLat = currentLocation:getLat()
52     currentLon = currentLocation:getLon()
53     currentAlt = currentLocation:getAlt()
54     currentLat = (currentLat * 180)/3.14159265
55     currentLon = (currentLon * 180)/3.14159265
56     tickTime = tickTime + 5
57
58     —some GUI adjustments
59     adjustedLat = (currentLat - 0.55059889695360) / 0.00000015696123
60     adjustedLon = (currentLon - 1.1484430190007) / 0.00000018418111
61     adjustedTime = tickTime
62     if ioControl > 0 then
63         arquivo, msg, cod = io.open("..\..\JeroLogs\\"..simRun.."\\Log—"
            ..currentName.." .txt", "a")
64         if arquivo == nil then
65             print("Log file openning error: "..msg)
66             ioControl = 1
67         else
68             os.execute("c:\\vrfMsg\\vrfEventLocationSender.exe "..serverIP.."
                "..serverPort.." "..currentName.." "..currentLat.." "
                ..currentLon.." "..currentAlt)
69         arquivo:write(currentName.."\\n")
70         arquivo:write(adjustedTime.."\\n")
71         arquivo:write(adjustedLat.."\\n")

```

```

72     arquivo:write(adjustedLon.."\\n")
73     arquivo:close()
74     ioControl = 2
75     end
76     else
77         print("Log file init error!")
78     end
79 end
80
81
82 function saveState()
83 end
84
85
86 function loadState()
87 end
88
89
90 function shutdown()
91     if ioControl > 0 then
92         arquivo, msg, cod = io.open("../..\\JeroLogs\\"..simRun.."\\Log-"
            ..currentName..".txt", "r")
93         if arquivo == nil then
94             print("Log file openning error: "..msg)
95         else
96             —
97             arquivo2, msg2, cod2 = io.open("../..\\JeroLogs\\"..simRun.."\\Mob-"
                ..currentName..".txt", "w")
98             if arquivo2 == nil then
99                 print("Log file openning error: "..msg2)
100            else

```



```

101     arquivo2:seek('set')
102     mobName = arquivo:read("*line")
103
104     mobTimeActual = arquivo:read("*line")
105     mobLatActual = arquivo:read("*line")
106     mobLonActual = arquivo:read("*line")
107
108     mobName = arquivo:read("*line")
109     mobTimeNext = arquivo:read("*line")
110     mobLatNext = arquivo:read("*line")
111     mobLonNext = arquivo:read("*line")
112
113     mobSpeed = (math.sqrt( (mobLatNext - mobLatActual) * (mobLatNext -
        mobLatActual) + (mobLonNext - mobLonActual) * (mobLonNext -
        mobLonActual) ) ) / tickPeriod
114
115     arquivo2:write("$node_("..mobName..) set X_ "..mobLonActual.."\n")
116     arquivo2:write("$node_("..mobName..) set Y_ "..mobLatActual.."\n")
117     arquivo2:write("$node_("..mobName..) set Z_ 0.00\n")
118     arquivo2:write("$ns_ at "..mobTimeActual.. \" $node("..mobName..) setdest
        "..mobLonNext..\" \"..mobLatNext..\" \"..mobSpeed..\"\\n")
119
120     checkEOF = arquivo:read("*line")
121     while checkEOF do
122         mobTimeActual = mobTimeNext
123         mobLatActual = mobLatNext
124         mobLonActual = mobLonNext
125
126         mobName = checkEOF
127         mobTimeNext = arquivo:read("*line")
128         mobLatNext = arquivo:read("*line")

```

```

129     mobLonNext = arquivo:read("*line")
130     checkEOF = arquivo:read("*line")
131
132     mobSpeed = (math.sqrt( (mobLatNext - mobLatActual) * (mobLatNext -
        mobLatActual) + (mobLonNext - mobLonActual) * (mobLonNext -
        mobLonActual) ) ) / tickPeriod
133
134     arquivo2:write("$ns_ at "..mobTimeActual.." \"$node("..mobName..")
        setdest "..mobLonNext.." "..mobLatNext.." "..mobSpeed.."\"\\n")
135     end
136     arquivo2:close()
137     end
138     arquivo:close()
139     end
140
141     if this:isDestroyed() then
142         os.execute("echo "..currentName.." KIA! > ..\\..\\JeroLogs\\"..simRun.."\\
            KIA-"..currentName..".txt")
143     else
144         os.execute("echo "..currentName.." Alive! > ..\\..\\JeroLogs\\"..simRun.."
            "\\Alive-"..currentName..".txt")
145     end
146
147     else
148         print("Log file init error!")
149     end
150 end

```

Appendix C: VR-Forces/CORE Interface Source Code

The interface between VR-Forces and CORE is composed by two modules responsible to send and receive the exchanged data. They were developed using Qt environment and each module contains four source files.

The *Send Data Module* is composed by the source files `vrfEventLocationSender.pro`, `main.cpp`, `myudpclient.h`, and `myudpclient.cpp`.

The *Receive Data Module* is composed by the source files `vrfEventLocationReceiver.pro`, `main.cpp`, `myudpclient.h`, and `myudpclient.cpp`.

Send Data Module - File <`vrfEventLocationSender.pro`>

```
1  QT += core
2  QT -= gui
3  QT += network
4
5  CONFIG += c++11
6
7  TARGET = vrfEventLocationSender
8  CONFIG += console
9  CONFIG -= app_bundle
10
11 TEMPLATE = app
12
13 SOURCES += main.cpp \
14         myudpclient.cpp
15
16 HEADERS += \
17         myudpclient.h
```

Send Data Module - File <main.cpp>

```
1  #include <QCoreApplication>
2  #include <QStringList>
3
4  #include <iostream>
5
6  #include "myudpclient.h"
7
8  using namespace std;
9
10 int main(int argc, char *argv[])
11 {
12     if (argc != 7) {
13         cout << "Use: dest_ip  port_number nem lat long alt" << endl;
14         exit(1);
15     }
16
17     QCoreApplication a(argc, argv);
18     a.setApplicationName("Score Sender");
19     QStringList args = a.arguments();
20     QString serverName(args[1]);
21     int serverPort = args[2].toInt();
22     QString nem(args[3]);
23     QString latitude(args[4]);
24     QString longitude(args[5]);
25     QString altitude(args[6]);
26     QString msg(nem);
27     msg.append("\n");
28     msg.append(latitude);
29     msg.append("\n");
30     msg.append(longitude);
```

```
31     msg.append("\n");
32     msg.append(altitude);
33
34     MyUDP client;
35     //client.startServer(serverName, serverPort);
36     client.sendUDP(serverName, serverPort, msg);
37
38     return 0; //a.exec();
39 }
```

Send Data Module - File <myudpclient.h>

```
1  #ifndef MYUDPCCLIENT.H
2  #define MYUDPCCLIENT.H
3
4  #include <QObject>
5  #include <QUdpSocket>
6
7
8  class MyUDP : public QObject
9  {
10      Q_OBJECT
11  public:
12      explicit MyUDP(QObject *parent = 0);
13      void sendUDP(QString, int, QString);
14      void startServer(QString, int);
15      void stopServer();
16
17  signals:
18
19  public slots:
20      void readyRead();
21
22  private:
23      QUdpSocket *socket;
24
25
26  };
27
28 #endif // MYUDPCCLIENT.H
```

Send Data Module - File <myudpclient.cpp>

```
1  #include "myudpclient.h"
2
3  MyUDP::MyUDP(QObject *parent) : QObject(parent)
4  {
5      socket = new QUdpSocket(this);
6  }
7
8  void MyUDP::startServer(QString serverName, int port){
9      QHostAddress sender;
10     //sender.setAddress(serverName);
11     sender.setAddress(QHostAddress::LocalHost);
12     socket->bind(sender, port);
13     connect(socket, SIGNAL(readyRead()), this, SLOT(readyRead()));
14 }
15
16 void MyUDP::stopServer(){
17     socket->close();
18 }
19
20
21 void MyUDP::sendUDP(QString serverName, int port, QString msg)
22 {
23     QByteArray Data;
24     Data.append(msg);
25
26     QHostAddress sender;
27     sender.setAddress(serverName);
28
29     socket->writeDatagram(Data, sender, port);
30     //socket->writeDatagram(Data, QHostAddress::LocalHost, port);
```

```

31     }
32
33     void MyUDP::readyRead()
34     {
35         QByteArray buffer;
36         buffer.resize(socket->pendingDatagramSize());
37
38         QHostAddress sender;
39         quint16 senderPort;
40
41         socket->readDatagram(buffer.data(), buffer.size(),
42                             &sender, &senderPort);
43
44         qDebug() << "Message from: " << sender.toString();
45         qDebug() << "Message port: " << senderPort;
46         qDebug() << "Message: " << buffer;
47     }

```


Receive Data Module - File <vrfLocRec-proj.pro>

```
1 QT += core
2 QT -= gui
3 QT += network
4
5 CONFIG += c++11
6
7 TARGET = vrfEventLocationSender
8 CONFIG += console
9 CONFIG -= app_bundle
10
11 TEMPLATE = app
12
13 SOURCES += main.cpp \
14     myudpclient.cpp
15
16 HEADERS += \
17     myudpclient.h
```

Receive Data Module - File <vrfLocRec-main.cpp>

```
1  #include <QCoreApplication>
2  #include <QStringList>
3
4  #include <iostream>
5
6  #include "myudpclient.h"
7
8  using namespace std;
9
10 int main(int argc, char *argv[])
11 {
12     if (argc != 7) {
13         cout << "Use: dest_ip port_number nem lat long alt" << endl;
14         exit(1);
15     }
16
17     QCoreApplication a(argc, argv);
18     a.setApplicationName("Score Sender");
19     QStringList args = a.arguments();
20     QString serverName(args[1]);
21     int serverPort = args[2].toInt();
22     QString nem(args[3]);
23     QString latitude(args[4]);
24     QString longitude(args[5]);
25     QString altitude(args[6]);
26     QString msg(nem);
27     msg.append("\n");
28     msg.append(latitude);
29     msg.append("\n");
30     msg.append(longitude);
```

```
31     msg.append("\n");
32     msg.append(altitude);
33
34     MyUDP client;
35     //client.startServer(serverName, serverPort);
36     client.sendUDP(serverName, serverPort, msg);
37
38     return 0; //a.exec();
39 }
```

Receive Data Module - File <vrfLocRec-myudpclient.h>

```
1  #ifndef MYUDPCCLIENT.H
2  #define MYUDPCCLIENT.H
3
4  #include <QObject>
5  #include <QUdpSocket>
6
7
8  class MyUDP : public QObject
9  {
10      Q_OBJECT
11  public:
12      explicit MyUDP(QObject *parent = 0);
13      void sendUDP(QString, int, QString);
14      void startServer(QString, int);
15      void stopServer();
16
17  signals:
18
19  public slots:
20      void readyRead();
21
22  private:
23      QUdpSocket *socket;
24
25
26  };
27
28 #endif // MYUDPCCLIENT.H
```

Receive Data Module - File <vrfLocRec-myudpclient.cpp>

```
1  #include "myudpclient.h"
2
3  MyUDP::MyUDP(QObject *parent) : QObject(parent)
4  {
5      socket = new QUdpSocket(this);
6  }
7
8  void MyUDP::startServer(QString serverName, int port){
9      QHostAddress sender;
10     //sender.setAddress(serverName);
11     sender.setAddress(QHostAddress::LocalHost);
12     socket->bind(sender, port);
13     connect(socket, SIGNAL(readyRead()), this, SLOT(readyRead()));
14 }
15
16 void MyUDP::stopServer(){
17     socket->close();
18 }
19
20
21 void MyUDP::sendUDP(QString serverName, int port, QString msg)
22 {
23     QByteArray Data;
24     Data.append(msg);
25
26     QHostAddress sender;
27     sender.setAddress(serverName);
28
29     socket->writeDatagram(Data, sender, port);
30     //socket->writeDatagram(Data, QHostAddress::LocalHost, port);
```

```

31     }
32
33     void MyUDP::readyRead()
34     {
35         QByteArray buffer;
36         buffer.resize(socket->pendingDatagramSize());
37
38         QHostAddress sender;
39         quint16 senderPort;
40
41         socket->readDatagram(buffer.data(), buffer.size(),
42                             &sender, &senderPort);
43
44         qDebug() << "Message from: " << sender.toString();
45         qDebug() << "Message port: " << senderPort;
46         qDebug() << "Message: " << buffer;
47     }

```

Appendix D: CMIDS Source Code

File <cmids.c>

```
1  /* CMIDS */
2  #include <sys/types.h>
3  #include <sys/socket.h>
4  #include <netinet/in.h>
5  #include <netdb.h>
6  #include <stdio.h>
7  #include <arpa/inet.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <time.h>
11 #include <sys/ipc.h>
12 #include <sys/shm.h>
13 #include <math.h>
14
15 #define TAMAX 80 //Broadcast buffer size
16 #define NOMSG 2000 // # collected msg
17 #define TICKS 240 // # ticks (1 per 5sec) in 20 minutes of simulation
18 #define NONOD 20 // # of simulated nodes
19 #define TIMEPREC 0.001 //time precision
20 #define LATPREC 1.000000 //latitude precision (1 meter)
21 #define LONPREC 1.000000 //longitude precision (1 meter)
22 #define CONFIRMED 3 // # of necessary confirmations to issue an alert
23
24 void error(char *msg){
25     perror(msg);
26     exit(0);
27 }
```

```

28
29  int main(int argc, char *argv[]) {
30      int sock, length, fromlen, n;
31      struct sockaddr_in server;
32      struct sockaddr_in from;
33      struct hostent *hp;
34      int broadcastEnable = 1;
35
36      char buf[TAMAX];
37      char msg[TAMAX];
38
39      unsigned long pkgSent=0;
40      unsigned long bytesSent=0;
41
42      FILE *file;
43      FILE *fileLoc;
44      FILE *fileNode;
45      FILE *fileAlert;
46      FILE *fileMission;
47      FILE *fileAlertLogN;
48      FILE *fileAlertLogT;
49      char fileName[80];
50      char fileNameLoc[80];
51      char fileNameNode[80];
52      char fileNameAlert[80];
53      char fileNameMission[80];
54      char fileNameAlertLogN[80];
55      char fileNameAlertLogT[80];
56      float tempoSimulacao;
57      time_t inicio, fim, fimReal, parcial;
58

```



```

59  char id[20];
60  char mStatus[20]; //1-> original message; 2-> copy message
61  char maxSpeed[20];
62  char lineTime[20];
63  char lineLat[20];
64  char lineLon[20];
65  char hostAddr[16];
66  int j=0;
67  int k=0;
68  int l=0;
69  int m=0;
70  int lAux = 0;
71  int mAux = 0;
72  int firstIntrusion = 0;
73  char charPosition;
74  double lTime, lLat, lLon;
75  int seqN=1;
76  struct pCollected {
77      int id;
78      int mStatus;
79      int seqN;
80      double lTime;
81      double lLat;
82      double lLon;
83      char hostAddr[16];
84  } bCol[NOMSG];
85  char type[20];
86  char hostAddrAux[20];
87  char thisHostAddr[20];
88  int currentNode=-1;
89  int pListFlag = 0; //1 -> pKnow[]; 2-> pNew[]

```

```

90  int nConfirmation = 0;
91  int alertCounter = 0;
92
93  //shared memory variables
94  char c;
95  int shmid;
96  key_t key;
97  struct pCollected *shm, *s;
98
99  //mission knowledge variables
100 struct mKnowledge{
101     int id;
102     double maxSpeed;
103 };
104
105 //position knowledge variables
106 struct pTime{
107     int nConfirmation; ///# of position confirmations
108     int alert;
109     double lTime;
110     double lLat;
111     double lLon;
112 };
113 struct pKnowledge{
114     int id;
115     int type;
116     char hostAddr[16];
117     double maxSpeed;
118     struct pTime position[TICKS];
119 };
120

```

```

121  struct pKnowledge pKnow[NONOD];
122  struct pKnowledge pNew[NONOD];
123  struct mKnowledge mKnow[NONOD];
124
125  struct nNodes {
126      int type; // 1-> Tactical; 2-> Sensor; 3-> Hybrid
127      int id;
128      char hostAddr[20];
129  } node[NONOD];
130
131  //number arguments test
132  if (argc < 7) {
133      fprintf(stderr, "Use positionKnowledgeFileName nodesFileName port key nodeID
          missionKnowledgeFileName\n");
134      exit(0);
135  }
136
137
138  strcpy(fileNameLoc, argv[1]);
139  strcpy(fileNameNode, argv[2]);
140  strcpy(fileNameMission, argv[6]);
141  key = ftok("/home/jeronymo/Downloads/virtualbox linux files/ids/aa.txt", atoi
          (argv[4]));
142  if (0 > key){
143      perror("ftok");
144  }
145  else{
146      //printf("ftok success: %lli\n", (long long int) key);
147  }
148  strcpy(fileName, "saidaIDS-");
149  strcat(fileName, argv[4]);

```

```

150
151 strcpy(fileNameAlert, "saidaAlert-");
152 strcat(fileNameAlert, argv[4]);
153
154 strcpy(fileNameAlertLogN, "saidaAlertN-");
155 strcat(fileNameAlertLogN, argv[4]);
156 strcat(fileNameAlertLogN, ".txt");
157
158 strcpy(fileNameAlertLogT, "saidaAlertT-");
159 strcat(fileNameAlertLogT, argv[4]);
160 strcat(fileNameAlertLogT, ".txt");
161
162 //initialize socket
163 sock= socket(AF_INET, SOCK_DGRAM, 0);
164 if (sock < 0) error("socket");
165
166 //Enables broadcast
167 int ret = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcastEnable, sizeof
    (broadcastEnable));
168
169 server.sin_family = AF_INET;
170
171 //initializes node list
172 for(l=0;l<NONOD;l++){
173     node[l].type = 0;
174     node[l].id = 0;
175     bzero(node[l].hostAddr,20);
176 }
177 bzero(hostAddrAux,20);
178 bzero(thisHostAddr,20);
179

```

```

180  //Opens node file
181  if( (fileNode = fopen(fileNameNode,"r")) == NULL ){
182      printf("Problem reading file: %s\n", fileNameNode);
183      return 1;
184  }
185
186  //reads node file and copy its content
187  l=0;
188  fgets(type, 19, fileNode);
189  while ( (!feof(fileNode)) && (l <NONOD) && (atoi(type)!=0) ) {
190      m=0;
191      node[l].type = atoi(type);
192      fgets(id, 19, fileNode);
193      node[l].id = atoi(id);
194      fgets(node[l].hostAddr, 19, fileNode);
195      while((node[l].hostAddr[m] != '\n')&&(m<20)){
196          m++;
197      }
198      node[l].hostAddr[m-1]='\0';
199      if( (atoi(argv[5]) == node[l].id) ) {
200          strcpy(thisHostAddr, node[l].hostAddr);
201      }
202      fgets(type, 19, fileNode);
203      l++;
204  }
205  fclose(fileNode);
206
207  //shared memory initialization
208  if ((shmids = shmget(key, NOMSG*(sizeof(struct pCollected)), 0666)) < 0) {
209      perror("shmget");
210      printf("ids - shmget\n");

```

```

211     exit(1);
212 }
213 if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
214     perror("shmat");
215     exit(1);
216 }
217 s = shm;
218
219 //opens file with locations
220 if( (fileLoc = fopen(fileNameLoc,"rb")) == NULL ){
221     printf("Problem reading file: %s\n", fileNameLoc);
222     return 1;
223 }
224 rewind(fileLoc);
225
226 //initializes position knowledge list
227 for(l=0;l<NONOD;l++){
228     pKnow[l].id = 0; //also used as loop condition
229     pKnow[l].type = 0;
230     pKnow[l].maxSpeed = 0.0;
231     bzero(pKnow[l].hostAddr,16);
232     for(m=0;m<TICKS;m++){
233         pKnow[l].position[m].nConfirmation = -1; //also used as loop condition
234         pKnow[l].position[m].alert = 0;
235         pKnow[l].position[m].lTime = 0.0;
236         pKnow[l].position[m].lLat = 0.0;
237         pKnow[l].position[m].lLon = 0.0;
238     }
239 }
240
241 //uploads position knowledge

```

```

242  while ( !feof(fileLoc) ){
243      l=0;
244      m=0;
245      bzero(id , 20);
246      bzero(lineTime , 20);
247      bzero(lineLat , 20);
248      bzero(lineLon , 20);
249      fgets(id , 19, fileLoc);
250      fgets(lineTime , 19, fileLoc);
251      fgets(lineLat , 19, fileLoc);
252      fgets(lineLon , 19, fileLoc);
253
254      while( (pKnow[l].id != 0) && (pKnow[l].id != atoi(id)) && (l<NONOD) ){
255          l++;
256      }
257
258      if(pKnow[l].id == atoi(id)){
259          //loop m and save info
260          while(pKnow[l].position[m].nConfirmation != -1){
261              m++;
262          }
263          pKnow[l].position[m].nConfirmation = 0;
264          pKnow[l].position[m].lTime = atof(lineTime);
265          pKnow[l].position[m].lLat = atof(lineLat);
266          pKnow[l].position[m].lLon = atof(lineLon);
267      }
268      else if(pKnow[l].id == 0){
269          //save info
270          pKnow[l].id = atoi(id);
271          pKnow[l].position[0].nConfirmation = 0;
272          pKnow[l].position[0].lTime = atof(lineTime);

```

```

273     pKnow[1].position[0].lLat = atof(lineLat);
274     pKnow[1].position[0].lLon = atof(lineLon); //node[l].id node[l].hostAddr
275     //identifies the node type at the message
276     lAux = 0;
277     while( (node[lAux].type != 0) && (lAux<NONOD) ){
278         if(node[lAux].id == pKnow[1].id){
279             strcpy(pKnow[1].hostAddr,node[lAux].hostAddr);
280             pKnow[1].type = node[lAux].type;
281             lAux=NONOD-1;
282         }
283         lAux++;
284     }
285     //
286 }
287
288 }
289
290 fclose(fileLoc);
291
292 //opens file with mission knowledge
293 if( (fileMission = fopen(fileNameMission,"rb")) == NULL ){
294     printf("Problem reading file: %s\n", fileNameMission);
295     return 1;
296 }
297 rewind(fileMission);
298
299 //initializes mission knowledge list
300 for(l=0;l<NONOD;l++){
301     mKnow[1].id = 0; //also used as loop condition
302     mKnow[1].maxSpeed = 0.0;
303 }

```



```

304
305  //uploads mission knowledge list
306  while ( !feof(fileMission) ){
307      l=0;
308      m=0;
309      bzero(id , 20);
310      bzero(maxSpeed , 20);
311      fgets(id , 19, fileMission);
312      fgets(maxSpeed , 19, fileMission);
313
314      while( (mKnow[l].id != 0) && (mKnow[l].id != atoi(id)) && (l<NONOD) ){
315          l++;
316      }
317
318      if(mKnow[l].id == atoi(id)){
319          //nothing to be done
320      }
321      else if(mKnow[l].id == 0){
322          //save info
323          mKnow[l].id = atoi(id);
324          mKnow[l].maxSpeed = atof(maxSpeed);
325      }
326
327  }
328
329  fclose(fileMission);
330
331  //updates position knowledge list with mission knowledge
332  l=0;
333  m=0;
334  while( (pKnow[l].id != 0) && (l<NONOD) ){

```

```

335   while( (mKnow[m].id != pKnow[l].id) && (m<NONOD) ){ //variable m is used in
        a different way here
336       m++;
337   }
338   if (mKnow[m].id == pKnow[l].id){
339       pKnow[l].maxSpeed = mKnow[m].maxSpeed;
340   }
341   l++;
342   m=0;
343 }
344
345 //tests read position knowledge list
346 l=0;
347 m=0;
348 while( (pKnow[l].id != 0) && (l<NONOD) ){
349     while( (pKnow[l].position[m].nConfirmation != -1) && (m<TICKS) ){
350         m++;
351     }
352     l++;
353     m=0;
354 }
355
356 //initializes new position list
357 for(l=0;l<NONOD;l++){
358     pNew[l].id = 0; //also used as loop condition
359     pNew[l].maxSpeed = 0.0;
360     bzero(pNew[l].hostAddr,16);
361     for(m=0;m<TICKS;m++){
362         pNew[l].position[m].nConfirmation = -1; //also used as loop condition
363         pNew[l].position[m].alert = 0;
364         pNew[l].position[m].lTime = 0.0;

```

```

365     pNew[l].position[m].lLat = 0.0;
366     pNew[l].position[m].lLon = 0.0;
367 }
368 }
369
370 //opens file to save broadcasted positions
371 if( ( file = fopen(fileName,"wb")) == NULL ){
372     printf("Problem writting to file: %s\n", fileName);
373     return 1;
374 }
375 rewind( file );
376
377 //opens file to save intrusion detections
378 if( ( fileAlert = fopen(fileNameAlert,"wb")) == NULL ){
379     printf("Problem writting to file: %s\n", fileNameAlert);
380     return 1;
381 }
382 rewind( fileAlert );
383
384 //opens file to save number of intrusion detections
385 if( ( fileAlertLogN = fopen(fileNameAlertLogN,"wb")) == NULL ){
386     printf("Problem writting to file: %s\n", fileNameAlertLogN);
387     return 1;
388 }
389 rewind( fileAlertLogN );
390
391 //opens file to save time of first intrusion detection
392 if( ( fileAlertLogT = fopen(fileNameAlertLogT,"wb")) == NULL ){
393     printf("Problem writting to file: %s\n", fileNameAlertLogT);
394     return 1;
395 }

```

```

396  rewind(fileAlertLogT);
397
398  //main loop to read data from shared memory
399  unsigned long i=1;
400  while (s[i].seqN != -1) {
401      if(i==1){
402          inicio = time(NULL);
403      }
404
405      //tests if there is a new broadcasted position
406      if(s[i].seqN > 0){
407
408          //writes data to file
409          bzero(buf,TAMAX);
410          sprintf(buf, "%i\n",s[i].seqN);
411          fwrite(buf,strlen(buf),1,file);
412          bzero(buf,TAMAX);
413          sprintf(buf, "%i\n",s[i].id);
414          fwrite(buf,strlen(buf),1,file);
415          bzero(buf,TAMAX);
416          sprintf(buf, "%i\n",s[i].mStatus);
417          fwrite(buf,strlen(buf),1,file);
418          bzero(buf,TAMAX);
419          sprintf(buf, "%s\n",s[i].hostAddr);
420          fwrite(buf,strlen(buf),1,file);
421          bzero(buf,TAMAX);
422          sprintf(buf, "%.11f\n",s[i].lTime);
423          fwrite(buf,strlen(buf),1,file);
424          bzero(buf,TAMAX);
425          sprintf(buf, "%.11f\n",s[i].lLat);
426          fwrite(buf,strlen(buf),1,file);

```

```

427     bzero( buf ,TAMAX);
428     sprintf( buf, "%11f\n", s[ i ].lLon);
429     fwrite( buf, strlen( buf ),1, file );
430
431     //process the message
432
433     //identifies the node type at the message
434     l=0;
435     while( ( node[ l ].type != 0) && (l<NONOD) ){
436         if( node[ l ].id == s[ i ].id){
437             currentNode = l;
438             l=NONOD-1;
439         }
440         l++;
441     }
442
443     /******
444     ATTENTION A1
445     if currentNode == -1 a new node has been identified
446     and it needs to be processed properly
447     *****/
448
449     //locates broadcasted position at pKnow list
450     l=0;
451     while( ( pKnow[ l ].id != node[ currentNode ].id) && (l<NONOD) ){
452         l++;
453     }
454     m=0;
455     while( ( pKnow[ l ].position[ m ].nConfirmation != -1) && (m<TICKS) ){
456         if( ( s[ i ].lTime > pKnow[ l ].position[ m ].lTime - TIMEPREC) &&
457             ( s[ i ].lTime < pKnow[ l ].position[ m ].lTime + TIMEPREC) &&

```

```

458         (s[i].lLat > pKnow[l].position[m].lLat - LATPREC) &&
459         (s[i].lLat < pKnow[l].position[m].lLat + LATPREC) &&
460         (s[i].lLon > pKnow[l].position[m].lLon - LONPREC) &&
461         (s[i].lLon < pKnow[l].position[m].lLon + LONPREC) ){
462     ++pKnow[l].position[m].nConfirmation;
463     m=TICKS;
464     pListFlag = 1;
465 }
466 m++;
467 }
468
469 //if m == TICKS or pKnow[l].position[m].nConfirmation == -1
470 //a new position in time has been identified
471 //and it needs to be found or inserted in pNew list
472 if( (m==TICKS) || (pKnow[l].position[m].nConfirmation == -1) ){
473     l=0;
474     while( (pNew[l].id != 0) && (pNew[l].id != node[currentNode].id) && (l<
         NONOD) ){
475         l++;
476     }
477     m=0;
478     while( (pNew[l].position[m].nConfirmation != -1) && (m<TICKS) ){
479         //warning:
480         //be aware if PREC is large there may be more confirmations than sensor
         nodes
481         if( (s[i].lTime > pNew[l].position[m].lTime - TIMEPREC) &&
482             (s[i].lTime < pNew[l].position[m].lTime + TIMEPREC) &&
483             (s[i].lLat > pNew[l].position[m].lLat - LATPREC) &&
484             (s[i].lLat < pNew[l].position[m].lLat + LATPREC) &&
485             (s[i].lLon > pNew[l].position[m].lLon - LONPREC) &&
486             (s[i].lLon < pNew[l].position[m].lLon + LONPREC) ){

```

```

487     ++pNew[l].position[m].nConfirmation;
488     m=TICKS;
489     pListFlag = 2;
490 }
491 m++;
492 }
493 if( m==TICKS ){
494     printf("New position of buffer overflow.\n");
495     exit(1);
496 }
497 else if(pNew[l].position[m].nConfirmation == -1){
498     pNew[l].id = s[i].id;
499     strcpy(pNew[l].hostAddr,s[i].hostAddr); //hostAddr is the sender's IP and
        not node IP
500     pNew[l].position[m].nConfirmation = 1;
501     pNew[l].position[m].lTime = s[i].lTime;
502     pNew[l].position[m].lLat = s[i].lLat;
503     pNew[l].position[m].lLon = s[i].lLon;
504     pListFlag = 2;
505 }
506 }
507
508 //verifies if it comes from tactical node, if it is original, and sends it
    to sensor nodes
509 //if it comes from sensor or hybrid node there is nothing left to be done
510 //if( (currentNode > -1) && (node[currentNode].type != 2) && (s[i].mStatus
    == 1) ){
511 if( (currentNode > -1) && (node[currentNode].type == 1) && (s[i].mStatus ==
    1) ){
512     bzero(msg,TAMAX);
513     bzero(id, 20);

```

```

514     bzero(mStatus, 20);
515     bzero(lineTime, 20);
516     bzero(lineLat, 20);
517     bzero(lineLon, 20);
518     sprintf(id, "%d\n", s[i].id);
519     sprintf(mStatus, "2\n");
520     sprintf(lineTime, "%.11f\n", s[i].lTime);
521     sprintf(lineLat, "%.11f\n", s[i].lLat);
522     sprintf(lineLon, "%.11f\n", s[i].lLon);
523     strcpy(msg, id);
524     strcat(msg, mStatus);
525     strcat(msg, lineTime);
526     strcat(msg, lineLat);
527     strcat(msg, lineLon);
528     l=0;
529     while( (node[l].type != 0) && (l< NONOD) ){
530         //if test avoids a node to send a message to itself
531         if( !(strcmp(node[l].hostAddr, thisHostAddr)==0) ){
532             //if test allows msg to be sent to every sensor and hybrid node
533             if(node[l].type != 1){
534                 hp = gethostbyname(node[l].hostAddr);
535                 if (hp==0) error("Host nao encontrado");
536
537                 bcopy( (char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);
538                 server.sin_port = htons(atoi(argv[3]));
539                 length=sizeof(struct sockaddr_in);
540
541                 n=sendto(sock, msg, sizeof(msg), 0, &server, length);
542                 if(n>0){
543                     bytesSent += (unsigned long) n;
544                     pkgSent += 1;

```



```

545         }
546     }
547 }
548     l++;
549 }
550 }
551
552 //Intrusion detection verifications
553 //Check #01
554 //confirmed pNew means intrusion
555 //*
556 l=0;
557 while( (pNew[l].id != 0) && (l<NONOD) ){
558     m=0;
559     while( (pNew[l].position[m].nConfirmation != -1) && (m<TICKS) ){
560         if(pNew[l].position[m].alert == 0){
561             if(pNew[l].position[m].nConfirmation >= CONFIRMED ){
562                 pNew[l].position[m].alert = 1;
563                 fim = time(NULL);
564                 ++alertCounter;
565
566                 //issues an alert
567                 bzero(buf,TAMAX);
568                 sprintf(buf, "Time: \t%f\n",difftime(fim,inicio));
569                 fwrite(buf,strlen(buf),1,fileAlert);
570                 //saves first intrusion time
571                 if(firstIntrusion == 0){
572                     bzero(buf,TAMAX);
573                     sprintf(buf, "%f\n",difftime(fim,inicio));
574                     fwrite(buf,strlen(buf),1,fileAlertLogT);
575                     fclose(fileAlertLogT);

```

```

576         firstIntrusion = 1;
577     }
578     bzero(buf,TAMAX);
579     sprintf(buf, "ID: \t%i\n",pNew[l].id);
580     fwrite(buf,strlen(buf),1,fileAlert);
581     bzero(buf,TAMAX);
582     sprintf(buf, "Addr: \t%s\n\n",pNew[l].hostAddr);
583     fwrite(buf,strlen(buf),1,fileAlert);
584 }
585 }
586 m++;
587 }
588 l++;
589 }/**/
590 //end of Check #01
591
592
593 //Check #02
594 //knetic envelope
595 /**
596 int mLeft = -1; // not used if value = -1
597 int mRight = -1; // not used if value = -1
598 double speed = 0.0;
599 double auxSqrt = 0.0;
600 lAux = 0;
601 mAux = 0;
602 int mayBeEmbarked = 0;
603 float minTime = 0;
604 l=0;
605 parcial = time(NULL);
606 while( (pKnow[l].id != 0) && (l<NONOD) ){

```

```

607     m=0;
608     while( (pKnow[l].position[m].nConfirmation != -1) && (m<TICKS) ){
609         minTime = difftime(parcial,inicio) - pKnow[l].position[m].lTime;
610         if( ( (pKnow[l].type == 3) && (pKnow[l].position[m].nConfirmation >= 1)
611             ) || (pKnow[l].position[m].nConfirmation >= CONFIRMED) )
612             && (pKnow[l].position[m].alert == 0) /*EE (m!=0)*/ && (minTime >=
613                 6.0) ){
614             if(mLeft == -1){
615                 mLeft = m;
616             }
617             else if(mRight == -1){
618                 mRight = m;
619             }
620             if(mRight != -1){
621                 //check knetic envelope
622                 auxSqrt = (
623                     ((pKnow[l].position[mLeft].lLat
624                     - pKnow[l].position[mRight].lLat) *
625                     (pKnow[l].position[mLeft].lLat
626                     - pKnow[l].position[mRight].lLat)) +
627                     ((pKnow[l].position[mLeft].lLon
628                     - pKnow[l].position[mRight].lLon) *
629                     (pKnow[l].position[mLeft].lLon
630                     - pKnow[l].position[mRight].lLon))
631                 );
632                 speed = sqrt(auxSqrt) / abs(pKnow[l].position[mRight].lTime - pKnow[l].
633                     position[mLeft].lTime);
634                 if(speed > pKnow[l].maxSpeed){
635                     ////check if embarked
636                     lAux = 0;

```

```

635     mayBeEmbarked = 0;
636     while( (pKnow[lAux].id != 0) && (lAux<NONOD) ){
637         mAux=0;
638         while( (pKnow[lAux].position[mAux].nConfirmation != -1) && (mAux<
            TICKS) ){
639             if( (lAux != 1) && ( (pKnow[lAux].position[mAux].nConfirmation >=
                CONFIRMED) || ( (pKnow[lAux].type == 3) && (pKnow[lAux].position
                [mAux].nConfirmation >= 1)) ) ){
640                 if( abs(pKnow[lAux].position[mAux].lTime - pKnow[l].position[mLeft
                    ].lTime ) < 4.0){
641                     if( (pKnow[lAux].position[mAux].lLat > (pKnow[l].position[mLeft].
                        lLat - 50 * LATPREC) ) &&
642                         (pKnow[lAux].position[mAux].lLat < (pKnow[l].position[
                            mLeft].lLat + 50 * LATPREC) ) &&
643                         (pKnow[lAux].position[mAux].lLon > (pKnow[l].position[mLeft].
                            lLon - 50 * LONPREC) ) &&
644                         (pKnow[lAux].position[mAux].lLon < (pKnow[l].position[mLeft].
                            lLon + 50 * LONPREC) ) ){
645                         if(pKnow[lAux].maxSpeed >= speed){
646                             mayBeEmbarked++;
647                         }
648                     }
649
650                 }
651             }
652             mAux++;
653         }
654         lAux++;
655     }
656     if(mayBeEmbarked == 0){
657         pKnow[l].position[mLeft].alert = 2;

```

```

658         fim = time(NULL);
659         ++alertCounter;
660
661         //issues an alert
662         bzero(buf,TAMAX);
663         sprintf(buf, "Time: \t%f\n",difftime(fim,inicio));
664         fwrite(buf,strlen(buf),1,fileAlert);
665         bzero(buf,TAMAX);
666         sprintf(buf, "ID: \t%i\n",pKnow[l].id);
667         fwrite(buf,strlen(buf),1,fileAlert);
668         bzero(buf,TAMAX);
669         sprintf(buf, "Addr: \t%s\n\n",pKnow[l].hostAddr);
670         fwrite(buf,strlen(buf),1,fileAlert);
671     }
672 }
673 //update aux variables
674     mLeft = mRight;
675     mRight = -1;
676 }
677 }
678     m++;
679 }
680     l++;
681     mLeft = -1;
682     mRight = -1;
683 }
684 //end of Check #02
685
686     i++;
687 }
688 }

```

```

689
690
691  //Although there is no more broadcast position
692  //may there exist extra procedures to be done
693  fclose(file);
694
695  //saves Alert information on several files
696
697  //File saidaAlert
698  bzero(buf,TAMAX);
699  sprintf(buf, "Number of alerts issued during the simulation: \t%i\n\n",
          alertCounter);
700  fwrite(buf,strlen(buf),1,fileAlert);
701  fclose(fileAlert);
702
703  bzero(buf,TAMAX);
704  sprintf(buf, "%i\n\n",alertCounter);
705  fwrite(buf,strlen(buf),1,fileAlertLogN);
706  fclose(fileAlertLogN);
707
708  //File saidaI
709  //File name
710  strcpy(fileName, "saidaI-");
711  strcat(fileName, argv[5]);
712
713  //Opens file
714  if( (file = fopen(fileName,"wb")) == NULL ){
715      printf("Problem writting to file: %s\n", fileName);
716      return 1;
717  }
718

```

```

719  //Saves data to file
720  rewind( file );
721  bzero( buf, TAMAX );
722  sprintf( buf, "Packages (sent): %lu\n", pkgSent );
723  fwrite( buf, strlen( buf ), 1, file );
724  bzero( buf, TAMAX );
725  sprintf( buf, "Bytes (sent): %lu\n", bytesSent );
726  fwrite( buf, strlen( buf ), 1, file );
727  fclose( file );
728
729  //File saidaIN
730  //File name
731  strcpy( fileName, "saidaIP-" );
732  strcat( fileName, argv[5] );
733  strcat( fileName, ".txt" );
734
735  //Opens file
736  if( ( file = fopen( fileName, "wb" ) ) == NULL ){
737      printf( "Problem writing file: %s\n", fileName );
738      return 1;
739  }
740
741  //Saves data to file
742  rewind( file );
743  bzero( buf, TAMAX );
744  sprintf( buf, "%lu\n", pkgSent );
745  fwrite( buf, strlen( buf ), 1, file );
746  fclose( file );
747
748  return( 0 );
749 }

```

Appendix E: Background and Position Announcement Traffic Generators

The file <mgenSimScript-sample.mgn> is used to generate background flows of just one node for a sample scenario of eight nodes.

```
1 ##### flow per ip_dst: packages of 300 bytes @ 2400 bps
2 #0.0 ON 1 UDP DST 10.0.0.1/33001 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
   5.0]
3 #0.0 ON 2 UDP DST 10.0.0.1/33001 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
   5.0]
4 0.0 ON 3 UDP DST 10.0.0.2/33002 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
5 0.0 ON 4 UDP DST 10.0.0.2/33002 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
6 0.0 ON 5 UDP DST 10.0.0.3/33003 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
7 0.0 ON 6 UDP DST 10.0.0.3/33003 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
8 0.0 ON 7 UDP DST 10.0.0.4/33004 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
9 0.0 ON 8 UDP DST 10.0.0.4/33004 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
10 0.0 ON 9 UDP DST 10.0.0.5/33005 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP 5.0]
11 0.0 ON 10 UDP DST 10.0.0.5/33005 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
12 0.0 ON 11 UDP DST 10.0.0.6/33006 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
13 0.0 ON 12 UDP DST 10.0.0.6/33006 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
14 0.0 ON 13 UDP DST 10.0.0.7/33007 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
15 0.0 ON 14 UDP DST 10.0.0.7/33007 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
16 0.0 ON 15 UDP DST 10.0.0.8/33008 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
```



```

17  0.0 ON 16 UDP DST 10.0.0.8/33008 BURST [RANDOM 10.0 PERIODIC [1.0 300] EXP
    5.0]
18
19  ##### flow per ip_dst: packages of 1500 bytes @ 32 Kbps
20  #0.0 ON 17 UDP DST 10.0.0.1/33001 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
21  #0.0 ON 18 UDP DST 10.0.0.1/33001 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
22  0.0 ON 19 UDP DST 10.0.0.2/33002 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
23  0.0 ON 20 UDP DST 10.0.0.2/33002 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
24  0.0 ON 21 UDP DST 10.0.0.3/33003 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
25  0.0 ON 22 UDP DST 10.0.0.3/33003 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
26  0.0 ON 23 UDP DST 10.0.0.4/33004 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
27  0.0 ON 24 UDP DST 10.0.0.4/33004 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
28  0.0 ON 25 UDP DST 10.0.0.5/33005 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
29  0.0 ON 26 UDP DST 10.0.0.5/33005 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
30  0.0 ON 27 UDP DST 10.0.0.6/33006 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
31  0.0 ON 28 UDP DST 10.0.0.6/33006 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
32  0.0 ON 29 UDP DST 10.0.0.7/33007 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]

```

```
33  0.0 ON 30 UDP DST 10.0.0.7/33007 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
34  0.0 ON 31 UDP DST 10.0.0.8/33008 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
35  0.0 ON 32 UDP DST 10.0.0.8/33008 BURST [RANDOM 20.0 PERIODIC [8.0 1024] EXP
    5.0]
```

The file <positionBroadcast.c> is the source code of an application used to generate position messages.

```
1  /* positionBroadcast */
2  #include <sys/types.h>
3  #include <sys/socket.h>
4  #include <netinet/in.h>
5  #include <arpa/inet.h>
6  #include <netdb.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <time.h>
11
12 #define TAMAX 80 //Broadcast buffer size
13 #define NONOD 20 //# of simulated nodes
14
15 void error(char *);
16 void espere(int);
17
18 int main(int argc, char *argv[]){
19     int sock, length, n;
20     struct sockaddr_in server;
21     struct hostent *hp;
22
23     char buf[TAMAX];
24
25     time_t startTime, presentTime;
26     char fileName[80];
27     char id[20]; //node id
28     char mStatus[20]; //1-> original message; 2-> copy message
29     char lineTime[20];
```

```

30  char lineLat[20];
31  char lineLon[20];
32  char msg[TAMAX];
33  int tick =0;
34  int ptick=0;
35  int l=0;
36  int m=0;
37  char type[20];
38  char hostAddr[20];
39  char thisHostAddr[20];
40
41  unsigned long pkgSent=0;
42  unsigned long bytesSent=0;
43
44
45  struct nNodes {
46      int type; // 1-> Tactical; 2-> Sensor; 3-> Hybrid
47      int id;
48      char hostAddr[20];
49  } node[NONOD];
50
51  FILE *file;
52
53  int broadcastEnable = 1;
54
55  startTime = time(NULL);
56
57  if (argc != 5) {
58      printf("Usar: fileName port nodeID logFileName\n");
59      exit(1);
60  }

```

```

61
62  //initializes node list
63  for (l=0;l<NONOD;l++){
64      node[l].type = 0;
65      node[l].id = 0;
66      bzero(node[l].hostAddr,20);
67  }
68  bzero(hostAddr,20);
69  bzero(thisHostAddr,20);
70
71  //File name
72  strcpy(fileName, argv[1]);
73
74  //Opens node file
75  if( (file = fopen(fileName,"r")) == NULL ){
76      printf("Problem reading file: %s\n", fileName);
77      return 1;
78  }
79
80  //reads node file and copy its content
81  l=0;
82  fgets(type, 19, file);
83  while ( (!feof(file)) && (l <NONOD) && (atoi(type)!=0) ) {
84      m=0;
85      node[l].type = atoi(type);
86      fgets(id, 19, file);
87      node[l].id = atoi(id);
88      fgets(node[l].hostAddr, 19, file);
89      while((node[l].hostAddr[m] != '\n')&&(m<20)){
90          m++;
91      }

```

```

92     node[l].hostAddr[m-1]='\0';
93     if( ( atoi(argv[3]) == node[l].id) ) {
94         strcpy(thisHostAddr, node[l].hostAddr);
95     }
96     fgets(type, 19, file);
97     l++;
98 }
99 fclose(file);
100
101 //initialize socket
102 sock= socket(AF_INET, SOCK_DGRAM, 0);
103 if (sock < 0) error("socket");
104
105 //Enables broadcast
106 int ret = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcastEnable, sizeof
    (broadcastEnable));
107
108 server.sin_family = AF_INET;
109
110 //Opens the position log file
111 strcpy(fileName, argv[4]);
112 if( ( file = fopen(fileName,"r")) == NULL ){
113     printf("Problem reading file: %s\n",fileName);
114     return 1;
115 }
116
117 tick=1;
118 ptick=1;
119
120 //traffic loop
121 while ( (!feof(file)) && (tick != 0) ) {

```

```

122     ptick = tick;
123     bzero(msg, TAMAX);
124     bzero(id, 20);
125     bzero(mStatus, 20);
126     bzero(lineTime, 20);
127     bzero(lineLat, 20);
128     bzero(lineLon, 20);
129     fgets(id, 19, file);
130     fgets(lineTime, 19, file);
131     fgets(lineLat, 19, file);
132     fgets(lineLon, 19, file);
133     sprintf(mStatus, "1\n");
134     tick = atoi(lineTime);
135     presentTime = time(NULL);
136     while ( difftime(presentTime, startTime) < tick) {
137         presentTime = time(NULL);
138     }
139     if ( (ptick != tick) && (!feof(file)) ){
140         strcpy(msg, id);
141         strcat(msg, mStatus);
142         strcat(msg, lineTime);
143         strcat(msg, lineLat);
144         strcat(msg, lineLon);
145         l=0;
146         while( (node[l].type != 0) && (l<NONOD) ){
147             //if test avoids a tactical node sends a message to itself
148             //a sensor node does not run this routine
149             //a hybrid node is allowed to send to itself (tactical sending to sensor)
150             if( !((node[l].type == 1)&&(strcmp(node[l].hostAddr, thisHostAddr)==0)) ){
151                 hp = gethostbyname(node[l].hostAddr);
152                 if (hp==0) error("Host nao encontrado");

```

```

153
154     bcopy( (char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);
155     server.sin_port = htons(atoi(argv[2]));
156     length=sizeof(struct sockaddr_in);
157
158     n=sendto(sock,msg,sizeof(msg),0,&server,length);
159     if(n>0){
160         bytesSent += (unsigned long) n;
161         pkgSent += 1;
162     }
163 }
164     l++;
165 }
166 }
167     else tick = 0;
168 };
169 fclose(file);
170
171
172 //saves Alert information on several files
173
174 //File saidaB
175 //File name
176 strcpy(fileName, "saidaB-");
177 strcat(fileName, argv[3]);
178
179 //Opens file
180 if( (file = fopen(fileName,"wb")) == NULL ){
181     printf("Problem writting to file: %s\n",fileName);
182     return 1;
183 }

```



```

184
185  //Saves data to file
186  rewind( file );
187  bzero( buf,TAMAX);
188  sprintf( buf, "Packages (sent): %lu\n", pkgSent);
189  fwrite( buf, strlen( buf ),1, file );
190  bzero( buf,TAMAX);
191  sprintf( buf, "Bytes (sent): %lu\n", bytesSent);
192  fwrite( buf, strlen( buf ),1, file );
193  fclose( file );
194
195  //File saidaBP
196  //File name
197  strcpy( fileName, "saidaBP-" );
198  strcat( fileName, argv[3] );
199  strcat( fileName, ".txt" );
200
201  //Opens file
202  if( ( file = fopen( fileName,"wb" ) ) == NULL ){
203      printf("Problem writting to file: %s\n",fileName);
204      return 1;
205  }
206
207  //Saves data to file
208  rewind( file );
209  bzero( buf,TAMAX);
210  sprintf( buf, "%lu\n", pkgSent);
211  fwrite( buf, strlen( buf ),1, file );
212  fclose( file );
213
214

```

```

215  //Print to stdout
216  printf("Total packages (sent): %lu\n",pkgSent);
217  printf("Total bytes (sent): %lu\n",bytesSent);
218  return 0;
219  }
220
221  void error(char *msg){
222      perror(msg);
223      exit(0);
224  }

```

The file <positionCollector.c> is the source code of an application used to collect position messages.

```
1  /* positionCollector */
2  #include <sys/types.h>
3  #include <sys/socket.h>
4  #include <netinet/in.h>
5  #include <netdb.h>
6  #include <stdio.h>
7  #include <arpa/inet.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <time.h>
11 #include <sys/ipc.h>
12 #include <sys/shm.h>
13
14 #define TAMAX 80 //Broadcast buffer size
15 #define NOMSG 2000 // # collected msg
16
17 void error(char *msg){
18     perror(msg);
19     exit(0);
20 }
21
22 int main(int argc, char *argv[]){
23     int sock, length, fromlen, n;
24     struct sockaddr_in server;
25     struct sockaddr_in from;
26     char buf[TAMAX];
27
28     FILE *file;
29     FILE *fileCP;
```

```

30  char fileName[80];
31  char fileNameCP[80];
32  float tempoSimulacao;
33  time_t inicio ,fim ,fimReal;
34
35  char firstLine[20];
36  char mStatus[20]; //1-> original message; 2-> copy message
37  char lineTime[20];
38  char lineLat[20];
39  char lineLon[20];
40  char hostAddr[16];
41  int j=0;
42  int k=0;
43  char charPosition;
44  double lTime, lLat, lLon;
45  int seqN=1;
46  struct pCollected {
47      int id;
48      int mStatus;
49      int seqN;
50      double lTime;
51      double lLat;
52      double lLon;
53      char hostAddr[16];
54  } bCol[NOMSG];
55
56  //shared memory variables
57  char c;
58  int shmid;
59  key_t key;
60  struct pCollected *shm, *s;

```

```

61
62 //number arguments test
63 if (argc < 4) {
64     fprintf(stderr, "Use: port_number total_time memory_id\n");
65     exit(0);
66 }
67
68 tempoSimulacao = (float) atoi(argv[2]);
69 key = ftok("/home/jeronymo/Downloads/virtualbox linux files/ids/aa.txt", atoi
    (argv[3]));
70 if (0 > key){
71     perror("ftok");
72 }
73 else{
74     //printf("ftok success: %lli\n", (long long int) key);
75 }
76 strcpy(fileName, "saidaC-");
77 strcat(fileName, argv[3]);
78
79 strcpy(fileNameCP, "saidaCP-");
80 strcat(fileNameCP, argv[3]);
81 strcat(fileNameCP, ".txt");
82
83 //shared memory initialization
84 if ((shm = shmget(key, NOMSG*(sizeof(struct pCollected)), IPC_CREAT | 0666)
    ) < 0) {
85     perror("shmget");
86     printf("positionColector - shmget\n");
87     exit(1);
88 }
89 if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {

```

```

90     perror("shmat");
91     exit(1);
92 }
93 s = shm;
94
95
96
97 //socket initialization
98 sock=socket(AF_INET, SOCK_DGRAM, 0);
99 if (sock < 0) error("Error openning the socket");
100
101 length = sizeof(server);
102 bzero(&server, length);
103 server.sin_family=AF_INET;
104 server.sin_addr.s_addr=INADDR_ANY;
105 server.sin_port=htons(atoi(argv[1]));
106 if (bind(sock, (struct sockaddr *)&server, length)<0)
107     error("ERROR: binding");
108 fromlen = sizeof(struct sockaddr_in);
109
110 //opens file
111 if( (file = fopen(fileName, "wb")) == NULL ){
112     printf("Problem writting to file: %s\n", fileName);
113     return 1;
114 }
115
116 //opens file
117 if( (fileCP = fopen(fileNameCP, "wb")) == NULL ){
118     printf("Problem writting to file: %s\n", fileNameCP);
119     return 1;
120 }

```

```

121
122  unsigned long i=0;
123  unsigned long bytesReceived =0;
124
125  bzero(hostAddr,16);
126
127  for (i=0;i<NOMSG;i++){
128      bCol[i].id = 0;
129      bCol[i].mStatus = 0;
130      bCol[i].seqN = 0;
131      bCol[i].lTime = 0.0;
132      bCol[i].lLat = 0.0;
133      bCol[i].lLon = 0.0;
134      bzero(bCol[i].hostAddr,16);
135      s[i].id = 0;
136      s[i].mStatus = 0;
137      s[i].seqN = 0;
138      s[i].lTime = 0.0;
139      s[i].lLat = 0.0;
140      s[i].lLon = 0.0;
141      bzero(s[i].hostAddr,16);
142  }
143
144  i=0;
145  inicio = time(NULL);
146  fim = inicio;
147
148  while (difftime(fim,inicio) < tempoSimulacao) {
149      n = recvfrom(sock,buf,TAMAX,0,(struct sockaddr *)&from,&fromlen);
150
151      if (n < 0) error("recvfrom");

```

```

152
153     if (i==0)
154         inicio = time(NULL);
155     fim = time(NULL);
156     if(difftime(fim, inicio) < tempoSimulacao){
157         bytesReceived += n;
158         i++;
159         fim = time(NULL);
160         fimReal = time(NULL);
161         bzero(hostAddr, 16);
162         strcpy(hostAddr, inet_ntoa(from.sin_addr));
163         hostAddr[15] = '\n';
164         bzero(firstLine, 20);
165         bzero(mStatus, 20);
166         bzero(lineTime, 20);
167         bzero(lineLat, 20);
168         bzero(lineLon, 20);
169         j=0;
170         k=0;
171         charPosition = buf[j];
172
173         while(charPosition != '\n'){
174             firstLine[k] = buf[j];
175             j++;
176             k++;
177             charPosition = buf[j];
178
179         }
180         firstLine[k] = '\n';
181         j++;
182         k=0;

```



```

183     charPosition = buf[j];
184     while(charPosition != '\n'){
185         mStatus[k] = buf[j];
186         j++;
187         k++;
188         charPosition = buf[j];
189
190     }
191     mStatus[k] = '\n';
192     j++;
193     k=0;
194     charPosition = buf[j];
195     while(charPosition != '\n'){
196         lineTime[k] = buf[j];
197         j++;
198         k++;
199         charPosition = buf[j];
200
201     }
202     lineTime[k] = '\n';
203     j++;
204     k=0;
205     charPosition = buf[j];
206     while(charPosition != '\n'){
207         lineLat[k] = buf[j];
208         j++;
209         k++;
210         charPosition = buf[j];
211
212     }
213     lineLat[k] = '\n';

```

```

214     j++;
215     k=0;
216     charPosition = buf[j];
217     while(charPosition != '\n'){
218         lineLon[k] = buf[j];
219         j++;
220         k++;
221         charPosition = buf[j];
222
223     }
224     lineLon[k] = '\n';
225     lTime = atof(lineTime);
226     lLat = atof(lineLat);
227     lLon = atof(lineLon);
228
229     bCol[i].id = atoi(firstLine);
230     bCol[i].mStatus = atoi(mStatus);
231     bCol[i].seqN = seqN;
232     bCol[i].lTime = atof(lineTime);
233     bCol[i].lLat = atof(lineLat);
234     bCol[i].lLon = atof(lineLon);
235     strcpy(bCol[i].hostAddr, hostAddr);
236
237     s[i].id = atoi(firstLine);
238     s[i].mStatus = atoi(mStatus);
239     s[i].seqN = seqN;
240     s[i].lTime = atof(lineTime);
241     s[i].lLat = atof(lineLat);
242     s[i].lLon = atof(lineLon);
243     strcpy(s[i].hostAddr, hostAddr);
244

```

```

245     seqN++;
246 }
247 }
248 s[i+1].seqN = -1;
249 rewind(file);
250 bzero(buf,TAMAX);
251 sprintf(buf, "Packages (received): %lu\n", i);
252 fwrite(buf,strlen(buf),1,file);
253 bzero(buf,TAMAX);
254 sprintf(buf, "Bytes (received): %lu\n", bytesReceived);
255 fwrite(buf,strlen(buf),1,file);
256 fclose(file);
257
258 rewind(fileCP);
259 bzero(buf,TAMAX);
260 sprintf(buf, "%lu\n", i);
261 fwrite(buf,strlen(buf),1,fileCP);
262 fclose(fileCP);
263
264 printf("Total packages (received): %lu\n",i);
265 printf("Total bytes (received): %lu\n",bytesReceived);
266 return 0;
267 }

```

Appendix F: CMIDS Multilateration Error Estimation

Source Code

```
1  % This routine calculates the multilateration error of CMIDS
2  % It uses part of the work of Hamid Ramezani.
3
4  % Global Setting
5  N = 5; % number of sensors
6  M = 10; % number of tactical nodes
7  numberOfSamples = 90; % number of samples
8
9  % distance dependent err (standard deviation of the noise normalized to
    distance)
10 distMeasurementErrRatio = 0.05; % it means that the accuracy of distance
    measurement is 95%
11
    % for instance the inaccuracy of a 1m
    measured distance
12
    % is around .05 meter.
13
14  networkSize = 2000;
15
16  clc; % Clear the command window.
17  workspace; % Make sure the workspace panel is showing.
18  format longg;
19  format compact;
20
21  % Define a starting folder.
22  start_path = fullfile('./data/');
23  topLevelFolder = start_path;
24  if topLevelFolder == 0
```

```

25  return;
26  end
27  % Get list of all subfolders.
28  allSubFolders = genpath(topLevelFolder);
29  % Parse into a cell array.
30  remain = allSubFolders;
31  listOfFolderNames = {};
32  while true
33      [singleSubFolder, remain] = strtok(remain, ':');
34      if isempty(singleSubFolder)
35          break;
36      end
37      listOfFolderNames = [listOfFolderNames singleSubFolder];
38  end
39  numberOfFolders = length(listOfFolderNames);
40
41  simRounds = numberOfFolders - 1; % number of simulation rounds
42  mError = zeros(simRounds,1);
43
44  % Process all files in those folders.
45  for k = 2 : numberOfFolders
46      % Get this folder and print it out.
47      thisFolder = listOfFolderNames{k};
48      fprintf('Processing folder %s\n', thisFolder);
49
50      % sensor N1
51      current_file = fullfile(thisFolder, 'LogS-1.txt');
52      fileID = fopen(current_file, 'r');
53      formatSpec = '%f';
54      sN1 = fscanf(fileID, formatSpec);
55      fclose(fileID);

```

```

56     szDim1 = size(sN1,1)/4;
57     sensorN1 = zeros(szDim1,2);
58
59     for n = 1:1:szDim1
60         sensorN1(n,1) = sN1(4*n-1);
61         sensorN1(n,2) = sN1(4*n);
62     end
63
64     % sensor N2
65     current_file = fullfile(thisFolder,'LogS-2.txt');
66     fileID = fopen(current_file,'r');
67     formatSpec = '%f';
68     sN2 = fscanf(fileID,formatSpec);
69     fclose(fileID);
70     szDim2 = size(sN2,1)/4;
71     sensorN2 = zeros(szDim2,2);
72
73     for n = 1:1:szDim2
74         sensorN2(n,1) = sN2(4*n-1);
75         sensorN2(n,2) = sN2(4*n);
76     end
77
78     % sensor N3
79     current_file = fullfile(thisFolder,'LogS-3.txt');
80     fileID = fopen(current_file,'r');
81     formatSpec = '%f';
82     sN3 = fscanf(fileID,formatSpec);
83     fclose(fileID);
84     szDim3 = size(sN3,1)/4;
85     sensorN3 = zeros(szDim3,2);
86     for n = 1:1:szDim3

```

```

87         sensorN3(n,1) = sN3(4*n-1);
88         sensorN3(n,2) = sN3(4*n);
89     end
90
91     % sensor N4
92     current_file = fullfile(thisFolder, 'LogS-4.txt');
93     fileID = fopen(current_file, 'r');
94     formatSpec = '%f';
95     sN4 = fscanf(fileID, formatSpec);
96     fclose(fileID);
97     szDim4 = size(sN4,1)/4;
98     sensorN4 = zeros(szDim4,2);
99     for n = 1:1:szDim4
100         sensorN4(n,1) = sN4(4*n-1);
101         sensorN4(n,2) = sN4(4*n);
102     end
103
104     % sensor N5
105     current_file = fullfile(thisFolder, 'LogS-5.txt');
106     fileID = fopen(current_file, 'r');
107     formatSpec = '%f';
108     sN5 = fscanf(fileID, formatSpec);
109     fclose(fileID);
110     szDim5 = size(sN5,1)/4;
111     sensorN5 = zeros(szDim5,2);
112     for n = 1:1:szDim5
113         sensorN5(n,1) = sN5(4*n-1);
114         sensorN5(n,2) = sN5(4*n);
115     end
116
117

```

```

118
119     % Tactical nodes
120
121     % tactical node 1
122     current_file = fullfile(thisFolder, 'Log-1.txt');
123     fileID = fopen(current_file, 'r');
124     formatSpec = '%f';
125     tN1 = fscanf(fileID, formatSpec);
126     fclose(fileID);
127     szDim = size(tN1, 1) / 4;
128     tacticalN1 = zeros(szDim, 2);
129     for n = 1:1:szDim
130         tacticalN1(n, 1) = tN1(4*n-1);
131         tacticalN1(n, 2) = tN1(4*n);
132     end
133
134     % tactical node 2
135     current_file = fullfile(thisFolder, 'Log-2.txt');
136     fileID = fopen(current_file, 'r');
137     formatSpec = '%f';
138     tN2 = fscanf(fileID, formatSpec);
139     fclose(fileID);
140     szDim = size(tN2, 1) / 4;
141     tacticalN2 = zeros(szDim, 2);
142     for n = 1:1:szDim
143         tacticalN2(n, 1) = tN2(4*n-1);
144         tacticalN2(n, 2) = tN2(4*n);
145     end
146
147     % tactical node 3
148     current_file = fullfile(thisFolder, 'Log-3.txt');

```



```

149     fileID = fopen(current_file , 'r ');
150     formatSpec = '%f';
151     tN3 = fscanf(fileID , formatSpec);
152     fclose(fileID);
153     szDim = size(tN3,1)/4;
154     tacticalN3 = zeros(szDim,2);
155     for n = 1:1:szDim
156         tacticalN3(n,1) = tN3(4*n-1);
157         tacticalN3(n,2) = tN3(4*n);
158     end
159
160     % tactical node 4
161     current_file = fullfile(thisFolder , 'Log-4.txt ');
162     fileID = fopen(current_file , 'r ');
163     formatSpec = '%f';
164     tN4 = fscanf(fileID , formatSpec);
165     fclose(fileID);
166     szDim = size(tN4,1)/4;
167     tacticalN4 = zeros(szDim,2);
168     for n = 1:1:szDim
169         tacticalN4(n,1) = tN4(4*n-1);
170         tacticalN4(n,2) = tN4(4*n);
171     end
172
173     % tactical node 5
174     current_file = fullfile(thisFolder , 'Log-5.txt ');
175     fileID = fopen(current_file , 'r ');
176     formatSpec = '%f';
177     tN5 = fscanf(fileID , formatSpec);
178     fclose(fileID);
179     szDim = size(tN5,1)/4;

```

```

180     tacticalN5 = zeros(szDim,2);
181     for n = 1:1:szDim
182         tacticalN5(n,1) = tN5(4*n-1);
183         tacticalN5(n,2) = tN5(4*n);
184     end
185
186     % tactical node 6
187     current_file = fullfile(thisFolder,'Log-6.txt');
188     fileID = fopen(current_file,'r');
189     formatSpec = '%f';
190     tN6 = fscanf(fileID,formatSpec);
191     fclose(fileID);
192     szDim = size(tN6,1)/4;
193     tacticalN6 = zeros(szDim,2);
194     for n = 1:1:szDim
195         tacticalN6(n,1) = tN6(4*n-1);
196         tacticalN6(n,2) = tN6(4*n);
197     end
198
199     % tactical node 7
200     current_file = fullfile(thisFolder,'Log-7.txt');
201     fileID = fopen(current_file,'r');
202     formatSpec = '%f';
203     tN7 = fscanf(fileID,formatSpec);
204     fclose(fileID);
205     szDim = size(tN7,1)/4;
206     tacticalN7 = zeros(szDim,2);
207     for n = 1:1:szDim
208         tacticalN7(n,1) = tN7(4*n-1);
209         tacticalN7(n,2) = tN7(4*n);
210     end

```

```

211
212     % tactical node 8
213     current_file = fullfile(thisFolder, 'Log-8.txt');
214     fileID = fopen(current_file, 'r');
215     formatSpec = '%f';
216     tN8 = fscanf(fileID, formatSpec);
217     fclose(fileID);
218     szDim = size(tN8,1)/4;
219     tacticalN8 = zeros(szDim,2);
220     for n = 1:1:szDim
221         tacticalN8(n,1) = tN8(4*n-1);
222         tacticalN8(n,2) = tN8(4*n);
223     end
224
225     % tactical node 9
226     current_file = fullfile(thisFolder, 'Log-9.txt');
227     fileID = fopen(current_file, 'r');
228     formatSpec = '%f';
229     tN9 = fscanf(fileID, formatSpec);
230     fclose(fileID);
231     szDim = size(tN9,1)/4;
232     tacticalN9 = zeros(szDim,2);
233     for n = 1:1:szDim
234         tacticalN9(n,1) = tN9(4*n-1);
235         tacticalN9(n,2) = tN9(4*n);
236     end
237
238     % tactical node 10
239     current_file = fullfile(thisFolder, 'Log-10.txt');
240     fileID = fopen(current_file, 'r');
241     formatSpec = '%f';

```

```

242     tN10 = fscanf(fileID ,formatSpec);
243     fclose(fileID);
244     szDim = size(tN10,1)/4;
245     tacticalN10 = zeros(szDim,2);
246     for n = 1:1:szDim
247         tacticalN10(n,1) = tN10(4*n-1);
248         tacticalN10(n,2) = tN10(4*n);
249     end
250
251     totalErr = zeros(numberOfSamples,1);
252
253     % main loop
254     for iter = 1:1:numberOfSamples
255
256         % location of sensors and tactical nodes
257         %iter = szDim;
258         sensorLoc = [sensorN1(iter,1)    sensorN1(iter,2);
259                     sensorN2(iter,1)    sensorN2(iter,2);
260                     sensorN3(iter,1)    sensorN3(iter,2);
261                     sensorN4(iter,1)    sensorN4(iter,2);
262                     sensorN5(iter,1)    sensorN5(iter,2)];
263
264         mobileLoc = [
265                     tacticalN1(iter,1)    tacticalN1(iter,2);
266                     tacticalN2(iter,1)    tacticalN2(iter,2);
267                     tacticalN3(iter,1)    tacticalN3(iter,2);
268                     tacticalN4(iter,1)    tacticalN4(iter,2);
269                     tacticalN5(iter,1)    tacticalN5(iter,2);
270                     tacticalN6(iter,1)    tacticalN6(iter,2);
271                     tacticalN7(iter,1)    tacticalN7(iter,2);
272                     tacticalN8(iter,1)    tacticalN8(iter,2);

```

```

273             tacticalN9(iter,1)    tacticalN9(iter,2);
274             tacticalN10(iter,1)    tactical10(iter,2);
275         ];
276
277
278         % Computing the Euclidian distances
279         distance = zeros(N,M);
280         for m = 1 : M
281             for n = 1 : N
282                 distance(n,m) = sqrt( (sensorLoc(n,1)-mobileLoc(m,1)).^2 + ...
283                                         (sensorLoc(n,2)-mobileLoc(m,2)).^2
284                                         );
285             end
286         end
287
288         % noisy measurements
289         distanceNoisy = distance + distance.*distMeasurementErrRatio.*(rand(N,M)
290                                     -1/2);
291
292
293         % using gussian newton to solve the problem
294         % (http://en.wikipedia.org/wiki/Gauss%E2%80%93Newton\_algorithm)
295
296         numOfIteration = 5;
297
298         % Initial guess (random location)
299         mobileLocEst = networkSize*rand(M,2);
300
301         % repeation
302         for m = 1 : M
303             for i = 1 : numOfIteration
304                 % computing the esimated distances

```

```

301         distanceEst = sqrt(sum( (sensorLoc - repmat(mobileLocEst(m,:),N
                                ,1)).^2 , 2));
302         % computing the derivatives
303         % d0 = sqrt( (x-x0)^2 + (y-y0)^2 )
304         % derivatives -> d(d0)/dx = (x-x0)/d0
305         % derivatives -> d(d0)/dy = (y-y0)/d0
306         distanceDrv = [(mobileLocEst(m,1)-sensorLoc(:,1))./distanceEst
                        ... % x-coordinate
307                        (mobileLocEst(m,2)-sensorLoc(:,2))./distanceEst];
                        % y-coordinate
308         % delta
309         delta = - (distanceDrv.'*distanceDrv)^-1*distanceDrv.' * (
                        distanceEst - distanceNoisy(:,m));
310         % Updating the estimation
311         mobileLocEst(m,:) = mobileLocEst(m,:) + delta.';
312     end
313 end
314
315     % Compute the Root Mean Squared Error
316     Err = mean(sqrt(sum((mobileLocEst-mobileLoc).^2)));
317
318     totalErr(iter) = Err;
319
320 end
321     mError(k-1) = median(totalErr);
322 end
323 mError = mError';
324 mError
325
326 % Plot graph 1
327     fl = figure(1);

```

```

328     clf
329     round = 1:1:simRounds;
330     plot(round, mError, '-');
331     grid on;
332     hold on;
333     title([ 'Multilateration Error per Simulation Round' ]);
334     xlabel( 'Simulation Round' );
335     ylabel( 'Multilateration Error (m)' );
336     dim = [.7 .6 .3 .3];
337     str = sprintf( 'Mean = %8.4f\nMax = %8.4f\nMin = %8.4f ',mean(mError),max(
        mError),min(mError));
338     annotation( 'textbox',dim, 'String',str, 'FitBoxToText', 'on' );
339     axis([0 simRounds 0.9*min(mError) 1.1*max(mError)]);
340
341
342 %Plot graph 2
343     f2 = figure(2);
344     clf
345     round = 1:1:simRounds;
346     plot(round, mError, '-');
347     grid on;
348     hold on;
349     title([ 'Multilateration Error per Simulation Round' ]);
350     xlabel( 'Simulation Round' );
351     ylabel( 'Multilateration Error (m)' );
352     dim = [.7 .1 .3 .3];
353     str = sprintf( 'Mean = %8.4f\nMax = %8.4f\nMin = %8.4f ',mean(mError),max(
        mError),min(mError));
354     annotation( 'textbox',dim, 'String',str, 'FitBoxToText', 'on' );
355     axis([0 simRounds 0 1.1*max(mError)]);

```

References

- [1] J. Kaur, S. Saxena, and M. A. Sayeed, “Securing mobile agent’s information in ad-hoc network,” in *Confluence The Next Generation Information Technology Summit, 2014 5th International Conference*. IEEE, 2014, pp. 442–446.
- [2] A. Nadeem and M. P. Howarth, “A Survey of MANET Intrusion Detection & Prevention Approaches for Network Layer Attacks,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2027–2045, 2013.
- [3] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, May 2002.
- [4] International Council of E-Commerce Consultants, Ed., *Ethical hacking and counter-measures: Attack Phases*. Clifton Park, NY: Course Technology, CENGAGE Learning, 2010.
- [5] D. I. S. Agency, Ed., *Cloud Computing Security Requirements Guide*, 1st ed. Department of Defense, Jan. 2015.
- [6] M. Elboukhari, M. Azizi, and A. Azizi, “Intrusion Detection Systems in mobile ad hoc networks: A survey,” in *Codes, Cryptography and Communication Systems (WCCCS), 2014 5th Workshop on*. IEEE, 2014, pp. 136–141.
- [7] I. Butun, S. D. Morgera, and R. Sankar, “A Survey of Intrusion Detection Systems in Wireless Sensor Networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 266–282, 2014.
- [8] O. Can and O. K. Sahingoz, “A survey of intrusion detection systems in wireless sensor networks,” in *Modeling, Simulation, and Applied Optimization (ICMSAO), 2015 6th International Conference on*. IEEE, 2015, pp. 1–6.
- [9] F. Sabahi and A. Movaghar, “Intrusion Detection: A Survey.” IEEE, 2008, pp. 23–26.
- [10] L. Dali, A. Bentajer, E. Abdelmajid, K. Abouelmehdi, H. Elsayed, E. Fatiha, and B. Abderahim, “A survey of intrusion detection system.” IEEE, Mar. 2015, pp. 1–6.
- [11] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, “Taxonomy and Survey of Collaborative Intrusion Detection,” *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–33, May 2015.
- [12] S. Marti, T. J. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” in *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM, 2000, pp. 255–265.

- [13] E. M. Shakshuki, N. Kang, and T. R. Sheltami, "EAACK - A Secure Intrusion-Detection System for MANETs," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 3, pp. 1089–1098, Mar. 2013.
- [14] J. F. C. Joseph, A. Das, B.-C. Seet, and B.-S. Lee, "CRADS: integrated cross layer approach for detecting routing attacks in MANETs," in *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*. IEEE, 2008, pp. 1525–1530.
- [15] R. Chaki and N. Chaki, "IDSX: a cluster based collaborative intrusion detection algorithm for mobile ad-hoc network," in *Computer Information Systems and Industrial Management Applications, 2007. CISIM'07. 6th International Conference on*. IEEE, 2007, pp. 179–184.
- [16] J. Hortelano, J. C. Ruiz, and P. Manzoni, "Evaluating the usefulness of watchdogs for intrusion detection in vanets," in *Communications Workshops (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.
- [17] DoD, Ed., *Modeling and Simulation Development Best Practices*, Jul. 2010, no. NSAD-R-2010-037.
- [18] —, *VV&A Recommended Practice Guide Core Document*, Jan. 2011.
- [19] H. A. Simon, *Qualitative Simulation Modeling and Analysis*, 1st ed., P. A. Fishwick and P. A. Luker, Eds. New York: Springer, Jan. 1991.
- [20] A. Borshchev, *The Big Book of Simulation Modeling: Multimethod Modeling with Anylogic 6*. Lisle, IL: AnyLogic North America, Jun. 2013.
- [21] V. MÄK, "VR-Forces: Computer Generated Forces and Simulator Development," 2015. [Online]. Available: <http://www.mak.com/products/simulate/vr-forces>
- [22] N. R. Laboratory and B. R. T. division, "Common Open Research Emulator (CORE)," 2015. [Online]. Available: <http://www.nrl.navy.mil/itd/ncs/products/core>
- [23] N. R. Laboratory, "Extendable Mobile Ad-hoc Network Emulator (EMANE)," 2015. [Online]. Available: <http://www.nrl.navy.mil/itd/ncs/products/emane>
- [24] O. Dictionaries, *Paperback Oxford English Dictionary*, 7th ed. Oxford: Oxford University Press, Nov. 2013.
- [25] IETF, "Proceedings of the Thirty-Ninth Internet Engineering Task Force," 1997. [Online]. Available: <https://www.ietf.org/proceedings/39/>
- [26] D. Magnoli, *Historia das Guerras*, 3rd ed. Sao Paulo: Editora Contexto, 2006.
- [27] G. Goebel. (2015) The Wizard War: WW2 & The Origins of Radar. [Online]. Available: <http://www.vectorsite.net/ttwiz.html>
- [28] E. Whitman. (2015) SOSUS: The Secret Weapon of Undersea Surveillance. [Online]. Available: http://www.public.navy.mil/subfor/underseawarfaremagazine/Issues/Archives/issue_25/sosus.htm

- [29] Z. Ruan, E. C.-H. Ngai, and J. Liu, "Wireless sensor network deployment in mobile phones assisted environment," in *Quality of Service (IWQoS), 2010 18th International Workshop on*. IEEE, 2010, pp. 1–9.
- [30] R. K. Schmidt, T. Leinmüller, E. Schoch, A. Held, and G. Schfer, "Vehicle behavior analysis to enhance security in vanets," in *Proceedings of the 4th IEEE Vehicle-to-Vehicle Communications Workshop (V2VCOM2008)*, 2008.
- [31] E. de Freitas, T. Heimfarth, A. Vinel, F. Wagner, C. Pereira, and T. Larsson, "Cooperation among Wirelessly Connected Static and Mobile Sensor Nodes for Surveillance Applications," *Sensors*, vol. 13, no. 10, pp. 12 903–12 928, Sep. 2013.
- [32] J. Zhang and V. Varadharajan, "Wireless sensor network key management survey and taxonomy," *Journal of Network and Computer Applications*, vol. 33, no. 2, pp. 63–75, Mar. 2010.
- [33] M. Li, Z. Li, and A. V. Vasilakos, "A Survey on Topology Control in Wireless Sensor Networks: Taxonomy, Comparative Study, and Open Issues," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2538–2557, Dec. 2013.
- [34] G. Acs and L. Buttyan, "A taxonomy of routing protocols for wireless sensor networks," *BUTE Telecommunication department*, 2007.
- [35] M. E. Patil and R. D. More, "Survey of intrusion detection system in multitier web application," *Int. J. Emerg. Technol. Adv. Eng*, vol. 2, no. 10, pp. 570–575, 2012.
- [36] O. Kachirski and R. Guha, "Effective intrusion detection using multiple sensors in wireless ad hoc networks," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. IEEE, 2003, pp. 8–pp.
- [37] A. P. R. da Silva, M. H. Martins, B. P. Rocha, A. A. Loureiro, L. B. Ruiz, and H. C. Wong, "Decentralized intrusion detection in wireless sensor networks," in *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*. ACM, 2005, pp. 16–23.
- [38] B. Sun, L. Osborne, Y. Xiao, and S. Guizani, "Intrusion detection techniques in mobile ad hoc and wireless sensor networks," *Wireless Communications, IEEE*, vol. 14, no. 5, pp. 56–63, 2007.
- [39] M. Sobirey, "Michael Sobirey's Intrusion Detection Systems page," 2000. [Online]. Available: <http://isl.cse.sc.edu/mirrorSobireys.shtml>
- [40] J. T. Chiang, J. J. Haas, and Y.-C. Hu, "Secure and precise location verification using distance bounding and simultaneous multilateration," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 181–192.
- [41] T. Leinmüller, C. Maihöfer, E. Schoch, and F. Kargl, "Improved security in geographic ad hoc routing through autonomous position verification," in *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*. ACM, 2006, pp. 57–66.

- [42] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 6, pp. 1067–1080, Nov. 2007.
- [43] M. Buehrer and S. Venkatesh, *Fundamentals of Time-of-Arrival-Based Position Locations*, in *Handbook of Position Location: Theory, Practice, and Advances*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011.
- [44] L. Xiao, W. Jin-kuan, and W. Yun, "A novel localization algorithm based on received signal strength for mobile wireless sensor networks," in *Microwave and Millimeter Wave Technology, 2008. ICMMT 2008. International Conference on*, vol. 1. IEEE, 2008, pp. 92–95.
- [45] O. Friedewald and M. Lange, "Localization methods by using phase of arrival in systems with passive RFID Tag's," in *Smart SysTech 2016; European Conference on Smart Objects, Systems and Technologies; Proceedings of*. VDE, 2016, pp. 1–6.
- [46] E. C. L. Chan and G. Baciuc, *Introduction to Wireless Localization: With iPhone SDK Examples*. John Wiley & Sons, May 2012.
- [47] M. Monteiro, A. Barreto, R. Division, T. Kacem, J. Carvalho, D. Wijesekera, and P. Costa, "Detecting malicious ADS-B broadcasts using wide area multilateration," in *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*. IEEE, 2015, pp. 4A3–1.
- [48] A. Andersen, "Signal Emitter Positioning Using Multilateration," Jan. 2012. [Online]. Available: <http://blog.andersen.im/2012/07/signal-emitter-positioning-using-multilateration/>
- [49] W. Neven, T. Quilter, R. Weedon, and R. Hogendoorn, "Wide Area Multilateration, Report on EATMP TRS 131/04, Version 1.1," Eurocontrol, Tech. Rep., 2005.
- [50] F. Niles, R. Conker, B. El-Arini, D. OLaughlin, and D. Baraban, "Wide Area Multilateration for Alternate Position, Navigation, and Timing (APNT)," FAA, Tech. Rep., 2012.
- [51] B. Selim and C. Y. Yeun, "Key management for the MANET: A survey," in *2015 International Conference on Information and Communication Technology Research (ICT-TRC)*, May 2015, pp. 326–329.
- [52] A. J. Menezes, P. C. v. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 1st ed. Boca Raton: CRC Press, Oct. 1996.
- [53] S. H. Talawar, S. Maity, and R. Hansdah, "Secure Routing with an Integrated Localized Key Management Protocol in MANETs." IEEE, May 2014, pp. 605–612.
- [54] R. A. Mollin, *RSA and Public-Key Cryptography*, 1st ed. Boca Raton, Fla: Chapman and Hall/CRC, Nov. 2002.
- [55] V. Cortier, S. Kremer, and B. Warinschi, "A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems," *Journal of Automated Reasoning*, vol. 46, no. 3-4, pp. 225–259, Apr. 2011.

- [56] P. Lafourcade, “Models and analysis of security protocols 1st Semester 2010-2011 Tools Lecture 9,” 2010.
- [57] B. Blanchet, “A Computationally Sound Mechanized Prover for Security Protocols,” *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 193–207, Oct. 2008.
- [58] D. Dolev and A. C. Yao, “On the security of public key protocols,” *Information Theory, IEEE Transactions on*, vol. 29, no. 2, pp. 198–208, Mar. 1983.
- [59] M. Abadi and P. Rogaway, “Reconciling two views of cryptography (the computational soundness of formal encryption)*,” *Journal of cryptology*, vol. 15, no. 2, pp. 103–127, 2002.
- [60] B. Blanchet, “Computationally sound mechanized proofs of correspondence assertions,” in *Computer Security Foundations Symposium, 2007. CSF’07. 20th IEEE*. IEEE, 2007, pp. 97–111.
- [61] S. Matsuo, K. Miyazaki, A. Otsuka, and D. Basin, “How to Evaluate the Security of Real-Life Cryptographic Protocols?” in *Financial Cryptography and Data Security*. Springer, 2010, pp. 182–194.
- [62] JTC-1.SC-27_Secretariat, *ISO/IEC 29128:2011 - Information technology – Security techniques – Verification of cryptographic protocols*, Dec. 2015.
- [63] B. Blanchet, “CryptoVerif: Computationally sound mechanized prover for cryptographic protocols,” in *Dagstuhl seminar Formal Protocol Verification Applied*, 2007, p. 117.
- [64] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: a proof assistant for higher-order logic*. Springer Science & Business Media, 2002, vol. 2283.
- [65] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*, ser. Information Security and Cryptography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
- [66] H. Ramezani, “Single/Multiple Target Location,” 2015. [Online]. Available: <https://www.mathworks.com/matlabcentral/fileexchange/33792-single-target-localization/>
- [67] M. Jamshidi, *Systems of systems engineering: innovations for the 21st century*. Hoboken, NJ: Wiley, 2009.
- [68] R. Liu and W. Trappe, *Securing wireless communications at the physical layer*. New York ; London: Springer, 2010.
- [69] C. Cremers and M. Horvat, “Improving the ISO/IEC 11770 standard for key management techniques,” in *International Conference on Research in Security Standardisation*. Springer, 2014, pp. 215–235.
- [70] PUC-Rio, “Lua (the programming language),” 2016. [Online]. Available: <https://www.lua.org>

- [71] “The future is written with Qt: Cross-platform software development for embedded & desktop,” Jun. 2015. [Online]. Available: <https://www.qt.io/>
- [72] EsAO, *Manual de Organização e Emprego das Forças Armadas*. EGGCF, 2010.
- [73] A.-S. K. Pathan, Ed., *Security of Self-Organizing Networks: MANET, WSN, WMN, VANET*, 1st ed. Boca Raton: Auerbach Publications, Oct. 2010.
- [74] Y. Wang, L. Liu, B. Sun, and Y. Li, “A survey of defense mechanisms against application layer distributed denial of service attacks,” in *Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference on*. IEEE, 2015, pp. 1034–1037.
- [75] R. L. NRL and R. G. PROTEAN, “Multi-Generator (MGEN),” 2015. [Online]. Available: <https://www.nrl.navy.mil/itd/ncs/products/mgen>
- [76] M. Roesch, “SNORT,” 2016. [Online]. Available: <https://www.snort.org>
- [77] OISF, “Suricata,” 2016. [Online]. Available: <https://suricata-ids.org>
- [78] ITU-R P.838-3, “Specific attenuation model for rain for use in prediction methods,” 2005.

Biography

Jeronymo Mota Alves de Carvalho graduated from Colégio Impacto Tijuca, Rio de Janeiro, Brazil, in 1998. He received his Bachelor of Computer Engineering from Military Institute of Engineering, Brazil, in 2003 and his Master of Systems Engineering from Military Institute of Engineering in 2009. He is an officer of the Brazilian Army.